Efficient Analysis in Multimedia Databases

Dissertation im Fach Informatik
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

von

Peter Kunath

Tag der Einreichung: 24.11.2006
Tag der mündlichen Prüfung: 19.12.2006

Berichterstatter:
Prof. Dr. Hans-Peter Kriegel, Ludwig-Maximilians-Universität München
Prof. Dr. Bernhard Seeger, Philipps-Universität Marburg

Acknowledgement

While I can not name all the people who have supported and encouraged me during the past years, I want to thank those that notably helped me with the development of this thesis.

First of all, like to express my warmest thanks to my supervisor, Prof. Dr. Hans-Peter Kriegel who initiated and supported this work. I learned a lot from his long standing experience and organizational background. Then I want to thank Prof. Dr. Bernhard Seeger for the interest in my work. He was kindly willing to act as second referee for this work.

Without the inspiring, productive and supportive working environment of the database research team this work would never have been possible. Therefore I express my gratitude to my colleagues, in particular to Matthias Renz, Dr. Peer Kröger, Alexey Pryakhin, Dr. Matthias Schubert, Stefan Brecheisen, Johannes Aßfalg, and Dr. Martin Pfeifle. I also want to thank Prof. Dr. Christian Böhm, Christian Mahrt, Karsten Borgwardt, and Arthur Zimek for many inspiring discussions. Thank you for the constructive and productive team work. Special thanks to Elke "Elki" Achtert, the power woman in our research team.

Furthermore, I want to thank Otmar Hilliges for fruitful multidisciplinary discussions about similarity of multimedia objects and musical genres.

I also appreciate the substantial help of the students whose study thesis or diploma thesis I supervised. This includes Rolph Kreis, Markus Dolic, Ilja Vishnevski, Tim Schmidt, Georg Straub, Oleg Galimov and Michael Gruber. They helped me in many ways including implementation, data processing,

and testing.

I am very grateful for the background support of Susanne Grienberger, who managed much of the administrative work. She also gave me invaluable hints for improving on my English. Furthermore, I want to express special thanks to Franz Krojer, who helped to master all technical issues. He promptly provided tools that helped to accelerate my work.

Finally I like to thank my family and friends, who constantly supported me during the development of this thesis. My parents who always supported my career and encouraged me to find my way. Frank Riffel, with whom I developed DeliTracker which sparked my interest in Multimedia content. Florian Vorberger, who is the co-programmer of DeliPlayer and who also implemented the MUSCLE approach. Thanks to Dr. Karen Richter for proof-reading the intro & outro part of my thesis.

Abstract

The rapid progress of digital technology has led to a situation where computers have become ubiquitous tools. Now we can find them in almost every environment, be it industrial or even private. With ever increasing performance computers assumed more and more vital tasks in engineering, climate and environmental research, medicine and the content industry. Previously, these tasks could only be accomplished by spending enormous amounts of time and money. By using digital sensor devices, like earth observation satellites, genome sequencers or video cameras, the amount and complexity of data with a spatial or temporal relation has gown enormously. This has led to new challenges for the data analysis and requires the use of modern multimedia databases.

This thesis aims at developing efficient techniques for the analysis of complex multimedia objects such as CAD data, time series and videos. It is assumed that the data is modeled by commonly used representations. For example CAD data is represented as a set of voxels, audio and video data is represented as multi-represented, multi-dimensional time series.

The main part of this thesis focuses on finding efficient methods for collision queries of complex spatial objects. One way to speed up those queries is to employ a *cost-based decompositioning*, which uses interval groups to approximate a spatial object. For example, this technique can be used for the Digital Mock-Up (DMU) process, which helps engineers to ensure short product cycles. This thesis defines and discusses a new similarity measure for time series called *threshold-similarity*. Two time series are considered similar if they expose a similar behavior regarding the transgression of a

given threshold value. Another part of the thesis is concerned with the efficient calculation of *reverse k-nearest neighbor (RkNN) queries* in general metric spaces using conservative and progressive approximations. The aim of such RkNN queries is to determine the impact of single objects on the whole database. At the end, the thesis deals with *video retrieval* and *hierarchical genre classification* of music using multiple representations. The practical relevance of the discussed genre classification approach is highlighted with a prototype tool that helps the user to organize large music collections.

Both the efficiency and the effectiveness of the presented techniques are thoroughly analyzed. The benefits over traditional approaches are shown by evaluating the new methods on real-world test datasets.

Zusammenfassung

Aufgrund der rasanten Entwicklung von digitalen Technologien ist der Computer, sowohl im privaten als auch im industriellen Umfeld, als zentrales Hilfsmittel heute allgegenwärtig. Mit zunehmender Leistungsfähigkeit haben Computer wichtige Aufgaben im Maschinenbau, in der Umwelt- und Klimaforschung, in der Medizin oder in der Medienbranche übernommen, die vorher nur unter Aufbietung enormer zeitlicher und finanzieller Ressourcen bewältigt werden konnten. Durch den Einsatz von digitalen Erfassungsgeräten wie z.B. Erdbeobachtungssatelliten, Gensequenzierern oder Videokameras wachsen die Daten mit räumlichem und zeitlichem Bezug in Menge und Komplexität drastisch an, was zu neuen Herausforderungen bei deren Analyse führt und den Einsatz moderner Multimedia-Datenbanksysteme notwendig macht.

Das Ziel dieser Doktorarbeit ist es, effiziente Verfahren für die Analyse von komplexen Multimedia-Objekten, wie z.B. CAD-Daten, Zeitreihen und Audio- bzw. Videodaten, zu entwickeln. Ausgegangen wird dabei von der Modellierung der Daten in geläufigen Darstellungsformen. So werden z.B. CAD-Daten als Mengen von Voxeln, Audio- bzw. Videodaten als multirepräsentierte, mehrdimensionale Zeitreihen aufgefaßt.

Im Einzelnen beschäftigt sich die Arbeit mit der effizienten Beantwortung von Kollisionsanfragen auf komplexen räumlichen Objekten. Die komplexen Objekte werden dabei mittels einer *kostenbasierten Zerlegung* in einen einfachen Grundtyp zerlegt, mit dessen Hilfe die Anfragebearbeitung erheblich beschleunigt werden kann. Dies ist z.B. beim digitalen Zusammenbau (Digital Mock-up) von Interesse, wodurch Ingenieure die Anforderungen immer

kürzer werdender Produktzyklen meistern können. Des weiteren wird die *Threshold-Similarity* als neues Ähnlichkeitsmaß auf Zeitreihen eingeführt. Dabei werden zwei Zeitreihen als ähnlich angesehen, wenn sie ein ähnliches Verhalten bezüglich der Überschreitungen eines vorgegebenen Grenzwertes aufweisen. Ein weiterer Abschnitt beschäftigt sich mit der effizienten Berechnung von *Reversen-k-Nächste-Nachbar-Anfragen (RkNN-Anfragen)* in metrischen Räumen mittels konservativer und progressiver Approximationen. Ziel von RkNN-Anfragen ist es, den Einfluß des Anfrageobjekts auf die gesamte Datenbank zu bestimmen. Schließlich befaßt sich die Arbeit mit der *Video-Suche* sowie der *hierarchischen Genre-Klassifikation* von Musikstücken basierend auf multiplen Repräsentationen. Die Praxisrelevanz des Genre-Klassifikationsverfahrens wird in einer realen Anwendung demonstriert.

Die Effizienz und Effektivität der vorgestellten Techniken wird ausgiebig untersucht und die Vorteile gegenüber herkömmlichen Verfahren mittels Real-Datenbanken experimentell nachgewiesen.

Contents

Acknowledgement iv

Abstract vi

Zusammenfassung viii

I Preliminaries 1

1 Introduction 3

1.1 Outline 5

2 Purpose of the Thesis 13

2.1 Analysis of Spatial Data 13

2.2 Modeling Spatial Data 14

2.2.1 Multi-Step Query Processing 15

2.2.2 Digital Mock-up 16

2.2.3 Modeling Spatial Objects 18

2.2.4 Triangle Meshes 19

2.2.5 Voxel-Sets and Voxel-Sequences 20

2.2.6 Decomposition Algorithm 23

2.2.7 Compression Techniques 23

2.2.8 Relational Spatial Indexing 26

2.2.9 Test Datasets 28

2.3 Analysis of Temporal Data 28
2.3.1 Measuring Similarity 29
2.3.2 Application Ranges for Threshold Queries 32
2.3.3 Test Datasets 34
2.4 Analysis using RkNN Queries 39
2.4.1 RkNN Search in Euclidean Space 41
2.4.2 Test Datasets 43
2.5 Analysis of Video & Audio Data 44
2.5.1 Video Retrieval 45
2.5.2 Summarization Techniques 47
2.5.3 Similarity Search Based on Multiple Representations . 48
2.5.4 Test Dataset 49
2.5.5 Audio Classification 50
2.5.6 Test Dataset 53

II Analysis of Spatial Data 55

3 Introduction 57

4 Cost-Based Decompositioning of Complex Spatial Objects 61
4.1 Interval Groups 62
4.2 Storing Interval Groups in an ORDBMS 64

5 Compression of Interval Groups 67
5.1 Patterns 67
5.2 Compression Rules 68
5.3 Spatial Compression Techniques 69
5.3.1 Quick Spatial Data Compressor (QSDC) 70

6 Grouping into Interval Groups 73
6.1 Query Distribution 73

6.2 Access Probability 75
6.3 Evaluation Cost 75
6.4 Decomposition Algorithm 77

7 Query Processing **79**
7.1 Decomposition of the Query Object 80
7.1.1 Query object is a database object 80
7.1.2 Query object is no database object 81
7.2 Intersection Query 82
7.3 The intersect SQL Statements 83
7.4 Optimizations 84
7.4.1 Fast Intersection Test for Interval Groups 85
7.4.2 Ranking 86

8 Experimental Evaluation **89**
8.1 Storage Requirements 91
8.2 Update Operations 92
8.3 Query Processing 93
8.3.1 MaxGap 94
8.3.2 GroupInt 95
8.3.3 Window Queries 97

III Analysis of Temporal Data **99**

9 Introduction **101**
9.1 Preliminaries 102
9.2 Threshold Based Similarity Measure 102
9.2.1 General Idea 104
9.2.2 Threshold Based Representation vs. Dimensionality Reduction 105
9.2.3 Similarity-Distance Measures for Intervals 106

9.2.4 Contributions and Outline 108

10 Threshold Based Similarity Search **109**

10.1 Threshold-Crossing Time Intervals 109

10.2 Similarity Model for Time Intervals 110

10.3 Similarity Model for Threshold-Crossing Time Intervals 111

10.4 Similarity Queries Based on Threshold Similarity 112

11 Threshold Based Indexing **115**

11.1 Managing Threshold-Crossing Time Intervals with Fixed τ . . 116

11.2 Managing Threshold-Crossing Time Intervals for Arbitrary τ . 118

11.3 Trapezoid Decomposition of Time Series 121

11.4 Parameter Space Indexing 123

12 Threshold Based Query Processing **127**

12.1 Preliminaries 128

12.2 Pruning Strategy for Threshold Queries 129

12.3 Threshold-Based ε-Range Query Algorithm 132

12.4 Filter Distance for the Threshold Similarity 133

12.4.1 Lower Bounding Threshold Distance 134

12.4.2 Pruning Based on Lower Bounding Distance 136

12.5 Threshold-Based Nearest-Neighbor Query Algorithm 139

13 Experimental Evaluation **145**

13.1 System Environment 145

13.2 Datasets 146

13.3 Performance Results 146

13.4 Evaluation of the Threshold Based Similarity Measure 151

13.4.1 Comparison to Traditional Distance Measures 152

13.4.2 Comparison of Different Similarity Distances for Time Intervals 152

13.4.3 Comparison of Different Similarity Distances for Sets of Time Intervals . . . 153
13.4.4 Results on Scientific Datasets . . . 153

IV Analysis using Reverse Nearest Neighbor Queries 155

14 Introduction 157
14.1 Contributions . . . 158
14.2 Problem Definition . . . 159

15 kNN Distance Approximations for RkNN Search 161
15.1 Conservative Approximation of k-NN Distances . . . 164
15.2 Optimization Step 1 . . . 168
15.3 Optimization Step 2 . . . 171
15.4 Optimization Step 3 . . . 172
15.5 Summary: The Optimization Algorithm . . . 176
15.6 Progressive Approximation of kNN Distances . . . 178
15.7 Aggregating the Approximations . . . 178

16 RkNN Search Algorithm 181

17 Experimental Evaluation 183
17.1 Metric RkNN Search . . . 183
17.1.1 Runtime w.r.t. database size . . . 185
17.1.2 Runtime w.r.t. parameter k . . . 185
17.1.3 Pruning capabilities . . . 186
17.2 Euclidean RkNN Search . . . 186
17.2.1 Naive approaches . . . 187
17.2.2 Runtime w.r.t. database size . . . 188
17.2.3 Runtime w.r.t. parameter k . . . 188

17.2.4 Pruning capabilities 188

V Analysis of Video & Audio **191**

18 Introduction **193**

19 Video Retrieval **195**

19.1 Multi-represented Similarity Search in Multimedia Databases . 196
19.2 Weighting Functions For Summarizations 197
19.2.1 A Weighting Function Based on Support 197
19.2.2 A Weighting Function Based on Specific Quality Measures 198
19.2.3 A Weighting Function Based on Local Neighborhood . 200
19.2.4 A Weighting Function Based on Entropy 200
19.3 Combining Multiple Representations for Similarity Detection . 203
19.3.1 Higher-order Summarizations 204
19.3.2 First-order Summarizations 204
19.4 Experimental Evaluation 205
19.4.1 Multi-represented vs. Uni-represented Similarity Search 205
19.4.2 Multi-represented Similarity Search Applications 207

20 Audio Classification **209**

20.1 Efficient Hierarchical Genre Classification 210
20.1.1 Hierarchical Instance Reduction. 211
20.1.2 Hierarchical Genre Classification by Using Multiple Representations. 212
20.2 Experimental Evaluation 214
20.2.1 Effectiveness 214
20.2.2 Efficiency 216
20.3 Prototype 216
20.4 Practical Benefits 217

VI Conclusions and Outlook 219

21 Summary 221

21.1 Summary of Contributions 222

21.1.1 Preliminaries (Part I) 222

21.1.2 Analysis of Spatial Data (Part II) 222

21.1.3 Analysis of Temporal Data (Part III) 223

21.1.4 Analysis using Reverse Nearest Neighbor Queries (Part IV) 223

21.1.5 Analysis of Video & Audio Data (Part V) 224

22 Outlook 227

22.1 Future Work 227

List of Figures 231

List of Tables 235

References 237

Part I

Preliminaries

Chapter 1

Introduction

In recent years, the processing of high-level content such as CAD-data, graphs, audio files or movie clips has experienced a boost in public interest. This was mainly possible due to the enormous progress in consumer electronics, data transmission techniques and computer technology in general. While in the past only a few experts had the know-how to manage large amounts of multimedia data, nowadays, hundreds of gigabytes of images, digital videos and audio files can be found in an ordinary household. As a consequence, the number of potential users of multimedia databases (MMDBS) is today much higher compared to a few years ago.

This leads to the problem that the accumulated data have to be managed in such a way that a user can easily retrieve the desired information from this huge pile of data. In the process, the requirements for analysis- and retrieval-systems have also increased drastically. In a professional environment, we can still assume that the users are rather well-trained. However, systems which are designed for private use have to operate on a much more intuitive level or they will not become accepted. Another important factor is speed, i.e. a fast response time. In general, most home-users lose interest in a system if it fails to produce results in a reasonable amount of time. This is also called efficiency.

An efficient algorithm is able to produce a result with reduced cost or

in shorter time compared to an inefficient version. Efficient algorithms are sought after when it comes to solving problems in both business and private life. They help the industry to produce cheaper goods and to provide better services in less time which means everyone can get more value for money, i.e. they help to improve the quality of life. Of course it is important not to forget effectiveness. Nowadays, an employer usually demands that employees do their daily work in an efficient way. Moreover, the employer also assumes that the work is done effectively. Which means that a high quality output is generally so important that it is simply implied without stating it explicitly. The solutions presented in this thesis are evaluated with respect to effectiveness in order to demonstrate their usefulness for the analysis in case the presented techniques have parameters which affect the output quality.

There are several different types of databases which manage multimedia content. Databases for audio-visual content are probably the best known, at least to the common consumer. With Internet access, videos and songs are rather easy to acquire. All it takes is an account on a media portal and millions of files are just a fingertip away. The easy availability of these multimedia files also creates the need to manage this data efficiently. A customer who likes a certain video will quite likely spend more money on similar videos. Categorizing songs into similar classes can also help to increase the total revenue. Ordinary music shops usually have separate areas where pop, folk or classical music is sold. So it would be profitable for online shops to present the content to the customers in a similar way. This thesis addresses two problems, similarity search in video databases as well as classification of audio files. The proposed solutions work fully content-based and do not rely on additional information.

Other types of multimedia databases which can mainly be found in industry and science are spatial and temporal databases. Spatial data denotes data which are related to a specific space. Typical instances of spatial data represent objects which reproduce our physical world or they are artificial objects created by engineers. Complex spatial objects usually consist of more than one part. This creates the need to ensure a perfect fit of the different parts of an object during the construction phase. Usually, this is accom-

plished by performing collision queries during the Digital Mock-Up (DMU) process. Another prevalent data type are time series. They can usually be found in temporal databases. The most common similarity measures for time series are the Euclidean distance and the Dynamic Time Warping (DTW) distance. Both similarity measures compare one point of time of the first time series to a compatible point of time of the second time series. In order to become more tolerant to perturbations, it would be beneficial to restrict the similarity of two time series to a given threshold. Our novel concept of threshold-similarity for time series databases realizes this idea. A threshold query reports those time series which exceed a user-defined query threshold at similar points of time compared to the query time series. Similar to the computation of the Euclidean and the DTW distance, our new similarity measure can be accelerated by index structures.

The increasing complexity of multimedia data also imposes new challenges in the area of query processing. For instance, let us consider stores which have a rather dynamic range of articles like shops of ringtones for cellphones, pieces of music, or computer games. In order to ensure efficient and effective sales, it makes sense to notify only those customers who would be interested in a new product because they have bought similar things in the past. For this task, a list of all customers is required which have the new product in the k-nearest neighbor (kNN) set of their previously acquired products. This type of query is also called reverse k-nearest neighbor (RkNN) query. A RkNN query reports these objects in a database which have a given query object q in the set of their k-nearest neighbors. The idea of a RkNN query is to identify the influence of the query object on the entire dataset.

1.1 Outline

In this thesis, various problems and solutions related to the efficient analysis in multimedia databases are discussed. It is divided into a short introductory part and four major parts called *analysis of spatial data*, *analysis of temporal data*, *analysis using RkNN queries*, and *analysis of video & audio data*. Some

Table 1.1: List of author's publications on which this thesis is based.

Part II	Analysis of Spatial Data	
	Der virtuelle Prototyp: Datenbankunterstützung für CAD-Anwendungen.	[BKK+04]
	Effective Decompositioning of Complex Spatial Objects into Intervals.	[KKPR04a]
	Object-Relational Management of Complex Geographical Objects.	[KKPR04b]
	Distributed Intersection Join of Complex Interval Sequences.	[KKPR05]
Part III	Analysis of Temporal Data	
	Threshold Similarity Queries in Large Time Series Databases.	[AKK+06b]
	Similarity Search on Time Series based on Threshold Queries.	[AKK+06a]
Part IV	Analysis using Reverse Nearest Neighbor Queries	
	Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces.	[ABK+06]
Part V	Video Retrieval & Audio Classification	
	Effective Similarity Search in Multimedia Databases using Multiple Representations.	[KKKP06]
	Hierarchical Genre Classification for Large Music Collections.	[BKKP06]
	MUSCLE: Music Classification Engine with User Feedback.	[BKK+06]

of the ideas and techniques discussed in the different parts of this thesis have already been published by the author. For clearness and convenience, these publications are listed in Table 1.1.

Specialized techniques are required to manage objects in modern multimedia databases efficiently and effectively. The contributions of this thesis mainly focus on techniques for speeding up the analysis of various types of multimedia data. In particular, several approaches are presented for the efficient analysis of spatial, temporal, and audio & video data.

The major contributions of this thesis include:

- A cost-based approximation of spatial data into interval groups which is a middle course between replicating and non-replicating spatial index structures.
- An approximation algorithm for spatial data which takes the access probability and the decompression cost of the interval groups into ac-

count in order to to reduce the I/O cost.

- A novel similarity measure for time series, enabling data mining tasks focused on a certain amplitude value.
- An efficient decomposition method for time series, allowing a threshold invariant representation of time series.
- An efficient query algorithm for threshold-based distance-range queries and threshold-based nearest-neighbor queries.
- An efficient reverse k-nearest neighbor algorithm which can be applied to general metric objects.
- A reverse k-nearest neighbor approach which is applicable to the generalized RkNN problem where the value of k is specified at query time.
- A novel approach for similarity search in multimedia databases which takes multiple representations of multimedia objects into account.
- Several weighting functions to rate the significance of a feature of each representation for a given database object during query processing.
- A novel approach for the hierarchical classification of music pieces into a genre taxonomy which is able to handle multiple characteristics of music content.
- MUSCLE, a tool which allows the user to organize large music collections in a genre taxonomy and to modify class assignments on the fly.

The following chapter of Part I provides a brief and rather general overview of existing methods for analysis in multimedia databases. It lists several common problems and gives a short sketch of our solution for these problems. The chapter also introduces the datasets which are used throughout this thesis.

Part II introduces an approach involving cost-based approximations in order to accelerate the processing of spatial intersection queries for linearized

spatial objects. The approach is based on interval groups which are created by using a cost-based decompositioning algorithm. This algorithm takes the access probability and the decompression cost of the interval groups into account.

Chapter 3 gives a brief introduction and indicates how to manage complex spatial objects in a linearized form that supports an easy integration into commercial database systems.

Chapter 4 introduces interval groups as a new and general concept to approximate interval sequences. In contrast to the common interval decompositions that suffer from the high redundancy of complex shaped objects rendered with high resolution, the interval groups combine sets of intervals into approximations. The chapter also introduces three grouping rules which help to find a good grouping of the intervals.

Chapter 5 presents a new effective data compressor which exploits gaps and patterns that are included in the byte representations of our interval groups. Thus, we store the exact information of the interval groups in a compressed way. It also states two compression rules which should be fulfilled by a good data compressor.

Chapter 6 The interval containers are created by using a cost-based decomposition algorithm which takes the access probability and the decompression cost of the interval containers into account.

Chapter 7 discusses how we can efficiently carry out intersection queries on top of the SQL-engine. Our approach is based on the RI-tree for efficiently detecting intersecting interval hulls. Furthermore, we present two optimizations in order to avoid unnecessary intersection tests. First, we try to determine intersecting interval groups by using aggregated information. Second, we introduce a probability model which leads to an ordering for the candidate pairs such that the most promising tests can be carried out first.

Chapter 8 presents the efficiency evaluation on real-world test data. The results point out that our new concept accelerates collision queries on the RI-tree by up to two orders of magnitude.

Part III introduces the new concept of threshold-based similarity search for time series databases. In this concept, time series are also represented by means of interval sequences, similar to the form of representation used for the spatial objects in Part II. In particular, we introduce a novel similarity measure which does not only provide new prospects in data mining in time series databases but also allows the development of efficient methods for searching in very large databases, comprising large and complex time series objects. We propose effective similarity search methods on this new time series representation. Furthermore, we show how the time series can be indexed so that the new similarity queries can be processed in a very efficient way.

Chapter 9 proposes a novel query type on time series databases called *threshold query* and contains some preliminary definitions that are relevant to the remaining chapters. Using various examples, it is shown that this new analysis concept is important for several applications in medicine, biology and for analysis of environmental air pollution. It also contains a short overview of commonly used distance measures for one-dimensional intervals.

Chapter 10 formally defines threshold queries and introduces two versions of this query type, the *threshold-based ε-range query* and the *threshold-based k-nearest-neighbor query*. First, we introduce a new form of time series representation called threshold-crossing time intervals. This representation consists of a sequence of intervals and indicates at which time slots the time series is above or below a specified threshold value. Given a query time series Q and a threshold τ, threshold queries return those time series in which the threshold-crossing time intervals are most similar to Q. This type of query is motivated by several practical application ranges.

Chapter 11 presents a novel approach for managing time series data to efficiently support threshold queries. In particular, we propose a threshold invariant representation of time series which allows the pre-computation of the threshold-based time series representations without the need to commit oneself to a fixed threshold value. This means that the query threshold can be chosen at query time. The proposed concept is based on an efficient

decomposition algorithm, decomposing time series into a set of trapezoids which are subsequently inserted into a conventional spatial access method. At query time, we have to access only the relevant parts of the decomposed time series which can be efficiently retrieved from the index.

In Chapter 12, we develop a scalable algorithm to answer threshold queries for arbitrary thresholds. The proposed methods are based on pruning strategies for the two threshold-query variants, the *threshold-based ε-range query* and the *threshold-based k-nearest-neighbor query*. Both pruning strategies are based on a lower bound criterion for the threshold distance which is used to filter out true drops.

Chapter 13 demonstrates the performance of the solutions proposed in the previous chapters by an extensive experimental evaluation on real-world and artificial time series data. The effectiveness of our novel similarity measure is proven against several competing approaches, in particular the Euclidean distance and the dynamic time warping (DTW) distance on several established benchmark datasets. Furthermore, we evaluate different similarity measures for the proposed time series representation.

Part IV proposes the first approach for efficient RkNN search in arbitrary metric spaces where the value of k is specified at query time. Our approach takes advantage of existing metric index structures and proposes to use conservative and progressive distance approximations in order to filter out true drops and true hits. In particular, we approximate the k-nearest-neighbor distance for each data object by upper and lower bounds using two functions of only two parameters each. Thus, our method does not generate any considerable storage overhead.

Chapter 14 stresses the need for a solution for the generalized RkNN problem where the value of k is not known in advance and may change from query to query. It also lists some definitions which form the basis of the proposed technique.

Chapter 15 shows how to compute space-saving functions that conservatively and progressively approximate the kNN distances of a database object. These approximations allow to identify objects that can be safely dropped,

or which are true hits. For the remaining objects, we need a refinement step inducing a kNN query for each candidate.

Chapter 16 presents a RkNN search algorithm which builds on a metric index structure. Due to the use of such a metric index structure it is applicable to general metric objects. It also can answer RkNN queries for any k specified at query time.

Chapter 17 shows in a broad experimental evaluation on real-world data the scalability and the usability of our novel approach. In metric spaces, our approach yields a significant speed-up over the sequential scan and naive indexing solutions. We also demonstrate that our proposed concept even outperforms the only existing solution for RkNN search for Euclidean vector data.

In Part V, solutions for video retrieval and audio classification of multi-represented objects are presented.

Chapter 18 indicates the need for efficient retrieval and classification techniques in order to organize large collections containing video and audio data.

Chapter 19 introduces a video retrieval approach which is able to integrate multiple representations such as audio and image features into the query processing. We propose methods for weighting each representation which consist of multiple instances. The weighting techniques can be applied to higher-order and first-order summarization techniques. In addition, we propose a method for combining multiple representations for similarity search by weighting each representation.

Chapter 20 again handles data which are described by multiple representations and multiple instances. We present a novel hierarchical semi-supervised technique for instance reduction. The reduced descriptions are used afterwards for hierarchical classification of pieces of music. Furthermore, we use object adjusted weighting in order to take advantage of multiple representations. Moreover, we present the tool MUSCLE which implements the techniques introduced in the beginning of this chapter. MUSCLE features a hierarchical classification in combination with interactive user feedback and

a flexible multi-assignment of songs to classes.

Part VI concludes this thesis.

Chapter 21 recapitulates and discusses the major contributions of the thesis. Finally, some possible future research directions are indicated in Chapter 22.

Chapter 2

Purpose of the Thesis

This chapter presents several problems regarding the efficient analysis in multimedia databases discusses previous approaches and briefly sketches our solution. A multimedia database system (MMDBS) not only manages objects such as video and audio data, but also spatial and temporal objects. All these objects have one thing in common: they have a complex structure. Since there is no definitive definition of what exactly a multimedia object is, we consider all databases which manage complex objects a MMDBS. Thus, spatial and temporal databases are possible occurrences of a MMDBS. This thesis addresses various application areas of MMDBSs, like collision queries in spatial databases, similarity search in temporal databases, reverse nearest neighbor search, video retrieval and audio classification. For each considered area, we describe typical problems, review previous methods and provide an efficient solution for these problems. We also present the datasets that are used in the following parts of this thesis in order to demonstrate the relevance of our new techniques.

2.1 Analysis of Spatial Data

Objects from the physical world or artificial objects derived from engineering design are typical instances of spatial data. Each database object has

a well-defined location and extension in the data space. Spatial objects are represented by spatial data types, while spatial predicates describe the relationships between them. In this section, we give a brief introduction into the area of spatial data modeling and spatial queries, with a special emphasis on voxelized objects. A more detailed overview of this area can be found in [KPP+03].

2.2 Modeling Spatial Data

A spatial object is regarded as a distinct entity occupying an individual location in a one- or multidimensional data space. Furthermore, it may be extended along some (or all) dimensions. The spatial objects of the real world can be considered as a collection of individual, two- or three-dimensional parts, while each part potentially represents a complex and intricate geometric shape. Examples of such complex objects are geographical regions or parts of a car or an airplane.

Data modeling requires that at least the following three main components be specified: spatial data type, data structures and operations on spatial data, in particular, spatial predicates [VLB05]. As spatial data is often associated with Geographic Information Systems (GIS), this type of data is often defined as information that describes the distribution of objects upon the surface of the earth. In the geographical context, spatial data contains information concerning the location, and shape of, and relationships among, geographic features [DeM97]. As a consequence, traditional spatial data types include points, lines and polygons, which allow for representation of any geographical entity.

Thus, spatial applications not only store spatial entities, but also allow for the spatial relationships between these entities to be specified. Queries on spatial databases typically evaluate the relationships by spatial predicates, which often return *true* or *false* as an answer. Spatial predicates can be mainly categorized into three different types: topological, directional and metrical. Topological properties are the most fundamental primitives for spa-

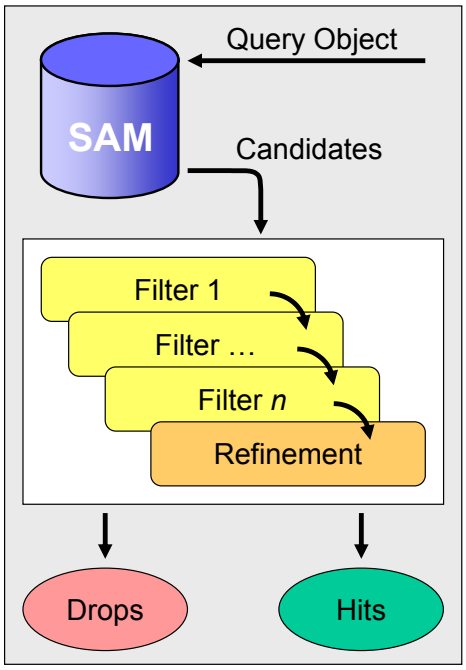


Figure 2.1: Multi-step query processing.

tial predicates. The principle topological relationships between two spatial objects have been captured by the 9-intersection model proposed by Egenhofer and Sharma [ES93]. Metric predicates are associated with quantitative information. This way, the user can find out, for example, the distance, the degree of overlap, or the depth of the penetration between two objects. In many mechanical engineering and architecture applications the *intersection* predicate is most important. It is a combination of the two basic predicates "*not disjoint*" and "*not meets*". In this thesis, we emphasize the efficient evaluation of the intersection predicate applied to two- and three-dimensional objects.

2.2.1 Multi-Step Query Processing

An efficient processing of queries on spatial data is provided by a multi-step architecture [KBS93, BHKS93] as shown in Figure 2.1. The idea behind the depicted spatial query processor is to reduce the number of objects for which

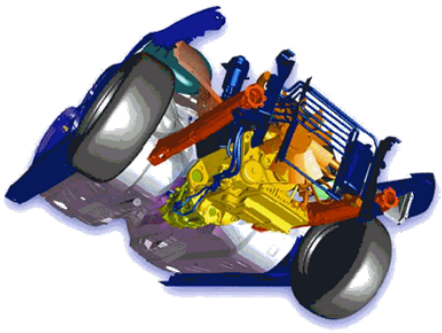
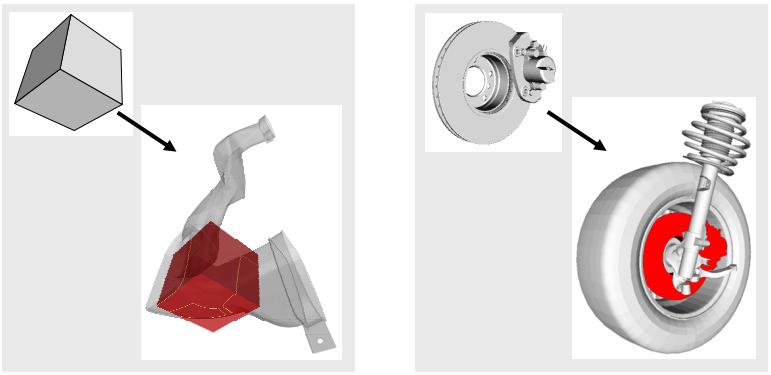


Figure 2.2: Virtual prototype of a car.

an expensive refinement step is performed by carrying out one or more preprocessing operations. If these preprocessing operations are selective enough, the number of objects investigated in the expensive refinement step can be decreased significantly. However, the cascade of filters should be inexpensive to evaluate. Otherwise the desired effect, fewer refinement operations, is dearly bought. A superset of the objects qualifying for the spatial predicate is computed in each the filter step. This set is also called candidate set. Subsequent filter steps may further reduce the number of candidates by using, for example, more accurate representations [BKS93]. For example, the filter for selection queries could be an intersection evaluation on the minimal bounding rectangles (MBR) of the objects. The intersection of rectangles can be evaluated more efficiently than the exact object geometry. Furthermore, they are suitable object approximations used in many spatial access methods (SAM) like the R*-tree [BKSS90]. In case of highly selective queries, the first filter step should be processed by a spatial access method (SAM). The multi-step query process is completed by the refinement step which tests the exact geometry of the remaining candidates.

2.2.2 Digital Mock-up

In CAD databases, each instance of a part occupies a specific region in the three-dimensional product space (cf. Figure 2.2). Together, all parts of a given product version and its variant represent a virtual prototype of the



(a) Box volume query (b) Collision query

Figure 2.3: Common queries on spatial data.

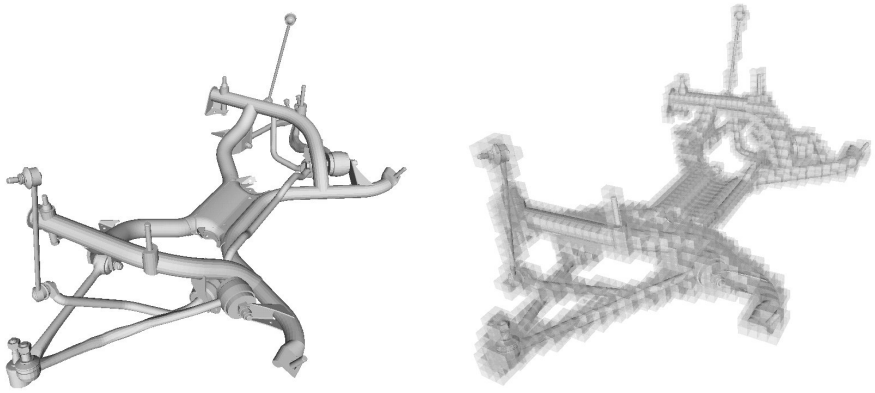
constructed geometry. Virtual engineering requires accessing the product space by spatial predicates in order to "*find all parts intersecting a specific query volume*" or to "*find all parts in the immediate spatial neighborhood of the disk brake*". Unfortunately, the inclusion of the respective spatial predicates is not efficiently supported by common, structure-related information systems.

In the automobile industry, late engineering changes caused by problems with fit, appearance or shape of parts account for 20-50 percent of the total die cost [CF91]. Therefore, tools for the digital mock-up (DMU) of engineering products have been developed to enable a fast and early detection of colliding parts, solely based on the available digital information. Unfortunately, these systems typically operate in main-memory and are not capable of handling more than a few hundred parts. They require, as input, a small, well-assembled list of the CAD files to be examined. With the traditional file-based approach, each user has to select these files manually. This can take hours or even days of preprocessing time, since the parts may be generated on different CAD systems, spread over many file servers, and managed by a variety of users [SBP98]. In a concurrent engineering process, several cross-functional project teams may be recruited from different departments, including engineering, production, and quality assurance, to develop their own parts as a contribution to the whole product.

For example, a team working on section "12B" of an airplane may not want to mark the location and the format of each single CAD file of the adjacent sections "12A" and "12C". In order to do a quick check of fit or appearance, they are only interested in the colliding parts. Moreover, the internet is gaining importance for industrial file exchange. Engineers working in the USA may want to upload their latest component design to the CAD database of their European customer in order to perform interference checks. Thus, they need a fast and comfortable DMU interface to the Engineering Data Management system (EDM). Figure 2.3 depicts two typical spatial queries on a three-dimensional product space, retrieving the parts intersecting a given box volume (box volume query), and detecting the parts colliding with the geometry of a query part (collision query). A spatial filter for DMU-related queries on huge CAD databases is easily implemented by a spatial access method, which determines a tight superset of the parts qualifying for the query condition. Then, the computationally intensive query refinement on the resulting candidates, including the accurate evaluation of intersection regions (cf. Figure 2.3(a)), can be delegated to an appropriate main memory-based CAD tool.

2.2.3 Modeling Spatial Objects

As they are the natural instances of the real world, the modeling of three-dimensional objects is very important and required for many application areas. Computer-Aided Design (*CAD*) and related areas, including Computer-Aided Engineering (*CAE*), Manufacturing (*CAM*), and Styling (*CAS*), are some of the most common emerging technologies in the field of spatial data management. These technologies are getting more and more indispensable in mechanical engineering facilities. In order to cope with the demands of accurate geometric modeling, we use universal representations which can be derived from any native geometric surface and solid. These representations can be successfully used to develop efficient query methods, in particular, collision queries. While geometric data models, including triangle meshes, are typically used for visualization, voxel sets can be used as a conservative



(a) Triangulated surface (b) Voxelized surface

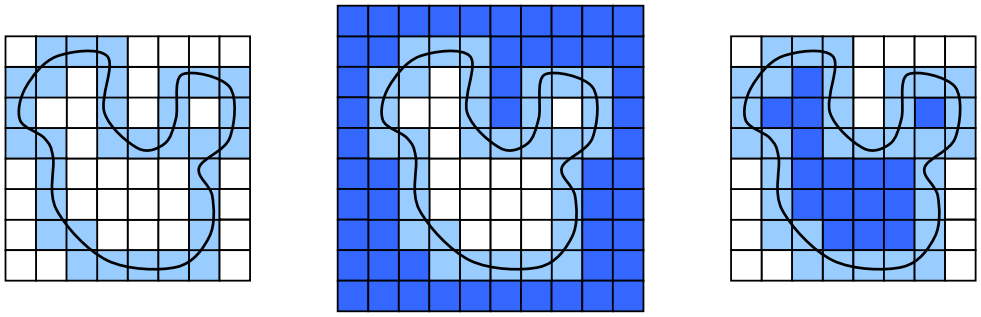
Figure 2.4: Scan conversion on a triangulated surface.

approximation for spatial keys.

2.2.4 Triangle Meshes

Accurate representations of CAD surfaces are typically implemented by parametric bicubic surfaces, including Hermite, Bézier, and B-spline patches. For many operations, such as graphical display or the efficient computation of surface intersections, these parametric representations are too complex [MH99]. A solution is to derive approximative polygon meshes, e.g. triangle meshes, from the accurate surface representation. These triangle meshes allow an efficient and interactive display of complex objects, by means of VRML (Virtual Reality Modeling Language) encoded files, and serve as an ideal input for the computation of spatial interference. For the digital mock-up (DMU), spatial interference detection or collision queries are a very important database primitive.

In the following, we assume a multi-step query processor that retrieves a candidate part S, which possibly collides with a query part Q. In order to refine such collision queries, a fine-grained spatial interference detection between Q and S can be implemented on their triangle meshes. We distinguish two actions for interference detection [MH99]: collision detection and



(a) Voxelized surface (b) Filled exterior (c) Inverted result

Figure 2.5: Filling a closed voxelized surface.

collision determination:

Collision detection: This basic interference check detects if the query part Q and a database part S collide. Thus, collision detection can be regarded as a geometric intersection join of the triangle sets for Q and S which terminates early after the first intersecting triangle pair has been found.

Collision determination: The actual intersection regions between a query part and a stored part are computed. In contrast to the collision detection, all intersecting triangle pairs and their intersection segments have to be reported by the intersection join.

2.2.5 Voxel-Sets and Voxel-Sequences

In order to employ efficient access methods, like the Relational Interval-tree (RI-tree) [KPS01, KPS00], as a query engine for a spatial database, we propose a conversion pipeline to transform the geometry of each single spatial object to an interval sequence by means of voxelization. A basic algorithm for the 3D scan-conversion of polygons into a voxel-based occupancy map has been proposed by Kaufmann [Kau87]. Similar to the well-known 2D scan-conversion technique, the runtime complexity required to voxelize a 3D polygon is $O(n)$, where n denotes the number of generated voxels. If we apply this conversion to the given triangle mesh of a CAD object (cf. Figure

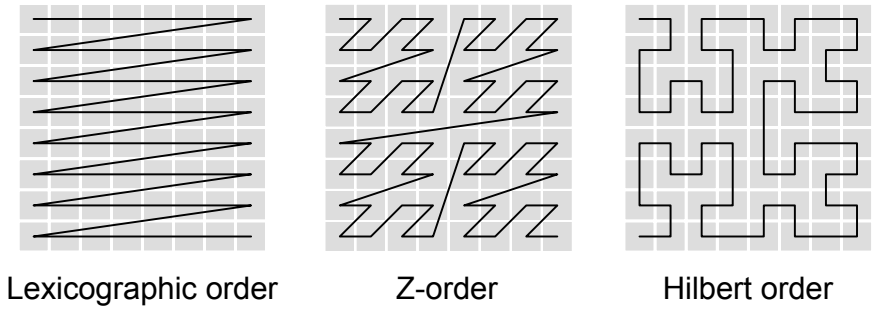


Figure 2.6: Different space-filling curves in a two-dimensional space.

2.4(a)), a conservative approximation of the part surface is produced (cf. Figure 2.4(b)). We assume a uniform three-dimensional voxel grid covering the entire product space.

If a triangle mesh is derived from an originally solid object, each triangle can be supplemented with a normal vector to discriminate the interior from the exterior space. Consequently, not only surfaces, but also solids, could potentially be modeled by triangle meshes. Unfortunately, triangle meshes generated by most faceters contain geometric and topological inconsistencies, including overlapping triangles and tiny gaps on the surface. Thus, a robust reconstruction of the original interior becomes very laborious. Therefore, we follow the common approach of voxelizing the triangle mesh of a solid object first (cf. Figure 2.5(a)), which yields a consistent representation of the object surface. Next, we apply a 3D flood-fill algorithm [FvDFH00] to compute the exterior voxels of the object (cf. Figure 2.5(b)). Accordingly, the outermost boundary voxels of the solid are determined. We restrict the flood-fill to the bounding box of the object, enlarged by one voxel in each direction. The initial fill seed is placed at the boundary of this enlarged bounding box. In the final step, we simply declare all voxels, which are neither boundary nor exterior voxels, as interior (cf. Figure 2.5(c)). In consequence, we obtain a volumetric reconstruction of the original solid, marking any voxel behind the outermost surface as interior. The above algorithm has a runtime complexity of $O(b)$, where b is the number of voxels in the enlarged bounding box.

The derived voxel set of an arbitrary surface or solid represents a consis-

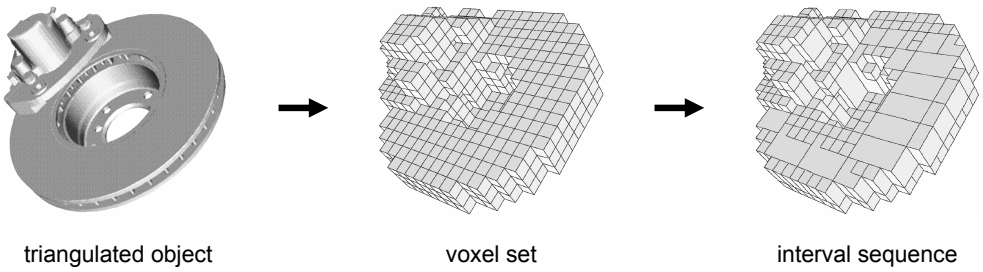


Figure 2.7: Conversion from a triangulated surface into an interval sequence.

tent input for computing interval sequences. The voxels correspond to cells of a grid, covering the complete data space. By means of space filling curves, each cell of the grid can be encoded by a single integer number, and thus, an extended object is represented by a set of integers. Most of these space filling curves achieve good spatial clustering properties. Therefore, cells in close spatial proximity are encoded by similar integers or, putting it another way, contiguous integers encode cells a in close spatial neighborhood. Examples for space filling curves include Hilbert-, Z-, and the Lexicographic-order, as depicted in Figure 2.6. The Hilbert-order generates the minimum number of intervals per object [Jag90, FR89], but unfortunately, it is the most complex linear order. Taking redundancy and complexity into consideration, the Z-order seems to be the best solution.

Voxels can be grouped together in such a way as an extended object can be represented by some continuous ranges of numbers. These ranges can then be described by a sequence of intervals. Figure 2.7 summarizes the complete transformation process from triangle meshes over voxel sets to interval sequences.

Gaede pointed out that the number of voxels representing a spatially extended object exponentially depends on the granularity of the grid approximation [Gae95]. Furthermore, the extensive analysis given in [MJFS96] and [FJM97] shows that the asymptotic redundancy of an interval- and tile-based decomposition is proportional to the surface of the approximated object. Thus, in the case of large high-resolution parts, e.g. wings of an airplane,

the number of tiles or intervals can become unreasonably high.

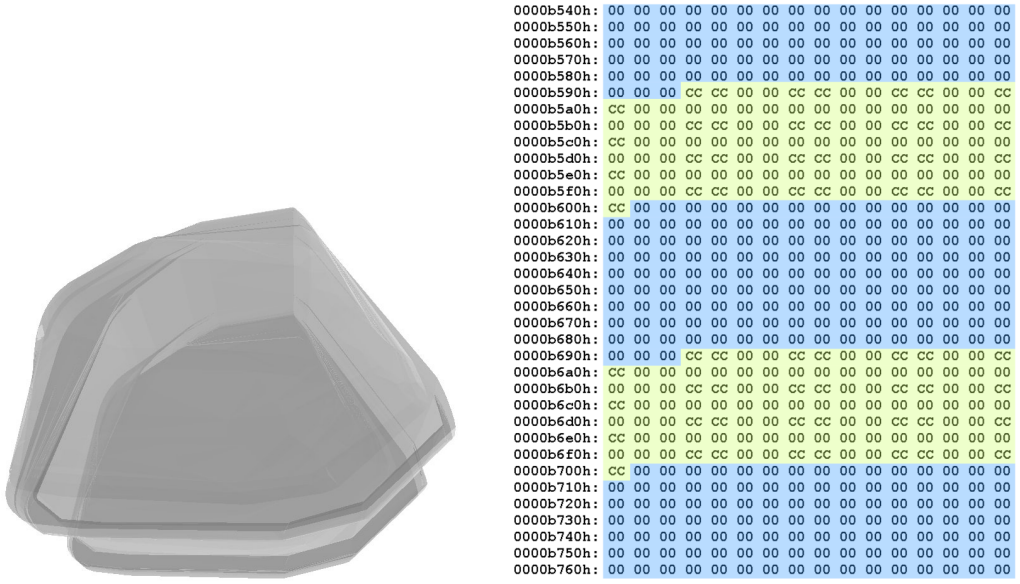
2.2.6 Decomposition Algorithm

In [SK93], Kriegel and Schiwietz tackled the complex problem of "complexity versus redundancy" for 2D polygons. They investigated the natural trade-off between the complexity of the components and the redundancy, i.e. the number of components, with respect to the effect on efficient query processing. The presented empirically derived root-criterion suggests to decompose a polygon consisting of n vertices in many $O(\sqrt{n})$ index entries. As this root-criterion was designed for 2D polygons and was not based on any analytical reasoning, it cannot be adapted to complex 3D objects. In this thesis, in contrast, we present an analytical decomposition approach which can be used for 2D and 3D objects.

2.2.7 Compression Techniques

In [KPPS03b], the interval representation of a spatial object is either stored uncompressed in the database, called *bit-oriented approach*, or by means of a simple run-length encoding, called *offset-oriented approach*. Depending on the length and the cardinality of the interval representation, it is individually decided which approach is preferable with respect to the use of secondary storage. The variant that uses less secondary disk space is then used for storage purposes. This approach is denoted as OPTRLE throughout the rest of this thesis. Because the OPTRLE technique is a combination of two different algorithms, each spatial object has to be compressed twice in order to chose which one leads to a more compact result.

Due to its simple design, OPTRLE achieves a rather poor compression ratio and slow (de-)compression speed (cf. Part II). In Figure 2.8(a) we see a bounding object of a three-dimensional automobile part, and in Figure 2.8(b) a small section of the corresponding bit-oriented representation taken from a hex-editor. Two observations are apparent in this figure. First, the



(a) Spatial object (b) Sequence representation with patterns

Figure 2.8: Patterns contained in a linearized object.

bit representation contains large portions of zeros, which belong to dead space. Second, some sections of the non-zero parts occur repeatedly. The offset-oriented approach of the OPTRLE technique tries to address the first observation, but achieves only mediocre results. The second observation is not exploited at all by OPTRLE.

Thus, applying more sophisticated compression techniques on the interval representations could be promising. In the following, we shortly describe some of the most prominent lossless decompression techniques. For a more detailed survey on lossless and lossy compression techniques, we refer the reader to [RH93] and [SN02].

Run-Length Coding

Data often contains sequences of identical bytes. By replacing these repeated byte sequences with the number of occurrences, a substantial reduction of data can be achieved. This is known as run-length coding.

Pattern Substitution

This technique substitutes single characters for patterns that occur frequently. This pattern substitution can be used to code, for example, the terminal symbols of high-level languages (begin, end, if, etc.). The eliminated patterns are often stored in a separate dictionary. A widespread pattern substitution algorithm is LZ77 [LZ77]. This compression algorithm detects sequences of data that occur repeatedly by using a sliding window. An n-byte sliding window is a record of the last n characters in the input respectively the output data stream. If a sequence of characters is identical to one that can be found within the sliding window, the current sequence is replaced by two numbers: a distance value, indicating the starting point of the found sequence within the sliding window, and a length value, representing the maximum number of characters for which the two sequences are identical.

Statistical Coding

There is no fundamental reason that different characters need to be coded with a fixed number of bits. For instance, in the case of morse code, frequently occurring characters are coded with short strings, while seldom occurring characters are coded with longer strings. Such statistical coding depends on the frequency of individual characters or byte sequences. There are different techniques based on statistical coding, e.g. arithmetic coding and Huffman coding.

Huffman Coding

Given the characters that have to be encoded, together with their probabilities of occurrence, the Huffman algorithm determines the optimal coding using the minimum number of bits [Huf52]. Frequently occurring characters are assigned to short code words, whereas seldom occurring characters are replaced by longer code words. A Huffman code can be determined by successively constructing a binary tree, whereby the leaves represent the characters

that have to be encoded. The resulting Huffman coding table is necessary for the compression and decompression process and has to be stored along with the encoded data.

ZLIB

This popular approach compresses data first with LZ77, followed by Huffman coding. A compressed dataset consists of a series of chunks, corresponding to successive chunks of input data. The chunk sizes are arbitrary. Each chunk has its own Huffman tree, whereas the LZ77 algorithm is not confined to these chunks, and may refer to identical byte sequences in previous chunks [Deu96].

BZLIB2

This rather new approach implements the Burrows-Wheeler transform (BWT) followed by Move To Front (MTF) transformation and Huffman coding. The BWT algorithm takes a chunk of data and applies a sorting algorithm to it. The rearranged output chunk contains the same characters in a different order. The original ordering of the characters can be restored, i.e. the transformation is reversible [BW94]. MTF is a transformation algorithm that does not compress data, but can help to reduce redundancy. This reduction is especially apparent after a BWT where the data is likely to contain a lot of repeated characters.

2.2.8 Relational Spatial Indexing

In order to guarantee efficient query processing together with industrial strength, spatial index structures have to be integrated into fully-fledged object-relational database management systems (ORDBMSs). A wide variety of access methods for spatially extended objects has been published. For a general overview on spatial index structures, we refer the reader to the surveys of Manolopoulos, Theodoridis and Tsotras [MTT00] or Gaede

and Günther [GG98]. Recently, a general survey on the paradigm of relational index structures has been published [KPPS03a]. The basic idea of relational access methods relies on the exploitation of the built-in functionality of existing database systems. A relational access method delegates the management of persistent data to an underlying relational database system, by strictly implementing the index definition and manipulation on top of an SQL interface. In this manner, the SQL layer of the ORDBMS is employed as a virtual machine managing persistent data. The Relational Interval Tree (RI-tree) [KPS01, KPS00], Relational R-tree (RR-tree) [KRSB99] and the Relational Quadtree (RQ-tree) [FFS00], which are all based on the B^+-tree, are examples of this paradigm. In [KPS01] and [KPS00] the RI-tree, the RR-tree and the RQ-tree were compared to each other. It was shown that the RI-tree outperforms the RR-tree and the RQ-tree by factors between 4.6 and 58.3. Therefore, we focus on the RI-tree in this thesis. Please note that the developed techniques can also be applied to other relational index structures with little modification.

In [KPPS03b], a new indexing approach was presented that considerably accelerates the RI-tree for high resolutions. Nevertheless, the authors only addressed high resolution spatial data, and even in this case, they did not show how to decompose a spatial object. Their presented approach for storing the spatial objects was rather inefficient, as they did not exploit the fact that voxel sequences obtained from space filling curves tend to contain patterns. In this thesis, we use the approach presented in [KPPS03b] as comparison partner and show that our new approach, based on interval group sequences, outperforms it by more than one order of magnitude.

A promising way to cope with spatial data can be found somewhere in between replicating and non-replicating spatial index structures. We use the concept of interval groups which is a compromise between these two extremes. Based on the interval groups, we introduce a cost-based decomposition method for accelerating the Relational Interval Tree (RI-tree). Our approach uses compression algorithms for the effective storage of the decomposed spatial objects. The experimental evaluation on real-world 2D and 3D test data points out that our new concept outperforms the RI-tree by up

to two orders of magnitude with respect to overall query response time and secondary storage space.

2.2.9 Test Datasets

Our tests are based on three test datasets ***CAR***, ***PLANE*** and ***SEQUOIA***. The first two test datasets were provided by our industrial partners, a German car manufacturer and an American plane producer, in the form of high resolution voxelized three-dimensional CAD parts. The third dataset is based on a subset of two-dimensional GIS data representing woodlands, rivers, and transportation networks derived from the SEQUOIA 2000 benchmark [SFGM93], called SEQUOIA throughout this thesis, for simplicity. The CAR dataset consists of approximately $14 \cdot 10^6$ voxels and 200 parts, whereas the PLANE dataset consists of about $18 \cdot 10^6$ voxels and 10,000 parts. The SEQUOIA dataset is composed of about 3,500 rasterized polygons approximated by $50 \cdot 10^6$ voxels. The CAR data space is of size 2^{33}, the PLANE data space is of size 2^{42} and the SEQUOIA data space is of size 2^{34}.

2.3 Analysis of Temporal Data

In the past decades, time series have become an increasingly prevalent type of data. As a result, a lot of work on similarity search in time series databases has been published. The analysis of time series data, in particular, the recognition of relationships in time series databases that have not previously been discovered, is of great practical importance in many application areas. Such applications include the stock market, astronomy, environmental analysis, molecular biology, and pharmacogenomics. As a consequence, a lot of research has recently focused on similarity search in time series databases.

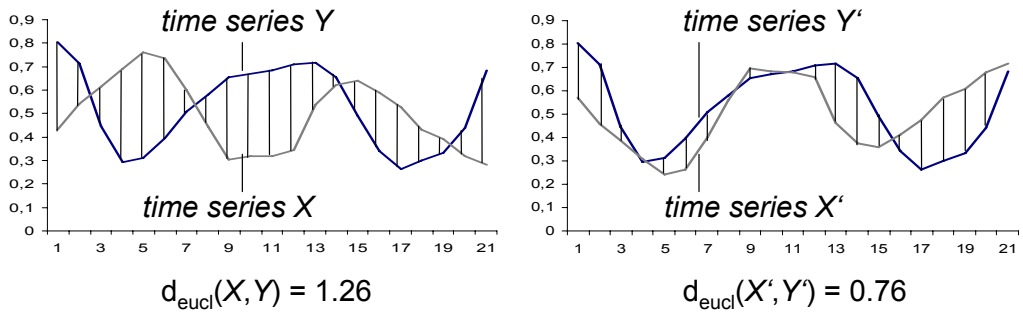


Figure 2.9: Euclidean distance between time series.

2.3.1 Measuring Similarity

The complex nature of time series represents a big challenge for effective and efficient search algorithms. The proposed methods mainly differ in the desired type of similarity, the type of application and the form of representation used for the time series objects. A survey is given in [KCMP01]. In the following, we review existing solutions for efficient similarity search on time series.

The most prominent (dis)similarity measure for time series is the Euclidean distance, which is defined as follows:

$$d_{eucl}(X,Y) = \frac{1}{N}\sqrt{\sum_{i=1..N} |x_i - y_i|^2}.$$

An example is shown in Figure 2.9. The Euclidean distance between time series X and Y is about 1.26 and the distance between X' and Y' is about 0.76. Consequently, time series X' and Y' are more similar than X and Y with respect to the Euclidean distance measure. Comparing the two pairs of time series in our example, the Euclidean distance conforms with our intuitive perception of similarity.

For many applications, the Euclidean distance may be too sensitive to minor distortions in the time axis. It has been shown that Dynamic Time Warping (DTW) [BC94] can fix this problem [KCMP01]. Using DTW to measure the distance between two time series X and Y, each value of X is matched with the best fitting value of Y based on certain constraints. The

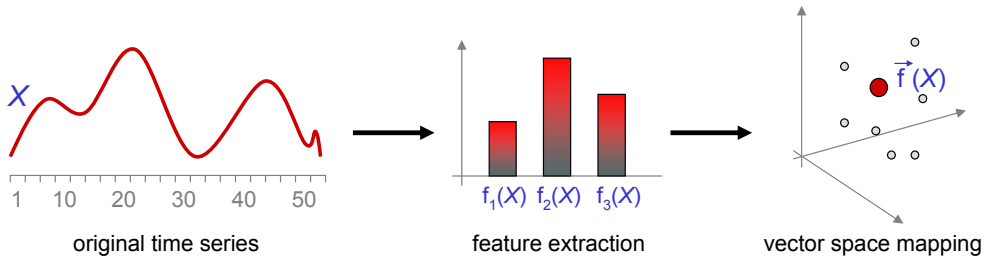


Figure 2.10: Feature based dimensionality reduction (GEMINI approach).

DTW distance can be computed by means of dynamic programming.

Indexing Time Series and Dimensionality Reduction Methods

Usually, time series are considered as points in n-dimensional space and any L_p-norm, e.g. the Euclidean distance, is used to measure the similarity between two time series. In this manner, time series can be indexed by spatial access methods, such as the R-tree and its variants [Gut84]. Nevertheless, most spatial access methods degrade rapidly with increasing data dimensionality due to the "*curse of dimensionality*". In order to utilize existing spatial access methods conveniently for time series, it is necessary to involve dimensionality reduction methods combined with the concept of multi-step query processing, as proposed in the GEMINI approach [FRM94]. Standard techniques for dimensionality reduction have been successfully applied to similarity search in time series databases, including Discrete Fourier Transform (DFT) [AFS93], Discrete Wavelet Transform (DWT) [CF99], Piecewise Aggregate Approximation (PAA) [YF00], Singular Value Decomposition (SVD) [KJF97], Adaptive Piecewise Constant Approximation (APCA) [KCMP01], and Chebyshev Polynomials [CN04]. All these methods qualify for the GEMINI framework since they provide similarity-distance measures in the reduced vector space as long as they lower bound the desired similarity-distance measure applied to the original time series.

The general idea of the GEMINI approach is to extract a few key features for each time series and map each time sequence X of length n to a point $f(X)$ in a lower dimensional feature space $\mathbb{R}^{n'}$ ($n' << n$) which can be

efficiently handled by spatial access methods. The transformation chain of GEMINI is depicted in Figure 2.10.

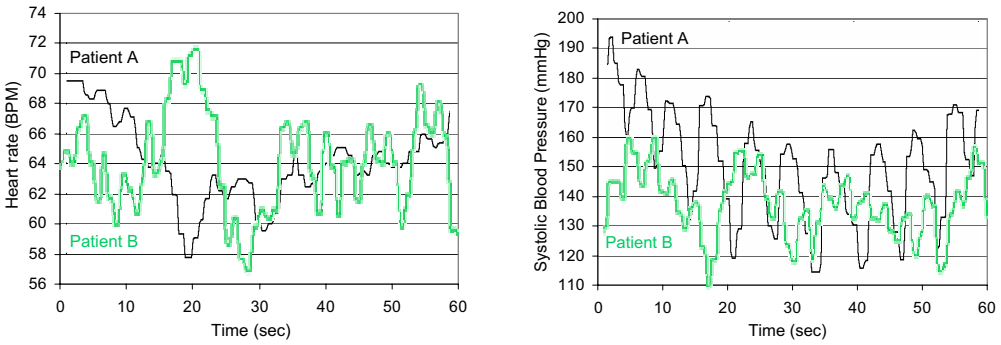
In [RKBL05] a novel bit level approximation of time series for similarity search and clustering is proposed. Each value of the time series is represented by a bit. The bit is set to 1 if the value of the time represented by the bit is strictly above the mean value of the entire time series, otherwise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

Threshold-based Similarity

In contrast to these traditional approaches that consider the course of the time series for the purpose of matching, coarse trend information about the time series could be sufficient to solve the above mentioned problem. In particular, temporal dependencies in time series can be detected by determining the points of time at which the time series exceeds a specific threshold. In this thesis, we introduce the novel concept of *threshold queries* in time series databases which report those time series exceeding a user-defined query threshold at similar time frames compared to the query time series. We present a new efficient access method which uses the fact that only partial information of the time series is required at query time.

Though the type of time series representation in [RKBL05] is quite similar to our threshold-based representation, it does not meet our needs. Furthermore, this kind of representation is restricted to a fixed threshold and, in contrast to our approach, does not allow the user to define the threshold at query time.

Moreover, 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 original 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 directly because they still represent the exact



(a) heart rate (b) systolic blood pressure

Figure 2.11: Patients heart rate and systolic blood pressure after drug treatment.

course of the time series rather than intervals of values above a threshold.

To the best of our knowledge, threshold-based analysis in time series databases has not been addressed before in the database community. In particular, there exists neither an access method for time series nor any similarity search technique which supports efficient threshold queries.

2.3.2 Application Ranges for Threshold Queries

The novel concept of threshold queries is an important technique, useful for many practical application areas.

Application 1

For the pharmaceutical industry it is interesting which drugs cause similar effects in the blood values of a patient. Obviously, effects like a certain blood parameter exceeding a critical level τ are of particular interest. We assume that after a certain drug treatment the heart rate and systolic blood pressure of several patients are measured for one minute, as shown in Figure 2.11, and the data were stored within a database. In our example, the recorded data of patient A shows an immediate effect of the drugs, which differs significantly

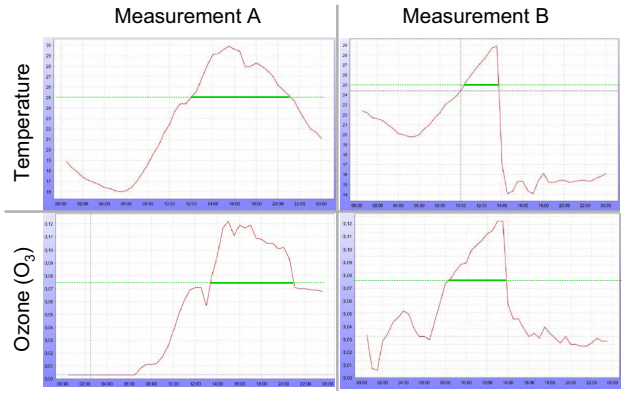


Figure 2.12: Detection of associations between different environmental and climatical attributes.

from the effects on patient B. A threshold query could return for a certain patient all other patients in the database whose heart rates and blood pressures show similar temporal reaction on the medical treatment with respect to certain thresholds which may be significant for the observed attributes.

Application 2

The amount of time series data, derived from environment observation centers, for example, has increased drastically. Furthermore, modern sensor techniques enable the user to record many attributes of the observed objects or scenes simultaneously. For instance, the analysis of environmental air pollution has been the focus of many European research projects in recent years. Many sensor stations have been installed at different locations in European cities and in rural areas. Each sensor station is equipped with several types of sensors that are used to measure multiple air pollution attributes (e.g. $SO_2, NO, NO_2, CO, BTX, O_3, H_2S$ and $C_mH_n - O$) as well as meteorological parameters such as wind direction, speed and temperature. As a result, German state offices for environmental protection maintain about 127 million time series, each representing the daily course of air pollution parameters. The gathered data are stored in terms of time series which have to be analyzed. Geo- and environmental scientists could be interested

in the dependencies that exist between meteorological attributes, e.g. humidity, and environmental attributes, e.g. particulate matter (PM_{10}). To discover which attributes nearly simultaneously exceed their legal threshold could help to find such dependencies. Hence, an effective and efficient processing of queries like "return all ozone time series that exceed the threshold $\tau_1 = 75\mu g/m^3$ when the temperature reaches the threshold $\tau_2 = 25°C$" could be very valuable. An example is depicted in Figure 2.12, showing two pairs of temperature-ozone curves where the characteristic of the ozone concentration (lower time series) is very similar to that of the corresponding temperature (upper time series) w.r.t. τ_1, τ_2 respectively. Analysis based on such similarity is provided by threshold queries. Obviously, the increasing amount of data to be analyzed represents a big challenge for methods supporting efficient threshold queries.

Application 3

The analysis of gene expression data is important for understanding of gene regulations and cellular mechanisms in molecular biology. Gene expression data contains the expression level of thousands of genes, indicating how *active* one gene is over a set of time slots. 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 solved by means of threshold queries with a threshold of $\tau = 0$. Each gene provides its own interval sequence, indicating the time slots classified as "up". Genes with a similar interval sequence have a similar "up" and "down" pattern.

2.3.3 Test Datasets

Our experimental evaluation of the threshold similarity is based on a wide variety of test datasets. In order to guarantee the reproducibility of the

experiments and to compare the results to other approaches, we used several publicly available datasets, mostly from the UCI KDD Archive[1], which we describe in the following.

The ***AUDIO*** dataset contains time sequences expressing the temporal behavior of the energy, the dynamics and the strongest peak in pieces of music. The three representations are computed 25 times per second for 6 octaves by using a cascade of bandpass filters. The resulting time series are then cut into pieces of length 300, resulting in an overall database of 700,000 time series. If not otherwise stated, the database size was set to 50,000 objects and the length of the objects was set to 50. This dataset is used to evaluate the performance of our approach (cf. Section 13.3).

The ***SCIENTIFIC*** datasets are derived from two different applications:

- the analysis of environmental air pollution (*SCIEN_ENV*) and
- gene expression data analysis (*SCIEN_GEX*).

The data on environmental air pollution is derived from the Bavarian State Office for Environmental Protection, Augsburg, Germany[2] and contains the daily measurements of 8 sensor stations distributed in and around the city of Munich, Germany from the year 2000 to 2004. One time series represents the measurement of one station at a given day, and contains 48 values for one of 10 different parameters such as temperature, ozone concentration etc.

The gene expression data from [SSZ+98] contains the expression level of approximately 6,000 genes measured at 24 different time slots.

The ***STANDARD*** datasets are derived from diverse fields and cover the complete spectrum of data characteristics, including stationary/non-stationary, noisy/smooth, cyclical/non-cyclical, symmetric/asymmetric etc. They are available from the UCR Time Series Data Mining Archive [KF02]. Due to their variety, they are often used as a benchmark for novel approaches in the field of similarity search in time series databases. We used the following

[1] kdd.ics.uci.edu/

[2] www.bayern.de/lfu

four datasets: GUN/POINT (*GunX*), TRACE (*Trace*), CYLINDER-BELL-FUNNEL (*CBF*) and CONTROL CHART (*SynCtrl*).

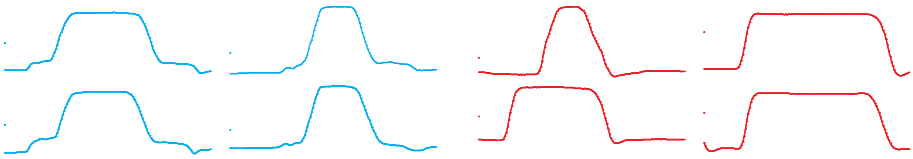


Figure 2.13: Example time series taken from the *GunX* dataset.

The ***GunX*** dataset is a two-class dataset which comes from the video surveillance domain [KR04]. It has two classes, each containing 100 instances. All instances were created by using one female actor and one male actor in a single session. The two classes are:

- Gun-Draw: The actors have their hands by their sides. They draw a replica of a gun from a hip-mounted holster, point it at a target for approximately one second, then return the gun to the holster, and their hands to their sides.
- Point: The actors have their hands by their sides. They point with their index fingers to a target for approximately one second, and then return their hands to their sides.

For both classes, we tracked the centroid of the right hand in X-axes. Each instance has a length of 150 data points and is z-normalized (i.e., $\mu = 0$, $\sigma = 1$).

The examples of this dataset, which are depicted in Figure 2.13, exhibit the classification-problem of this dataset: the actual time for pointing at the target greatly varies within the allowed time frame of one second. This inconsistency poses a challenge for classifying this dataset. It is often the case that the classification is based on the length of the pointing time interval, and not on the slight irregularities when drawing the gun.

The ***Trace*** dataset is a four-class dataset which is a subset of the Transient Classification Benchmark (trace project) used in [Rov02] for nuclear

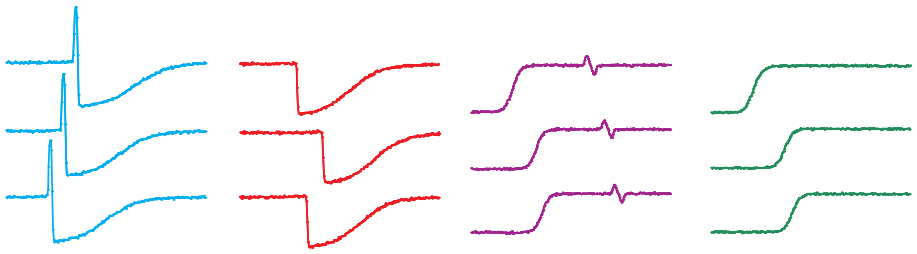


Figure 2.14: Example time series taken from the *Trace Data* dataset.

power plant malfunction diagnostics. It is a synthetic dataset designed by Davide Roverso 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 classes 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, and are z-normalized.

Figure 2.14 depicts some examples from this dataset. It is clearly visible that the time series within a class are relatively similar, but are heavily shifted along the time axis.

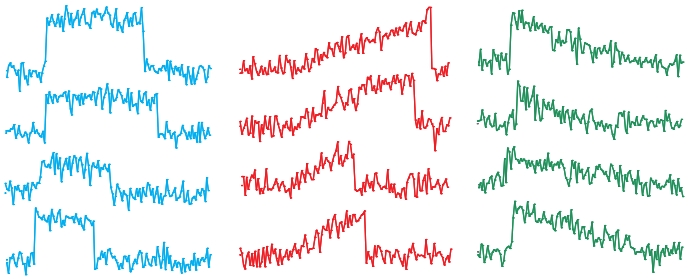


Figure 2.15: Example time series taken from the *Cylinder-Bell-Funnel* dataset.

The ***CBF*** dataset is an artificial dataset that was defined by Saito in [Sai94] and later used in several other publications (cf. [Geu01]). It consists of three classes *cylinder*, *bell* and *funnel* which are defined by the following functions:

$$
\begin{aligned}
c(t) &= (6+\eta)\cdot\chi_{[a,b]}(t) + \epsilon(t) \\
b(t) &= (6+\eta)\cdot\chi_{[a,b]}(t)\cdot\frac{t-a}{b-a} + \epsilon(t) \\
f(t) &= (6+\eta)\cdot\chi_{[a,b]}(t)\cdot\frac{b-t}{b-a} + \epsilon(t) \\
\text{with} & \\
\chi_{[a,b]}(t) &= \begin{cases} 1 & \text{if } a \le t \le b \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

The values for η and $\epsilon(t)$ are standard normal variates, a and b are uniformly distributed integers in the range $[16, 32]$, respectively $[64, 128]$. For our experiments, we generated a *CFB* dataset containing 50 time series of each class.

The biggest problem when classifying this dataset is the rather strong noise which is added on the time series and the large window where the characteristic feature of each class can be located (cf. Figure 2.15).

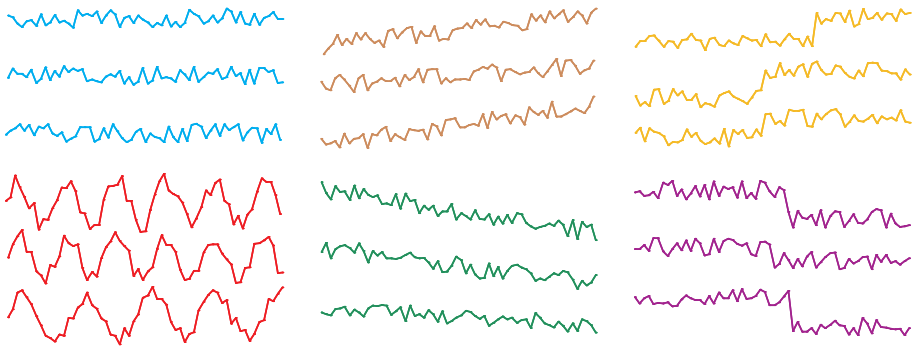


Figure 2.16: Example time series taken from the *Synthetic Control* dataset.

The ***SynCtrl*** dataset was created by Alcock and Manolopoulos for [AM99] and contains 600 examples of synthetically generated control charts. It consists of the cyclic pattern subset of the control chart data from the UCI KDD archive. The data is effectively a sine wave with noise consisting of 6,000 data points. There are six different classes (100 instances per class) of control charts: *normal cyclic*, *increasing trend*, *decreasing trend*, *upward*

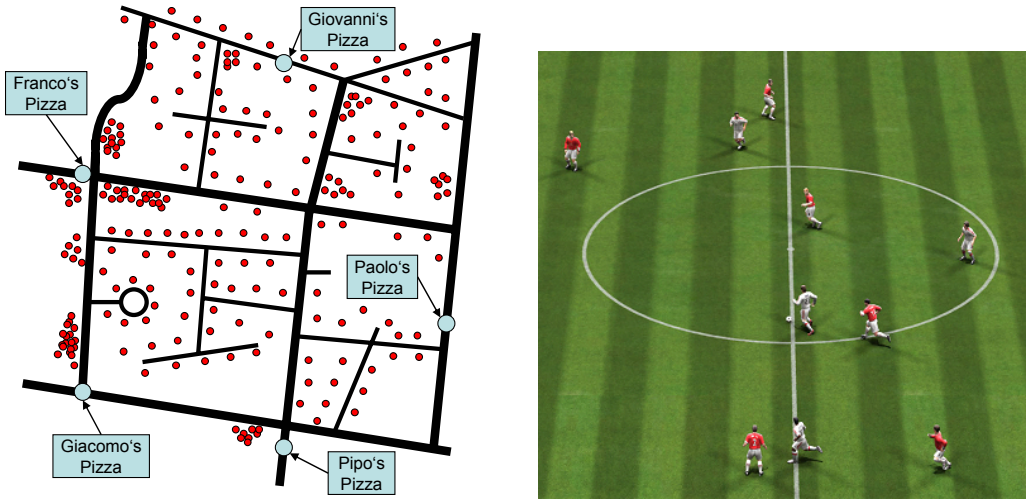
shift and *downward shift*. Figure 2.16 depicts three representations each from these classes.

The biggest challenge for most distance measures is to distinguish between an *increasing trend* and an *upward shift*, resp. a *decreasing trend* and *downward shift*.

2.4 Analysis using RkNN Queries

A reverse k-nearest neighbor (RkNN) query returns the data objects that have the query object in the set of their k-nearest neighbors. It is the complimentary problem to that of finding the k-nearest neighbors (kNN) of a query object and has been studied extensively in the past few years for Euclidean data [KM00, SAA00, YL01, SFT03]. The goal of a reverse k-nearest neighbor query is to identify the "influence" of a query object on the whole dataset.

For example, consider a decision support system with the goal of choosing the location for a new store. Given several choices for the new location, the strategy is to pick the location that can attract the most customers. A RkNN query would return the customers who would be likely to use the new store because of its geographical proximity. The RkNN problem appears in many practical situations such as geographic information systems (GIS), traffic networks, or molecular biology where the database objects are general metric objects rather than Euclidean vectors. In these application areas, the database objects are polygons or sequences and an arbitrary metric distance function is defined on these objects to evaluate object similarity. For example, the increasing progress in telecommunication techniques and location tracking systems like GPS, significantly extends the scope for location-based service applications. Beside the nearest neighbor search, the reverse nearest neighbor query is one of the most important query types for location based services, e.g. in applications where stationary or moving objects agree to provide some kind of service to each other. Objects or individuals usually want to request services from their nearest neighbors. Conversely, the objects or



(a) Merchandising (b) Game tactics

Figure 2.17: Applications for RkNN queries.

individuals which provide some services may be interested in the number of expected service requests, or which objects could be interesting candidates to provide the services, and maybe, where they are actually located. For example, all filling stations of one company in a town want to provide their own advertisements to all cars of which they are the nearest neighbor.

Another example consisting of pizza restaurants (large dots) and potential customers (small dots) is depicted in Figure 2.17(a). To keep down costs when carrying out an advertising campaign, it would be profitable for a restaurant owner to send menu cards only to those customers which have his restaurant as the nearest pizza restaurant. RkNN queries can also be useful to improve the game tactics in real-life and video games, as shown in Figure 2.17(b). To identify possible uncovered players, and therefore to improve the strategy of defense, each player should always closely watch his reverse nearest neighbors on the opposing team.

In many cases, the objects are nodes in a traffic network. Instead of the Euclidean distance, graph algorithms like Dijkstra have to be applied. Another important application area of RkNN search in general metric databases is molecular biology. Here, the detection of new sequences call for efficient

solutions of the general RkNN problem in sequence databases. These databases usually contain a large number of biological sequences. Researchers all over the world steadily detect new biological sequences that need to be tested for originality and importance. To decide about the originality of a newly detected sequence, the RkNNs of this sequence are computed and examined. Usually, in this context, the similarity of biological sequences is defined in terms of a metric distance function such as the edit distance or the Levenstein distance. More details on this application of RkNN search in metric databases can be found in [DP03].

2.4.1 RkNN Search in Euclidean Space

All RkNN methods proposed so far are only applicable to Euclidean vector data, i.e. $\mathcal{D}$ contains feature vectors of arbitrary dimensionality d ($\mathcal{D} \in \mathbb{R}^d$).

In [KM00], an index structure called RNN-Tree is proposed for reverse 1-nearest neighbor search. The basic idea is that if the distance from an object p to the query q is smaller than the 1-nearest neighbor distance of p, then p can be added to the result set. This approach saves a nearest neighbor query w.r.t. p. Thus, the RNN-Tree stores for each object p the distance to its 1-nearest neighbor, i.e. $nndist_1(p)$. In particular, the RNN-Tree does not store the data objects itself but for each object p a sphere with radius $nndist_1(p)$. Thus, the data nodes of the tree contain spheres around objects, rather than the original objects. The spheres are approximated by minimal bounding rectangles (MBRs). Since the tree suffers from a high overlap of the data MBRs and, thus, from a high overlap of the directory MBRs, [KM00] propose to use two trees: (1) a traditional R-Tree-like structure for nearest neighbor search (called NN-Tree) and (2) the RNN-Tree for reverse 1-nearest neighbor search.

The RdNN-Tree [YL01] extends the RNN-Tree by combining the two index structures (NN-Tree and RNN-Tree) into one common index. It is also designed for reverse 1-nearest neighbor search. For each object p, the distance to p's 1-nearest neighbor, i.e. $nndist_1(p)$ is precomputed. In general,

the RdNN-Tree is a R-Tree-like structure containing data objects in the data nodes and MBRs in the directory nodes. In addition, for each data node N, the maximum of the 1-nearest neighbor distance of the objects in N is aggregated. An inner node of the RdNN-Tree aggregates the maximum 1-nearest neighbor distance of all its child nodes. A reverse 1-nearest neighbor query is processed top down by pruning those nodes N where the maximum 1-nearest neighbor distance of N is greater than the distance between query object q and N, because in this case, N cannot contain true hits anymore. Due to the materialization of the 1-nearest neighbor distance of all data objects, the RdNN-Tree needs not to compute 1-nearest neighbor queries for each object.

A geometric approach for reverse 1-nearest neighbor search in a 2D dataset is presented in [SAA00]. It is based on a partition of the data space into six equi-sized units where the gages of the units cut at the query object q. The nearest neighbors of q in each unit are determined and merged together to generate a candidate set. This considerably reduces the cost for the nearest-neighbor queries. The candidates are then refined by computing for each candidate c the nearest neighbor. If this nearest neighbor is q then c is added to the result. Since the number of units in which the candidates are generated increases exponentially with d, this approach is only applicable for 2D datasets.

An approximative approach for reverse k-nearest neighbor search in higher dimensional space is presented in [SFT03]. A two-way filter approach is used to generate the results. However, the method cannot guarantee the completeness of the result, but trades loss of accuracy for a gain of performance.

Recently, in [TPL04], the first approach for reverse k-nearest neighbor search that ensures complete results, was proposed. The method uses any hierarchical tree-based index structure such as R-Trees to compute a nearest neighbor ranking of the query object q. The key idea is to iteratively construct a Voronoi cell around q from the ranking. Objects that are beyond k Voronoi planes w.r.t. q can be pruned and need not to be considered for Voronoi construction. The remaining objects must be refined. For each of

Figure 2.18: Road network graph of the city of Oldenburg.

these candidates, a k-nearest neighbor query must be launched.

We propose the first approach for efficient RkNN search in arbitrary metric spaces where the value of k is specified at query time. Our approach uses the advantages of existing metric index structures but proposes to use conservative and progressive distance approximations in order to filter out true drops and true hits. In particular, we approximate the k-nearest neighbor distance for each data object by upper and lower bounds using two functions of only two parameters each. Thus, our method does not generate any considerable storage overhead.

2.4.2 Test Datasets

Our RkNN experiments are based on several real-world datasets. For the metric RkNN search, we used a road network dataset derived from the city of *Oldenburg*, which contains 6,105 nodes and 7,035 edges. The average degree of the nodes in this network is 1.15. The dataset is available online[3]. The nodes of the network graph were taken as database objects from which

[3] www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/

subsets of different size were selected to form the test data set. For the distance computation we used the shortest-path distance computed by means of the Djikstra algorithm. The Oldenburg road network graph is depicted in Figure 2.18.

The used Euclidean datasets include a set of 5-dimensional vectors generated from the well-known SEQUOIA 2000 benchmark dataset and two "Corel Image Features" benchmark datasets from the UCI KDD Archive. One dataset contains 9 values for each image, the other dataset contains 16-dimensional texture values.

2.5 Analysis of Video & Audio Data

With the rapid development of digital technologies, computer networks and the Internet, the amount of multimedia data is growing enormously. This is mainly possible because digital images, videos and pieces of music are easily copied and distributed. Thus, multimedia databases are employed in order to store such audio and video data. Typical applications in the area of multimedia databases are content-based retrieval and automatic annotation. While content-based retrieval systems are often unsupervised and do not need any assistance from the user besides a query object, the quality of the annotations usually improves if some training data is provided. If a classifier is used in the annotation process which has been trained by user-labeled data, the generated annotations are more likely to match the user's expectations.

A typical MMDBS creates additional challenges due to the nature of multimedia data and the requirements of possible applications. We can summarize these challenges as follows. A MMDBS must support storage of large objects, because multimedia data such as audio or video can require gigabytes of storage. Thus, special preprocessing, storage and similarity search techniques are needed. Furthermore, the availability of versatile aspects and different feature transformations for multimedia data leads to demand for handling multiple, heterogeneous descriptions in retrieval and analysis methods.

Video clips are an important type of multimedia data. Due to recent technical advances, the amount of video data that is available in digital formats as well as the possibility to access and display such video files has increased enormously. The approach in this thesis focuses on the following scenario: given a database of movies or video clips, we want to retrieve all movies from the database that are similar to the given query video. For this type of scenario, there are various applications. For example, a company wants to determine if a given video podcast or shared video file is similar to a copyrighted movie or video clip. In this scenario, the company would store all of its movies in a database and automatically check if the the video podcast matches to a video in the database. Another example is a database of news programs recorded on various days from various TV stations. A user can retrieve all news programs that are likely to contain a given video clip featuring a particular event. Since most news programs use videos that are provided by video news agencies, it is very likely that the news programs dealing with similar topics contain similar news clips.

In addition to the content-based retrieval of similar objects it is often useful to assign objects to different classes, which helps to manage them more effectively. With this technique, only objects in the same or similar classes have to be analyzed in order to locate similar objects. This can be achieved in a preprocessing step which identifies classes that contain a certain number of similar objects. Another possibility would be to assign the given multimedia objects to predefined classes, which can either reflect the personal preferences of a user or be created by some experts. The process of assigning objects to these classes is then typically performed by classification algorithms.

2.5.1 Video Retrieval

Efficient and effective similarity search in such huge amounts of multimedia data has become a major issue in several important applications such as video copyright matters and multimedia retrieval [TKR99]. In fact, video similarity detection has been proposed as a promising approach for copyright issues

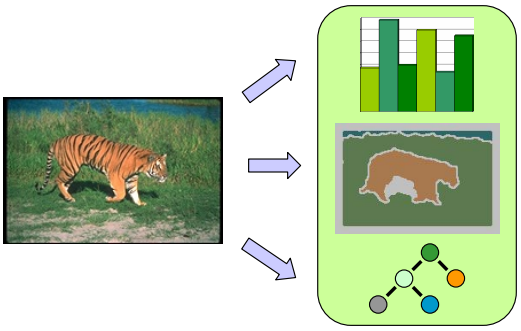


Figure 2.19: An image described by multiple representations.

which is complementary to the approach of digital watermarking [HB01]. In addition, video similarity search is the key step towards content-based video retrieval. As a consequence, a large amount of work has been directed toward the field of similarity search in multimedia databases [CZ02, IL00, NWH01, TKR99].

Multimedia data such as movies can usually be viewed as multi-represented objects, i.e. for each object there are multiple representations modeling different features of the object. For example, for music videos, we can collect audio features, such as pitch [TK00] or rhythm [TC02], and video features, such as color histograms or textures [AY99]. Each of these multiple representations models a different aspects of a music video. Figure 2.19 depicts an image which is described by multiple representations: a color histogram, a shape descriptor, and a segment tree. Obviously, the effectiveness of similarity search methods could greatly benefit from taking multiple representations into account. However, most existing approaches for multimedia similarity search do not consider the multi-represented structure of multimedia objects but usually use only one representation for similarity search.

Usually, multimedia objects consist of thousands or even millions of feature vectors. In order to handle such data efficiently, summarization techniques are usually applied to the original data, so that the original feature vectors are grouped together and each group is represented by a summarization vector or summarization representative. Similarity is then defined between these summarizations or the according summarization representa-

tives.

2.5.2 Summarization Techniques

In general, we can distinguish two classes of summarization techniques: higher-order and first order summarization.

Higher-order summarization techniques are usually generated by applying optimization algorithms on feature vectors. They describe a multi-instance object as a mix of statistical distributions or cluster representatives. In [GGM02], a higher-order summarization technique is presented which is based on Gaussian distributions or mixtures of Gaussian distributions. The authors use methods such as Expectation-Maximization for parameter estimation [GGM02]. The resulting summarizations are represented by Gaussian distributions. The Kullback-Leibler distance can be used to compute the distance between two Gaussian distributions [IL00]. The authors of [CSL99] propose an approach for obtaining a compact representation of videos that computes the optimal representatives by minimizing the Hausdorff distance between the original video and its representation. If the Euclidian metric is used as distance function on the feature vectors, the k-means method can be applied for the summarization of video clip content [ZRHM98]. K-means minimizes the variance w.r.t. the representative vectors, this function is also called TD^2. For the case of general metric spaces, the k-medoid method can be applied for summarization. This method minimizes the distance between a video and its description.

First-order techniques calculate a small set of representative feature vectors as summarization vectors in order to describe a video. A randomized technique for summarizing videos, called video signature, is proposed in [CZ02]. A video sequence in the database is described by selecting a number of its frames closest to a set of random vectors. This method requires only a single scan over the video or audio sequences, and delivers a compact and reliable description that can be used for similarity search. The authors in [CZ02] also propose a specialized distance function on the derived first-order

summarization vectors.

2.5.3 Similarity Search Based on Multiple Representations

Recently, some work has been done on multi-represented similarity search in multimedia databases. The interactive search fusion method [SJL+03] provides a set of fusion functions that can be used for combining different representations. This method supports a manual and an interactive search that is supervised by the user's assistance or by a user-defined query. In addition, boolean operators on aggregation functions are supported, e.g. "AND" can be applied to the product aggregation function. Let us note that this technique is supervised. It requires strong interaction with the user, which is not always desirable since it requires the user to understand the basic concepts of the method. Moreover, the proposed technique does not support individual weighting for each query object. In [NWH01], a template matching method based on the time warping distance is presented. This approach can measure the temporal edit similarity. However, temporal order is not necessary in many applications. In addition, this technique is not applicable to large databases because it is linear in the number of feature vectors of all video and audio sequences in the database.

From a technical point of view, video data consists of a sequence of images, so-called frames, that might be accompanied by some soundtracks. In our approach, we exploit both the image- and the sound-track. To allow similarity search on video clips, each frame is represented by a several feature vectors. For example, we use color histograms and texture features for the image part of the video, i.e. our novel approach for similarity search in multimedia databases takes multiple representations of multimedia objects into account. In particular, we present weighting functions to rate the significance of a feature of each representation for a given database object. This allows to weight each representation during query processing. Let us note that the same video taken from different sources might be described by a considerably different set of feature vectors due to varying encoding quality.

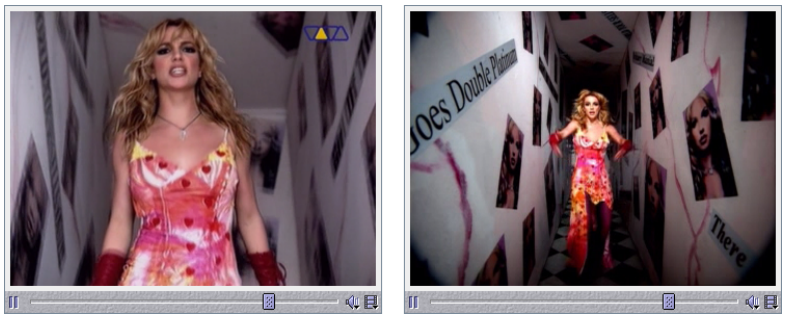


Figure 2.20: Screenshots of similar music videos.

In Figure 2.20, two screenshots of similar music videos are depicted. The left hand image was taken from a TV broadcast and contains a TV station logo, while the image on the right hand side comes from a DVD version and has a higher resolution and brighter colors.

2.5.4 Test Dataset

We evaluated our concepts using a database of 500 music videos collected by a focused web crawler. The video collection is herein referred to as ***MUSICVIDEO*** from now on. We extracted the image representations of the videos on a per-frame basis, i.e. we generated 25 features/second for PAL and 30 features/second for NTSC videos. From each image, we extracted four representations, namely a color histogram and three textural features. For the color histogram, we used the HSV color space (cf. Figure 2.21). In order to be independent from the brightness of the color, we discarded the Value (V) information. The idea is to use a feature representation which is somewhat insensitive to the recording media. Old recordings that have been digitized from a VHS tape often show a loss of brightness or brightness fluctuations. So we divided the HSV color space into 32 subspaces, 8 ranges of hue and 4 ranges of saturation. The textural features were generated from 16 gray-scale conversions of the images. We computed contrast, entropy and inverse difference moment using the co-occurrence matrix [HSD73]. For extracting the audio features, we divided the audio signal of a video into short time frames, each having a length of 1/50 second. Every audio frame is rep-

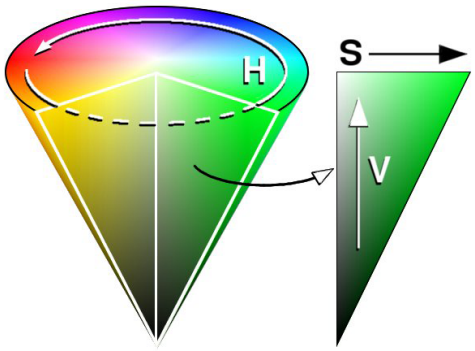


Figure 2.21: Conical representation of the HSV color space.

resented by two features in the time- and frequency-domain. We computed autocorrelation and threshold-crossing for the time-domain, spectral flux and mel-frequency cepstral coefficients for the frequency-domain [TC02].

2.5.5 Audio Classification

Recently, powerful music compression tools and cheap mass storage devices have become widely available. This allows average consumers to transfer entire music collections from distribution media, such as CDs and DVDs, to their computer hard drive. To locate specific pieces of music, they are usually labeled with artist and title. Yet the user would benefit from a more intuitive organization based on the music style to get an overview of the music collection. This means there is a need for content-based music classification methods to organize these collections automatically using a given genre taxonomy. To provide a versatile description of the music content, several kinds of features like rhythm, pitch or timbre characteristics are commonly used. Taking the highly dynamic nature of music into account, each of these features should be calculated up to several hundred times per second. Thus, a piece of music is represented by a complex object given by several large sets of feature vectors.

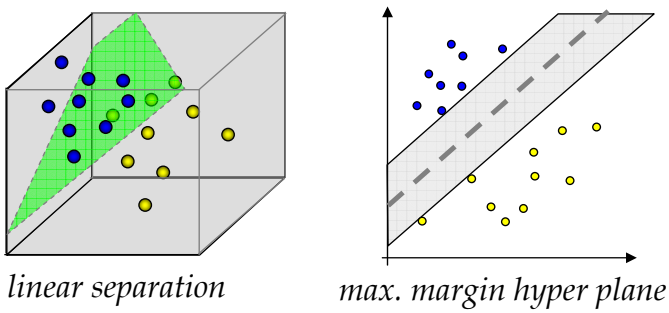


Figure 2.22: Basic idea of a Support Vector Machine (SVM).

Support Vector Machines

In recent years, *support vector machines* (SVMs) [CV95] have received much attention since they offer superior performance in various applications. For example, [WLCS04] presents a fusion technique for multimodal objects. Basic SVMs use the idea of linear separation of two classes in feature space and distinguish between two classes by calculating the maximum margin hyperplane between the training examples of both given classes as illustrated in Figure 2.22. To employ SVMs for distinguishing more than two classes, several approaches were introduced [PCST99]. In order to handle sets of feature vectors in SVMs so called kernel functions were introduced [GFKS02]. A weakness of multi-instance kernels is the need to calculate distances between all instances, i.e. $O(n^2)$ single distance calculations are required in order to compare two multi-instance objects with n instances. Thus, multi-instance kernels seem to be unsuitable for solving large-scale classification problems in music collections.

Instance Reduction Techniques

As mentioned above, a piece of music is usually described by a set of feature vectors and is an multi-instance object. The number of instances can vary from tens to hundreds per second, i.e. a song is represented by 10,000 to 50,000 feature vectors. Similar to video data, we can apply higher-order and first order instance reduction techniques on multi-instance audio objects.

Both first and higher-order techniques reduce a multi-instance object to a small set of feature vectors. In order to process the reduced set-valued representations of a multi-instance object by an SVM, special kernel functions are required. As indicated before, the use of kernel functions seems impractical for efficient classification in the context of large databases.

Hierarchical Classification

The general idea of hierarchical classification is that a classifier located on an inner node solves only a small classification problem and therefore achieves more effective results more efficiently than a classifier that works on a large number of flat organized classes. The use of class hierarchies to improve large-scale classification problems has predominantly been applied in text classification. Several approaches have been introduced picking up this idea. The authors of [KKPS04] investigated multiple representations of objects in the context of hierarchical classification and proposed an *object adjusted weighting* for a linear combination of multi-represented objects.

There exist only a few approaches for automatic genre classification of audio data. In [CVK04], music pieces are classified into either rock or classical music using k-NN and MLP classifiers. An approach for hierarchical genre classification which does not support user feedback is presented in [TC02]. Zhang [Zha03] proposes a method for a hierarchical genre classification which follows a fixed schema, and there is only limited support for user-created genre folders. Moreover, all of above mentioned hierarchical classification methods do not take full advantage of multi-instance and multi-represented music objects.

In contrast, we present an approach that handles such rich object representations as well as an arbitrary genre hierarchy, deals with user feedback and supports multi-assignment of songs to classes. We also demonstrate the practical relevance of our approach in a prototype called MUSCLE.

2.5.6 Test Dataset

A music collection consisting of almost 500 songs was the basis for the classification experiments. This collection is herein referred to as ***SONGDATA***. The *SONGDATA* dataset contains musical pieces from 15 different classes, which results in approximately 30 songs per class. Depending on the representation, we extracted 30 to 200 features per second.

Timbre features are derived from the frequency domain and were mainly developed for the purpose of speech recognition. The extraction of the timbral texture is performed by computing the short time Fourier transform. We use the Mel-frequency cepstral coefficients (MFCCs), spectral flux and spectral rolloff as timbral representations [TC02]. Rhythmic content features are useful for describing the beat frequency and beat strength of a piece of music. In our framework, we use features derived from beat histograms [TC02] as the description of the rhythmic content. Pitch extraction tries to model the human perception by simulating the behavior of the cochlea. Similar to the rhythmic content features, we derive pitch features from pitch histograms which were generated by a multipitch analysis model [TK00].

Part II

Analysis of Spatial Data

Chapter 3

Introduction

The efficient management of spatially extended objects has become an enabling technology for many novel database applications including geographical information systems (GIS), computer-aided design (CAD), medical imaging, molecular biology or haptic rendering. As a common and successful approach, spatial objects can conservatively be approximated by a set of voxels, i.e. cells of a grid covering the complete data space. By expressing spatial region queries as an intersection of the corresponding voxel sets, vital operations for two-dimensional GIS and environmental information systems [MP94] can be supported. Efficient and scalable database solutions are also required for three-dimensional CAD applications to cope with rapidly growing amounts of dynamic data. Such applications include the digital mock-up of vehicles and airplanes, and virtual reality applications, e.g. haptic simulations in virtual product environments [MPT99]. For instance, the "777" from boeing was completely digitally designed and assembled. It consists of about three million parts, whereby some of these parts are composed of several millions of voxels. Although the voxels can further be grouped together to intervals, the number of the resulting spatial primitives still remains very high. On the other hand, one-value approximations of spatially extended objects often are far too coarse. In many applications, GIS or CAD objects feature a very complex and fine-grained geometry. The rectilinear bounding box of the brake line of a car, for example, would cover the whole bottom of the indexed

data space. A non-replicating storage of such data causes region queries to produce too many false hits that have to be eliminated by subsequent filter steps. For the above mentioned application ranges suitable index structures, which guarantee efficient spatial query processing and industrial-strength, are indispensable. Fortunately, a lot of traditional database servers have evolved into Object-Relational Database Management Systems (ORDBMS). Object types and other features, such as binary large objects (BLOBs), external procedures, extensible indexing, user-defined aggregate functions and query optimization, can be used to build powerful, reusable server-based components. Relational access methods perfectly fit to the common relational data model and are highly compatible with the extensible indexing frameworks of existing object-relational database systems. In this part, we introduce a cost-based decompositioning algorithm of complex spatial objects managed by relational index structures. Our approach takes compression algorithms for the effective storage of decomposed spatial objects and access probabilities of these decompositions into account. Thereby, it helps to range between the two extremes of one-value approximations and the use of unreasonably many approximations. For modern engineering tasks and GIS applications this new approach is highly beneficial, as it yields an excellent query response behavior along with industrial strength.

The remainder of this part is organized as follows. In Chapter 4, we introduce interval group objects, which can be stored within a spatial index. Chapter 4.2 is dedicated to the storage of these interval groups. In Chapter 5, we first discuss why it is beneficial to store the interval groups in a compressed way. Furthermore, we introduce a new spatial packer, called QSDC. In Chapter 6, we discuss in detail our cost-based grouping algorithm which can be used together with arbitrary packing algorithms but is especially useful in combination with our new Quick Spatial Data Compressor (QSDC). In Chapter 7, we discuss how intersection queries based on compressed interval groups can be posted on top of the SQL engine. In Chapter 8, we present the empirical results, which are based on three real-world test datasets. The 2D dataset was derived from the SEQUOIA 2000 benchmark [SFGM93], while the 3D datasets were provided by our industrial partners, a German car man-

ufacturer and an American plane producer. We show in a broad experimental evaluation that our new concept accelerates spatial query processing based on the Relational Interval Tree (RI-tree) [KPS01, KPS00].

Chapter 4

Cost-Based Decompositioning of Complex Spatial Objects

High resolution spatial objects may consist of several hundreds of thousands of voxels (cf. Figure 4.1). For each object, there exist a lot of different possibilities to decompose it into approximations by grouping numerous voxels together. We call these groups *interval groups* throughout the rest of this paper (cf. Figure 4.2). Informally spoken, interval groups bridge the gap between single intervals; a formal definition follows later. The question at issue is, which grouping is most suitable for efficient query processing. A good grouping should take the following "grouping rules" into consideration:

- The number of interval groups should be small.
- The dead area of all interval groups should be small.
- The interval groups should allow an efficient evaluation of the contained voxels.

The first rule guarantees that the number of index entries is small, as the hulls of the interval groups are stored in appropriate index structures, e.g. the RI-tree (cf. Figure 4.2). The second rule guarantees that many unnecessary candidate tests can be omitted, as the number and size of gaps

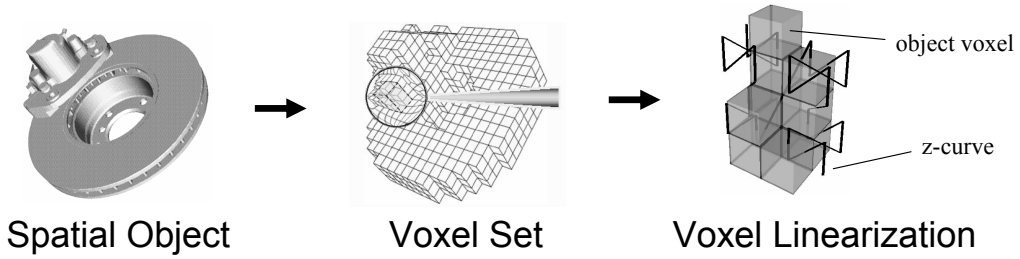


Figure 4.1: Voxel Linearization Process Chain

included in the interval groups is small. Finally, the third rule guarantees that a candidate test can be carried out efficiently. A good query response behavior results from an optimum trade-off between these grouping rules.

4.1 Interval Groups

Interval groups are formed by voxels which are used for describing complex spatial objects. We start with defining voxelized objects formally:

Definition 4.1 (voxelized object).
Let O be the domain of all object identifiers and let $id \in O$ be an object identifier. Furthermore, let $\mathbb{N}^d$ be the domain of d-dimensional points. Then we call a pair $(id, \{\nu_1, ..., \nu_n\}) \in O \times 2^{\mathbb{N}^d}$ a d-dimensional voxelized object. We call each of the ν_i an object voxel, where $i \in \{1, ..., n\}$.

A voxelized object consists of a set of d-dimensional points, which can be naturally ordered in the one-dimensional case. If $d > 1$, such an ordering does not longer exist. By means of space filling curves, $\rho : \mathbb{N}^d \rightarrow \mathbb{N}$, all multidimensional voxelized objects can be mapped to one-dimensional voxelized objects (cf. Figure 4.1).

After linearization, we can group adjacent voxels together to intervals and store these intervals in appropriate index structures, e.g. the RI-tree. If we conservatively approximate an object by only one interval, numerous *error voxels* are included in this approximation, which requires an expensive refinement step during the query process. A promising trade-off between

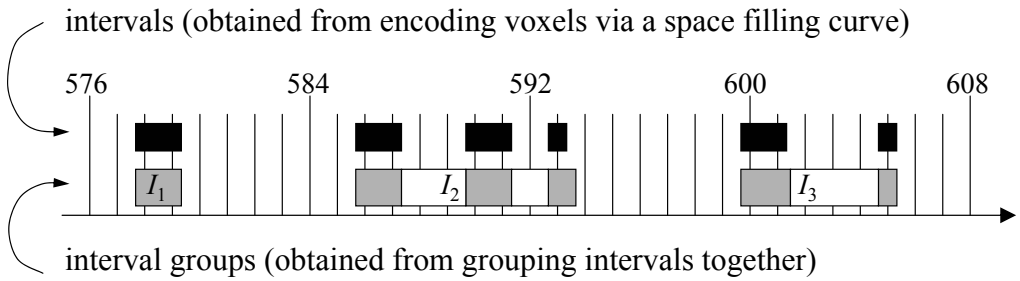


Figure 4.2: Intervals and interval groups.

the filter and refinement step can be found by means of interval groups. Intuitively, an interval group sequence is a covering of our voxelized and linearized object, where each voxel is assigned to exactly one interval group (cf. Figure 4.2).

Definition 4.2 (interval group, interval group sequence).
Let $(id, \{\nu_1, ..., \nu_n\})$ *be a* d*-dimensional voxelized object, and let* $\pi : \{1, ..., n\} \rightarrow \{1, ..., n\}$ *be a bijective function. Moreover, let* $m \leq n$ *and* $0 = i_0 < i_1 < i_2 < ... < i_m = n \in \mathbb{N}^+$. *Then we call* $O_{group} = (id, \langle\{\nu_{\pi(i_0+1)}, ..., \nu_{\pi(i_1)}\}, ..., \{\nu_{\pi(i_{m-1}+1)}, ..., \nu_{\pi(i_m)}\}\rangle)$ *an interval group sequence. We call each of the* $j = 1, ..., m$ *groups* $I_j = \{\nu_{\pi(i_{j-1}+1)}, ..., \nu_{\pi(i_j)}\}$ *of* O_{group} *an interval group.*

In the following, we present operators for interval groups which enable us to introduce our approach formally. Throughout this paper, we refer repeatedly to the definitions summarized in Table 1 which assumes that $I_{group} = \{\nu_1, ..., \nu_c\}$ is a d-dimensional interval group and $\rho : \mathbb{N}^d \rightarrow \mathbb{N}$ an arbitrary space filling curve. For clarity, Figure 4.2 demonstrates the values of the most important operators on interval groups. We use $B(I_{group})$ as an abbreviation for a byte sequence containing the complete information of the intervals which have been grouped together to I_{group}. In other words $B(I_{group})$ corresponds to a sequence of voxels within $H(I_{group})$ along the space filling curve ρ, transformed into a bit sequence of length $L(I_{group})$. A "1" bit denotes an object voxel and a "0" bit denotes a non-object voxel. For high data space resolutions, the byte-string $B(I_{group})$ might contain long sequences of zero bytes, i.e. large gaps.

In the following, we confine ourselves to non-overlapping approximations,

Table 4.1: Operators on interval groups.

Operator	Description and Definition
$H(I_{group})$	minimal covering of the interval group I_{group}
$H(I_{group}) = (l_r, u_s)$	
$L(I_{group})$	length of the interval group I_{group}
$L(I_{group}) = u_s - l_r + 1$	
$D(I_{group})$	density of interval group I_{group}
$D(I_{group}) = c/L(I_{group})$	
$G(I_{group})$	maximum gap between two intervals in I_{group}
$G(I_{group}) = \begin{cases} 0 & r = s \\ max\{l_i - u_{i-1} - 1, i = r+1, ..., s\} & \text{else} \end{cases}$	
$B(I_{group})$	byte sequence describing the interval group I_{group}
$B(I_{group}) = \langle s_0, ..., s_n \rangle$, where $s_i \in \mathbb{N}$ and $0 \leq s_i < 2^8$, $n = \lfloor u/8 \rfloor - \lfloor l/8 \rfloor$	
$s_i = \sum_{k=0}^{7} \begin{cases} 2^{7-k} & \exists (l_t, u_t) \in I_{group} : l_t \leq \lfloor l_r/8 \rfloor \cdot 8 + 8i + k \leq u_t, \text{for } r \leq t \leq s \\ 0 & \text{otherwise} \end{cases}$	

which are more suitable for efficient query processing, as no expensive duplicate elimination is required.

Definition 4.3 (non-overlapping interval groups).
Let $O_{group} = (id, \langle I_1, ..., I_m \rangle)$ be an interval group sequence. O_{group} is non-overlapping, iff $\forall i, j \in \{1, ..., m\} \forall \nu \in \mathbb{N}^d : (i \neq j) \wedge (\nu \in S(I_i)) \Rightarrow \nu \notin S(I_j)$.

4.2 Storing Interval Groups in an ORDBMS

The traditional approach for storing an object *id* in a database is to map its intervals to a set of tuples in an object-relational table *Interval* $(id, cnt, data)$ (cf. Table 4.3). The primary key is formed by the object identifier id and a unique number *cnt* for each interval. If we map our d-dimensional spatial object via space-filling curves onto one dimension, the result is an interval. A spatial index on the attribute data supports efficient query processing.

Table 4.2: Operators applied on the example given in Figure 4.2.

Operator	I_1	I_2	I_3
$H(I_x)$	[578,579]	[586,593]	[600,605]
$L(I_x)$	2	8	6
$D(I_x)$	1	$\frac{5}{8}$	$\frac{3}{6}$
$G(I_x)$	0	2	3
$B(I_x)$	'00110000'$_B$	'00110011 01000000'$_B$	'11000100'$_B$

For high resolution spatial data, this approach yields a high number of table and index entries and, consequently, leads to a critical query response behavior. A key idea of our approach is to store the non-overlapping interval group sequence $O_{group} = (id, \langle I_1, ..., I_m \rangle)$ in a set of m tuples in an object-relational table *IntervalGroup* $(id, cnt, data)$ (cf. Table 4.3(b)). Again, the primary key is formed by the object identifier id and a unique number cnt for each interval group. The set of voxels $\{\nu_1, ..., \nu_c\}$ of each interval group $I_{group} = \{\nu_1, ..., \nu_c\}$ is mapped to the complex attribute *data* which consists of aggregated information, i.e. the hull $H\ (I_{group})$ and the density $D\ (I_{group})$, and a byte sequence $B(I_{group})$ containing the complete information of the voxel set. In order to guarantee efficient query processing, we apply spatial index structures on $H(I_{group})$ and store $B(I_{group})$ in a compressed way within a binary large object (BLOB). $H(I_{group})$ is an interval group having a length $L(I_{group})$. If we map our d-dimensional spatial object via space-filling curves onto one dimension, $H(I_{group})$ is an interval group which can be managed by an index structure suitable for intervals, e.g. the RI-tree.

The two important advantages of this approach are as follows: First, the number of table and index entries can be controlled and reduced dramatically. Secondly, the access to the interval groups is efficiently supported by established relational access methods, e.g. the RI-tree. Note that the usage of interval groups is not restricted to the RI-tree. The approach can be extended to other relational spatial index structures such as the RQ-tree or the RR-tree by using using hyper-rectangular boxes. In any case, these access methods have to be created on $H(I_{group})$. There are two different problems

Table 4.3: Storage of interval groups applied on the example given in Figure 4.2.

(a) traditional approach

Interval

id	*cnt*	*data*
...	...	...
1	1	$[578, 579]$
1	2	$[586, 587]$
1	3	$[590, 591]$
1	4	$[593, 593]$
1	5	$[600, 601]$
1	6	$[605, 605]$
...	...	...

(b) new approach

IntervalGroup

id	*cnt*	*data*	
		$H(I_x)$	$B(I_x)$
...	...	...	...
1	1	$[578, 579]$	$'30'_H$
1	2	$[586, 593]$	$'33\ 40'_H$
1	3	$[600, 605]$	$'C4'_H$
...	...	...	...

related to the storage of interval group sequences: the compression problem and the grouping problem which is discussed in the following two chapters.

Chapter 5

Compression of Interval Groups

In this chapter, we first motivate the use of data compressors, by showing that $B(I_{group})$ contains patterns. Therefore, $B(I_{group})$ can efficiently be shrunken down by using data compressors. After discussing the properties which a suitable compression algorithm should fulfill, we introduce a new effective packer which exploits gaps and patterns included in the byte sequence $B(I_{group})$ of our interval group I_{group}.

5.1 Patterns

To describe a rectangle in a 2D vector space we only need 4 numerical values, e.g. we need two 2-dimensional points. In contrast to the vector representation, an enormous redundancy is contained in the corresponding voxel sequence of this object as depicted in Figure 5.1. As space filling curves enumerate the data space in a structured way, we can find such "structures" in the resulting voxel sequence representing simply shaped objects. We can pinpoint the same phenomenon not only for simply shaped parts but also for more complex real-world spatial parts. Assuming we cover the whole voxel sequence of an object id by one group, i.e. $O_{group} = (id, \langle I_{group} \rangle)$, and survey its byte representation $B(I_{group})$ in a hex-editor, we can notice that some byte sequences occur repeatedly. We now discuss how these patterns can be

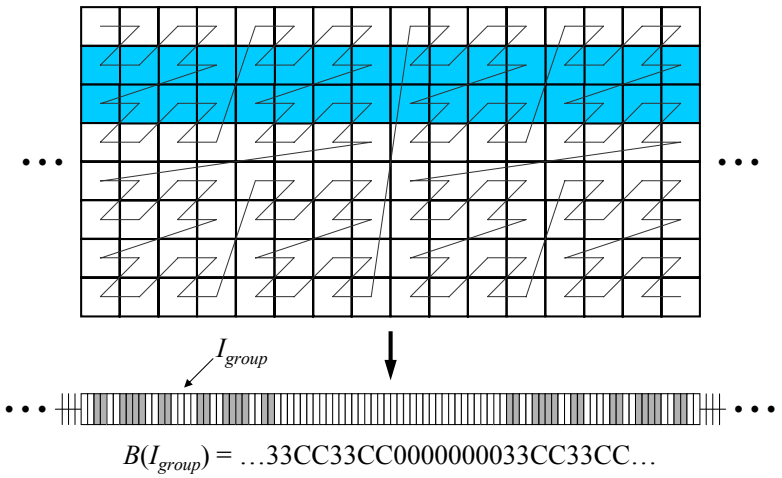


Figure 5.1: Pattern derivation by linearizing a voxelized object.

used for the efficient storage of interval groups in an ORDBMS.

5.2 Compression Rules

A voxel set belonging to an interval group $I_{group} = \{\nu_1, ..., \nu_n\}$ can be materialized and stored in a BLOB in many different ways. A good materialization should consider two "compression rules":

- As little as possible secondary storage should be occupied.
- As little as possible time should be needed for the (de)compression of the BLOB.

A good query response behavior is based on the fulfillment of both aspects. The first rule guarantees that the I/O cost $c_{I/O}^{load}$ are relatively small whereas the second rule is responsible for low CPU cost c_{CPU}^{decomp}. The overall time $c^{BLOB} = c_{I/O}^{load} + c_{CPU}^{decomp}$ for the evaluation of a BLOB is composed of both parts. Unfortunately, these two requirements are not necessarily in accordance with each other. If we compress the byte sequence $B(I_{group})$, we

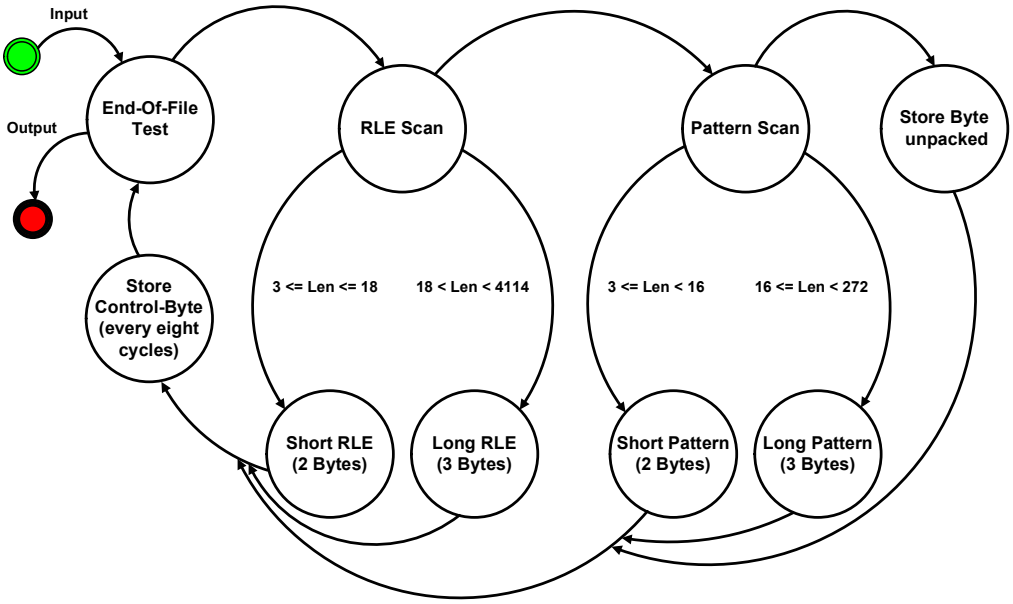


Figure 5.2: Flow diagram of QSDC compression algorithm.

can reduce the demand of secondary storage and consequently $c_{I/O}^{load}$. Unfortunately, c_{CPU}^{decomp} might rise because we first have to decompress the data before we can evaluate it. On the other hand, if we store $B(I_{group})$ without compressing it, $c_{I/O}^{load}$ might become very high whereas c_{CPU}^{decomp} might be low. Furthermore, a good update behavior also depends on the fulfillment of both rules.

As we show in our experiments, it is very important for a good query response- and update-behavior to find a well-balanced way between these two compression rules.

5.3 Spatial Compression Techniques

In this section, we look at a new specific compression technique, which is designed for storing the voxel set of an interval in a BLOB. According to our experiments, the new data compressor outperforms popular standard data compressors such as ZLIB or BZIP2.

5.3.1 Quick Spatial Data Compressor (QSDC)

The QSDC algorithm is especially designed for high resolution spatial data and includes specific features for the efficient handling of patterns and gaps. It is optimized for speed and does not perform time intensive computations as for instance Huffman compression. QSDC is a derivation of the LZ77 technique [LZ77]. However, it compresses data in only one pass and much faster than other Lempel-Ziv based compression schemes as for example XRAY [CW00]. QSDC operates on two main memory buffers. The compressor scans an input buffer for patterns and gaps (cf. Figure 5.2). QSDC replaces the patterns with a two- or three-byte compression code, the gaps with a one- or two-byte compression code. Then it writes the code to an output buffer. QSDC packs an entire BLOB in one piece, the input is not split into smaller chunks. At the beginning of each compression cycle QSDC checks if the end of the input data has been reached. If so, the compression stops. Otherwise another compression cycle is executed. Each pass through the cycle adds one item to the output buffer, either a compression code or a non-compressed character. To distinguish compressed from uncompressed data, a control bit is written at the end of each compression cycle. Unlike other data compressors, no checksum calculations are performed to detect data corruption because the underlying ORDBMS ensures data integrity. The decompressor reads compressed data from an input buffer, expands the codes to the original data, and writes the expanded data to the output buffer. When an extremely long run-length sequence occurs, the actual output buffer containing the decompressed data is returned to the calling process, and a new output buffer is allocated.

Lemma 5.1 (worst case compression ratio of QSDC).
The worst packed vs. unpacked ratio of QSDC for an input sequence of L bytes is $(\lceil L/8 \rceil + L)/L$.

Proof. *The QSDC packing algorithm makes sure that a compression code is always shorter than the original unpacked sequence. This means the only situation where the output can grow is when the compressor encounters unpackable bytes. An input byte which cannot be packed is directly passed through.*

Plus, an additional bit is consumed by the control byte. This sums up to L control bits, i.e. $\lceil L/8 \rceil$ *control bytes, which proves the lemma.* □

Chapter 6

Grouping into Interval Groups

Our grouping algorithm is based on the following two parameters:

- An average query distribution function QDF which is used to compute the access probability $P(I_{group}, QDF)$ of an interval group I_{group}.
- A look-up table LUT related to the evaluation of the BLOB.

6.1 Query Distribution

For many application areas, e.g. in the field of CAD and GIS, the average query distribution can be predicted very well. It is obvious that queries in rather dense areas, e.g. a cockpit in an airplane or a big city like New York, are much more frequently inquired than less dense areas. Furthermore, often small selective queries are posted.

As we do not know the actual query distribution, we have to assume an average query distribution function QDF which influences our grouping algorithm. We start with normalizing the coordinates of our one-dimensional query intervals to ensure that all data lies within the normalized data space $D := \{(l, u) \in [0, 1]^2 : l \leq u\}$ (cf. Figure 6.1 for one-dimensional query intervals Q_i).

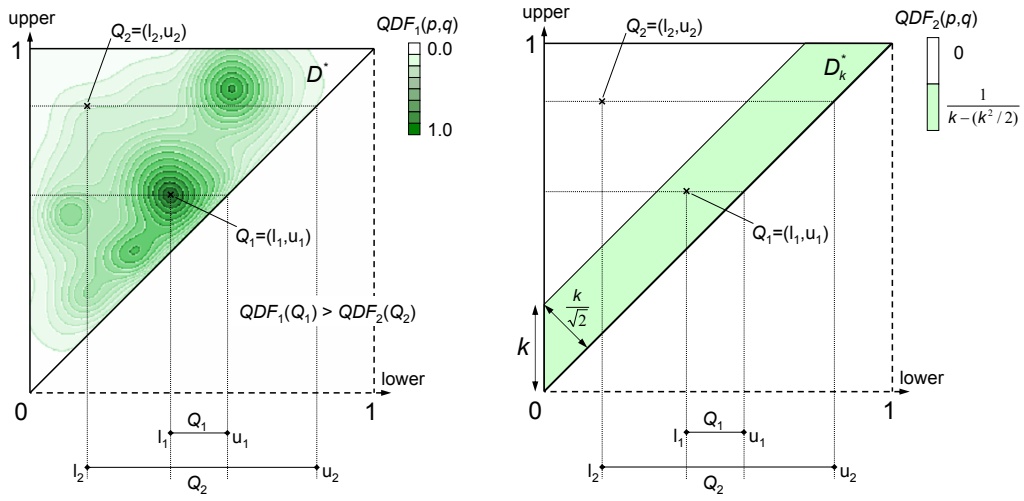


Figure 6.1: Query distribution functions $QDF_i(x, y)$.

In the following, we examine intervals and their point transformation into the upper triangle $D := \{(l, u) \in [0, 1]^2 : l \leq u\}$ of the two-dimensional hyper cuboid. An interval $Q = [x, y]$ therefore corresponds to the point (x, y) with $x \leq y$. Examples are visualized in Figure 6.1. To each of these points $Q = (x, y)$ we assign a numerical value $QDF(Q) \geq 0$. As the probability is equal to one that a query is somewhere located in the upper triangle D, the following equation has to hold:

$$\int_{l=0}^{l=1} \int_{u=l}^{u=1} QDF((l, u))\, du\, dl = 1.$$

Figure 6.1 shows two different query distribution functions. A potential query Q_2 is unlikely in Figure 6.1(a) and does not occur at all in Figure 6.1(b). On the other hand, query Q_1 is very likely in both cases. Let us note, that we used the simple query distribution function of Figure 6.1(b) throughout our experiments. In all considered application areas the common query objects only comprise a very small portion of the data space D. Therefore, we introduce the parameter $k \in [0, 1]$, which restricts the extension of the possible query objects. For the computation of the access probability we only consider normalized query objects whose extensions do not exceed k in each dimension.

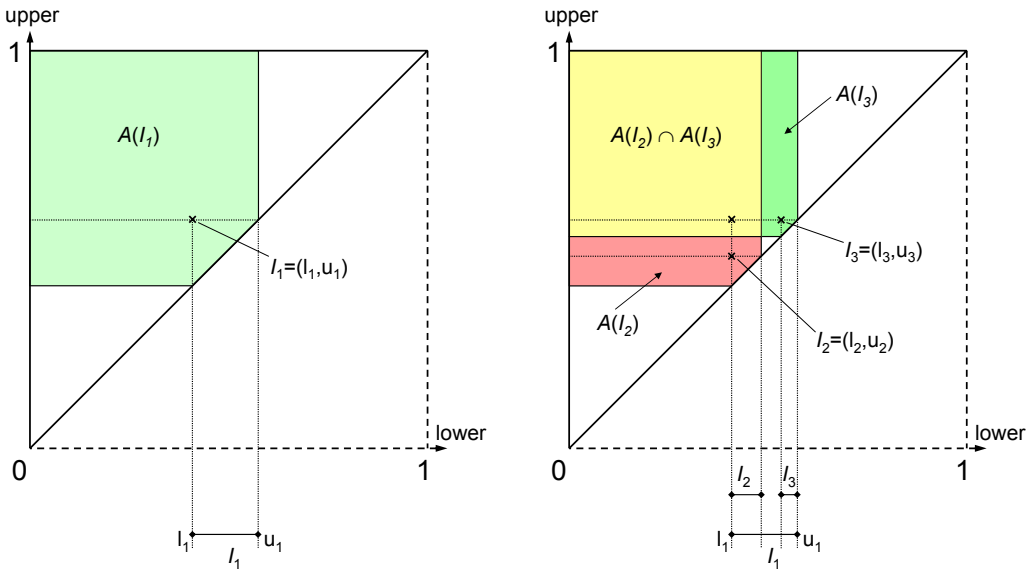


Figure 6.2: Computation of the access probability of interval groups.

6.2 Access Probability

The access probability $P(I_0, QDF)$ related to a grouped object I_0 denotes the probability that an arbitrary query object has an intersection with the two-dimensional hull $H(I_0)$. All possible query intervals that intersect I_0 are visualized by the shaded area $A(I_0)$ in Figure 6.2(a). The area displays all intervals whose lower bounds are smaller or equal to b and whose upper bounds are larger or equal to a. These query intervals are exactly the ones that have a non empty intersection with I_0. The probability that an interval $I_0 = [a_0, b_0]$ is intersected by an arbitrary query interval is

$$P(I_0, QDF) = \int_{l \in A(I_0)} \int_{u \in A(I_0)} QDF((l,u))\, du\, dl.$$

6.3 Evaluation Cost

Furthermore, the expected query cost depend on the cost related to the evaluation of the byte sequence stored in the BLOB of an intersected interval

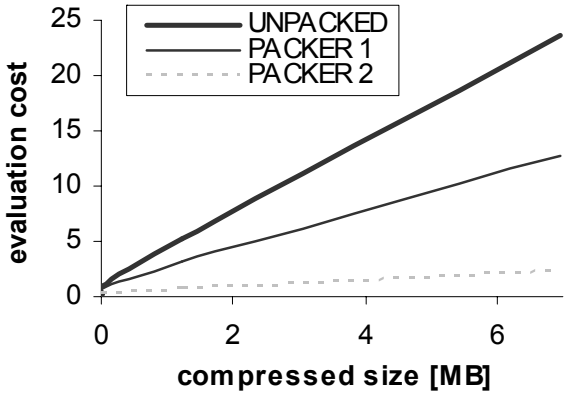


Figure 6.3: Evaluation cost $cost_eval$ for different data compressors.

group I_{group}. The evaluation of the BLOB content requires to load the BLOB from disk and decompress the data. Consequently, the evaluation cost depends on both the length $L(I_{group})$ of the uncompressed BLOB and the length $L_{comp}(I_{group}) \ll L(I_{group})$ of the compressed data. Additional, the evaluation cost $cost_{eval}$ depend on a constant $c_{I/O}^{load}$ related to the retrieval of the BLOB from secondary storage, a constant c_{CPU}^{decomp} related to the decompression of the BLOB, and a constant c_{CPU}^{test} related to the intersection test. The cost $c_{I/O}^{load}$ and c_{CPU}^{decomp} heavily depend on how we organize $B(I_{group})$ within our BLOB, i.e. on the used compression algorithm. A highly effective but not very time efficient packer, e.g. an arithmetic packer, would cause low loading cost but high decompression cost. In contrast, using no compression technique, leads to very high loading cost but no decompression cost. On the other hand, e.g. ZLIB is an effective and very efficient compression algorithm which yields a good trade-off between the loading and decompression cost. Finally, c_{CPU}^{test} solely depends on the used system. The overall evaluation cost are defined by the following formula:

$$cost_{eval}(I_{group}) = L_{comp}(I_{group}) \cdot c_{I/O}^{load} + L(I_{group}) \cdot (c_{CPU}^{decomp} + c_{CPU}^{test})$$

For each compression algorithm we provide statistics, i.e. a empirically derived look-up table LUT (cf. Figure 6.3), by means of which we can estimate the cost $c_{I/O}^{load}$ and c_{CPU}^{decomp} related to a possible evaluation of the BLOB. Roughly speaking, the evaluation cost depend on the length of our interval

```
ALGORITHM GroupInt(I_group, QDF, LUT)
BEGIN
  interval_pair := split_at_maximum_gap(I_group);
  I_left := interval_pair.left;
  I_right := interval_pair.right;
  cost_group := P(I_group, QDF) · cost_eval(I_group, LUT);
  cost_dec := P(I_left, QDF) · cost_eval(I_left, LUT) +
              P(I_right, QDF) · cost_eval(I_right, LUT);
  IF cost_group > cost_dec THEN
    GroupInt(I_left, P);
    GroupInt(I_right, P);
  ELSE
    report(I_group);
  END IF;
END
```

Figure 6.4: Decomposition Algorithm $GroupInt$.

group $L(I_{group})$ and on the used data compressor.

To sum up, the access cost related to an interval group I_{group} can be computed as follows

$$cost(I_{group}) = P(I_{group}, QDF) \cdot cost_{eval}(I_{group}, LUT)$$

6.4 Decomposition Algorithm

Orenstein [Ore89] introduced the size- and error bound decomposition approach. Our first grouping rule "the number of interval groups should be small" can be met by applying the size-bound approach, while applying the error-bound approach results in the second rule "the dead area of all interval groups should be small". For fulfilling both rules, we introduce the following top-down grouping algorithm for interval groups, called $GroupInt$ (cf. Figure 6.4). $GroupInt$ is a recursive algorithm which starts with an approximation $O_{group} = (id, \langle I_{group} \rangle)$, i.e. we approximate the object by one interval group. In each step of our algorithm, we look for the maximum gap g within the actual interval group. We carry out the split along this gap, if the aver-

age query cost caused by the decomposed intervals is smaller than the cost caused by our input interval I_{group}. The expected cost related to an interval group I_{group} can be computed as described in the foregoing paragraph. An interval group which is reported by the $GroupInt$ algorithm is stored in the database and no longer taken into account in the next recursion step. Data compressors which have a high compression rate and a fast decompression method, result in an early stop of the $GroupInt$ algorithm generating a small number of interval groups. Our experimental evaluations suggest that this grouping algorithm yields results which are very close to the optimal ones for different data compression techniques and data space resolutions.

Chapter 7

Query Processing

In this chapter, we discuss how we can efficiently carry out intersection queries on top of the SQL-engine. Our approach uses the RI-tree for efficiently detecting intersecting interval hulls. In contrast to the last section, we do not assume any arbitrary data distribution but use statistical information reflecting the distribution of the interval group hulls managed by the RI-tree. The algorithm for decomposing a query object is basically the same as the one presented in Figure 6.4. The main difference is that the assumed intersection probability P, is replaced by a more accurate selectivity estimation σ reflecting the actual data distribution. In [KPPS02] it was shown how we can effectively and efficiently estimate the selectivity of interval intersection queries based on the RI-tree. Both index structure and the corresponding cost-model can be integrated into modern object relational database systems by using their extensible indexing and optimization frameworks (cf. Figure 7.1). By exploiting the statistical information provided by the cost model, we can find an optimum decomposition for the query object. The traditional error- and size-bound decomposition approaches [Ore89] decompose a large query object into smaller query objects optimizing the trade-off between accuracy and redundancy. In contrast, the idea of taking the actual data distribution into account in order to decompose the query object, leads to a new selectivity-bound decomposition approach, which tries to minimize the overall number of logical reads.

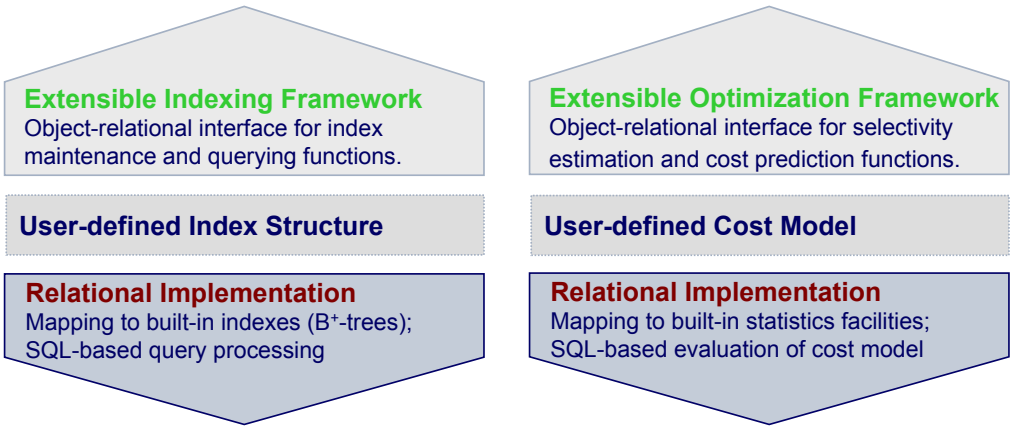


Figure 7.1: Analogous architectures for the object-relational embedding.

7.1 Decomposition of the Query Object

In this section, we shortly sketch the decompositioning of the query object into suitable interval group sequences. Thereby two different cases have to be distinguished. First, the query object is already stored in a decomposed way in the database. Second, it has to be decomposed in real-time from scratch.

7.1.1 Query object is a database object

In this case, the query object is already stored in a decomposed way within the database according to our $GroupInt$ algorithm which assumes a potential query distribution QDF (cf. Figure 6.1). Usually areas which are often inquired also contain a lot more objects than seldomly inquired areas. For instance, New York contains a lot of geographical objects and is much more often inquired than Alaska. As we assume that the characteristic QDF coincides with the data distribution σ of the actual databases objects, we use the already decomposed objects as starting point for a further statistic-driven generation of the interval groups instead of starting from scratch. In order to carry-out a fine-tuning of our grouped query-sequence we test whether two already decomposed interval groups should be merged to one interval group. This is beneficial if $\sigma < QDF$ holds for the query region. On

the other hand, if $\sigma > QDF$ holds, we further decompose the query interval group according to the algorithm presented in Figure 6.4. That way, we do not assume a potential query distribution QDF, but replace QDF by the actual selectivity estimation σ w.r.t. the corresponding interval group. We carry out the selectivity estimation for I_{group} and for the two potentially new interval groups I_{left} and I_{right} which result from a split at the maximum gap of I_{group}. Based on the computed cost, we decide whether we actually carry out the split.

7.1.2 Query object is no database object

Second, if the query object has to be decomposed from scratch, we carry out a decompositioning starting with one interval group conservatively approximating the query object. Similar to the approach of the last paragraph, we decide based on an accurate selectivity estimation whether to further decompose the query object. This approach is quite feasible if the query object is already available as voxelized object. If not, which is the case for some common query objects used in GIS or CAD, they are often very simple and can be described by a few parameters, e.g. a rectilinear box or cube can be described by two points. Again, we approximate the query object by one interval group and apply the already sketched decompositioning approach for query objects. The few parameters which are necessary to describe the query object can be stored in the $BLOB$ of each interval group instead of the exact interval sequence. These few values contain the whole information in the most compressed way. In the blobintersection routines of Figure 7.2, $B(I_{group})$ is created on demand from this simple and compact geometric information. As the geometric information is already in the most compressed form we do not need a special data compressor, but only a decompression algorithm specific to the geometry of the query object.

In the following, we show how we can carry out an interval intersection query on top of an ORDBMS.

7.2 Intersection Query

In this section, we discuss the query processing of a boolean and a ranked intersection predicate on top of the SQL-engine. The presented approaches can easily be embedded by means of extensible indexing interfaces (cf. Figure 7.1) into modern ORDBMS. Most ORDBMSs, including Oracle [Cor99b, SMS+00], IBM DB2 [Cor99a, CCF+99] or Informix IDS/UDO [IS98, BSSJ99], provide these extensibility interfaces in order to enable database developers to seamlessly integrate custom object types and predicates within the declarative DDL and DML. As we represent spatial objects by interval group sequences, we first clarify when two of these sequences intersect.

Definition 7.1 (Object Intersection).
Let $W = \{(l,u) \in \mathbb{N}^2, l \leq u\}$ *be the domain of intervals and let* $i_1 = (l_1, u_1)$ *and* $i_2 = (l_2, u_2)$ *be two intervals. Further, let* $I^1 = \langle i_1^1, ..., i_n^1 \rangle$ *and* $I^2 = \langle i_1^2, ..., i_n^2 \rangle$ *be two interval groups, and let* $O^1 = (id^1, \langle I_1^1, I_2^1, ..., I_{m_1}^1 \rangle)$ *and* $O^2 = (id^2, \langle I_1^2, I_2^2, ..., I_{m_1}^2 \rangle)$ *be two interval group sequences. Then, the notions intersect, Xintersect and interlace are defined in the following way:*

1a. *Two intervals* i_1 *and* i_2 *intersect if* $l_1 \leq u2$ *and* $l_2 \leq u_1$.

1b. $Xintersect(i_1, i_2) = max\{0, min\{u_1, u_2\} - max\{l_1, l_2\} + 1\}$.

2a. *Two interval groups* I^1 *and* I^2 *intersect if for any* $i \in \{1, ..., n_1\}$, $j \in \{1, ..., n_2\}$ *the intervals* i_i^1 *and* i_j^2 *intersect.*

2b. $Xintersect(I^1, I^2) = \sum_{i=1...n_1, j=1...n_2} Xintersect(i_i, i_j)$.

2c. *Two interval groups* I^1 *and* I^2 *interlace, if their hulls* $H(I^1)$ *and* $H(I^2)$ *intersect.*

3a. *Two objects* O^1 *and* O^2 *intersect, if for any* $i \in \{1, ..., m_1\}$, $j \in \{1, ..., m_2\}$ *the interval groups* I_i^1 *and* I_j^2 *intersect.*

3b. $Xintersect(O^1, O^2) = \sum_{i=1...m_1, j=1...m_2} Xintersect(I_i^1, I_j^2)$.

3c. *Two objects* O^1 *and* O^2 *interlace, if for any* $i \in \{1, ..., m_1\}, j \in \{1, ..., m_2\}$ *two interval groups* I_i^1 *and* I_j^2 *interlace.*

In other words, the *intersect*-predicate returns a boolean result whether two intervals or interval groups have an empty intersection or not. The *Xintersect*-predicate measures the intersection magnitude of two intervals or interval groups.

7.3 The intersect SQL Statements

In [KPPS03a], many index structures for interval intersection queries were surveyed, which can be integrated into the extensible indexing framework of modern ORDBMSs. These index structures also support the evaluation of the interlace predicate on interval groups. As we defined object interval group sequences as a conservative approximation of normal object interval sequences, we can use the hulls of the interval groups in a first conservative filter step. Thereby, we can take advantage of the same access methods as used for the detection of intersecting interval pairs. As shown in Section 4.2, the interval group sequences can be mapped to an object-relational schema *IntervalGroups*. Following this approach, we can also clearly express the intersect predicates on top of the SQL engine (cf. Figure 7.2).

In the case of the intersect-predicate (cf. Figure 7.2(a)), the nesting function *table* groups references of interlacing grouped query and database interval pairs together. The NF2-operator *table* was realized by a user-defined aggregate function as provided in the SQL:1999 standard. In order to find out which database objects are intersected by a specific query object, the interlacing grouped intervals have to be tested for intersection. This test is carried out by a stored procedure *blobintersection*. If one intersecting grouped database and query interval pair is found, no grouped interlacing interval pairs belonging to the same database object have to be examined. This skipping principle is realized by means of the exists-clause within the SQL-statement. In the case of the *Xintersect*-predicate (cf. Figure 7.2(a)), the intersection length has to be determined for each interlacing interval pair. No BLOB tests can be skipped. The results are summed up in the user-defined aggregate function *Xblobintersection*. In both *blobintersection*

```
SELECT candidates.id FROM (
  SELECT db.id AS id, table (pair(db.rowid, q.rowid)) AS ctable
  FROM IntervalGroups db, :QueryIntervalGroups q
  WHERE intersects (hull(db.data), hull(q.data))
  GROUP BY db.id
) candidates
WHERE EXISTS (
  SELECT 1
  FROM IntervalGroups db, :QueryIntervalGroups q, candidates.ctable ctable
  WHERE db.rowid = ctable.dbrowid AND q.rowid = ctable.qrowid
    AND blobintersection (db.data, q.data)
);
```

(a) intersect-predicate

```
SELECT db.id, Xblobintersection(db.data, q.data)
FROM IntervalGroups db, :QueryIntervalGroups q
WHERE intersects (hull(db.data), hull(q.data))
GROUP BY db.id;
```

(b) Xintersect-predicate

Figure 7.2: SQL statements for spatial object intersection, based on interval group sequences.

routines, we first decompress the data and then test the two byte sequences in the interlacing area for intersection. As already mentioned in Section 5.2 it is important that the compressed BLOB size is small in order to reduce the I/O cost. Obviously, the small I/O cost should not be at the expense of the CPU cost. Therefore, it is important that we have a fast decompressing algorithm in order to evaluate the BLOBs quickly.

7.4 Optimizations

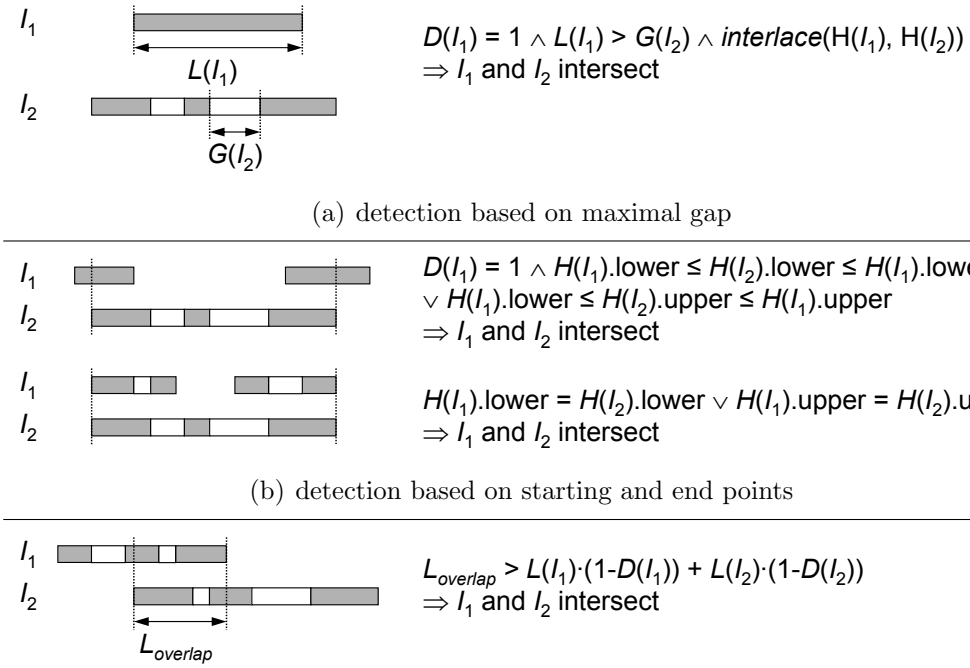
For the *intersect* predicate, it suffices to find a single intersecting query and database group pair in order to issue the database id. Obviously, it is desirable to detect such intersecting pairs as early as possible in order to avoid unnecessary *blobintersection* tests. In this section, we present two optimiza-

tions striking for this goal. First, we introduce a fast second filter step which tries to determine intersecting pairs without examining the BLOBs. This test is entirely based on aggregated information of the interval groups. Second, we introduce a probability model which leads to an ordering for the candidate pairs such that the most promising *blobintersection* tests are carried out first. In order to put these optimizations into practice, we pass $D(Igroup)$ and $H(I_{group})$ as additional parameters to the user-defined aggregate function table. Thus, the following two optimizations can easily be integrated into this user-defined aggregate function. If the fast second filter step determines an intersecting interval group, all other candidate pairs are deleted so that the resulting table of candidate pairs, called *ctable*, consists only of one intersecting pair (cf. Section 7.4.1). Nevertheless, there might be database objects where this second filter step does not determine an intersection for any of the corresponding candidate pairs. In this case, we sort the candidate pairs at the end of our user-defined aggregation function table such that the most promising *blobintersection* tests are carried out first (cf. Section 7.4.2).

7.4.1 Fast Intersection Test for Interval Groups

We speak of 'overlapping' intervals, if the hulls of the intervals intersect. We now discuss how interval groups have to look like so that we can decide whether two overlapping intervals actually intersect each other or not without accessing their BLOBs. If any of the following five conditions holds, then two interval groups intersect:

- If two intervals overlap, they necessarily intersect as well.
- If an interval is longer than the maximum gap between two intervals contained in I_{group}, then the two intervals intersect (cf. Figure 7.3(a)).
- If an interval overlaps the start or end of an interval group, then the intervals intersect. This is due to the fact that any interval group ends and starts with a "1" voxel (cf. Figure 7.3(b)).
- If interval groups start or end at the same point, then the intervals



(a) detection based on maximal gap

(b) detection based on starting and end points

(c) detection based on density

Figure 7.3: Intersection optimizations for interval groups.

intersect. This is due to the fact that any interval group ends and starts with a "1" voxel (cf. Figure 7.3(b)).

- If the sum of the number of the white voxels of two grouped overlapping intervals is smaller than the length of the overlapping area, then the two intervals necessarily intersect. (cf. Figure 7.3(c)).

Note that the above mentioned optimizations are only suitable for the *intersect*-predicate on interval groups.

7.4.2 Ranking

As shown above, we can sometimes pinpoint, based on relatively little information, whether two interval group pairs intersect. Nevertheless, there might be cases where we cannot do this for any of the database and query candidate pair. But if the hulls of two interval groups intersect, it is still helpful if

we can predict how likely an object intersection according to Definition 7.1 might be in order to rank the pairs properly in the set of all candidate pairs belonging to the same database object.

The following probability model is rather simple as it is equal to the coin-toss experiment, i.e. it is a Bernoulli experiment. It assumes that the intervals and gaps covered by an interval group are equally distributed.

Lemma 7.1 (Probability for an Intersection).
Let I_{group} and I'_{group} be two interval groups with densities $d = D(I'_{group})$ and $d' = D'(I'_{group})$, hulls $H(I_{group}) = [h_l, h_u]$ and $H(I'_{group}) = [h'_l, h'_u]$ and an intersection length of $L = min\{h_u, h'_u\} - max\{h_l, h'_l\} + 1$. Then the probability $P(I_{group}, I'_{group})$ that the two grouped intervals I_{group} and I'_{group} intersect is equal to:

$$P(I_{group}, I'_{group}) = 1 - (1 - D \cdot D')^L$$

Proof. *Let x be one of the points in the interlacing area. Obviously, the probability that this point is covered by an interval contained in I_{group} is equal to the density D. Subsequently, the probability that two intervals I_{group} and I'_{group} intersect at the point x is $P_x = D \cdot D'$. The probability, that either x or another point $y \neq x$ is covered by intervals from I_{group} and I'_{group} is $P_{x,y} = D \cdot D' + (1 - D \cdot D') \cdot D \cdot D'$. As we assume that the interval bounds are mapped to discrete integer values, the probability that I_{group} and I'_{group} share at least one point can be computed as follows*

$$\begin{aligned} P(I_{group}, I'_{group}) &= \sum_{i=0}^{L-1} D \cdot D' \cdot (1 - D \cdot D')^i \\ &= D \cdot D' \frac{1 - (1 - D \cdot D')^L}{1 - (1 - D \cdot D')} \\ &= 1 - (1 - D \cdot D')^L \end{aligned}$$

□

In [KKPR05], the idea of ranking the intervals has been developed further to a distributed intersection join for interval sequences of high-cardinality which tries to minimize the transmission cost. The approach is based on the

above probability model for interval intersections which is used on the server as well as on the various clients. On the client sites, intervals are grouped together based on this probability model. The locally created approximations are sent to the server. The server ranks all intersecting approximations according to our probability model. As not all approximations have to be refined in order to decide whether two objects intersect, we fetch the exact information of the most promising approximations first. This strategy helps to cut down the transmission cost considerably.

Chapter 8

Experimental Evaluation

In this section, we evaluate the performance of our generic approach for accelerating relational spatial index structures, with a special emphasis on the various data compression techniques introduced in Chapter 2.1. We evaluate different grouping algorithms GRP in combination with various data compression techniques DC. We used the following data compressors DC:

NOOPT:	$B(I_{group})$ is unpacked.
BZIP2:	$B(I_{group})$ is packed according to the BZIP2 approach [BW94].
ZLIB:	$B(I_{group})$ is packed according to the ZLIB approach [Deu96].
OPTRLE:	$B(I_{group})$ is packed according to the approach in [KPPS03b].
QSDC:	$B(I_{group})$ is packed according to the QSDC approach.

Furthermore, we grouped voxels into interval groups depending on two grouping algorithms, called $MAXGAP$ and $GroupInt$.

MaxGap. This grouping algorithm tries to minimize the number of interval groups while not allowing that a maximum gap $G(I_{group})$ of any interval group I_{group} exceeds a given $MAXGAP$ parameter. By varying this $MAXGAP$ parameter, we can find the optimum trade-off between the first two opposing grouping rules of Chapter 4, namely a small number of interval groups and a small amount of dead space included in each of these interval groups.

Table 8.1: Summary of the Spatial Test Datasets.

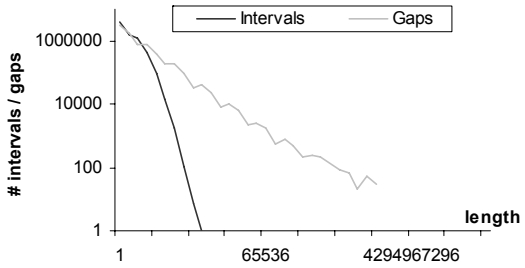
Dataset	dimensions	data space size	# objects	voxels
SEQUOIA	2	2^{34}	3500	$50 \cdot 10^6$
CAR	3	2^{33}	200	$14 \cdot 10^6$
PLANE	3	2^{42}	10,000	$18 \cdot 10^6$

GroupInt. We grouped the interval groups according to our cost-based grouping algorithm *GroupInt* (cf. Chapter 6), where we used the query distribution function from Figure 6.1(b) with $k = 1/100,000$ and a look-up table for each packer. The look-up table was created by experimentally determining the average cost for evaluating an interval group I_{group}, dependent on the length of its byte sequence. Note, that the grouping based on $MaxGap(DC)$ ignores the DC parameter (i.e. does not depend on DC), whereas $GroupInt(DC)$ takes the actual data compressor DC into account for performing the grouping.

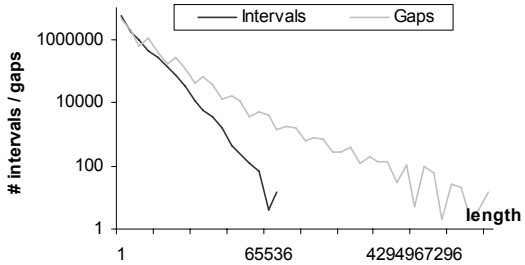
In order to support the first filter step of GRP(DC), we can take any arbitrary access method for spatial data, e.g. the RI-tree. We have implemented the RI-tree [KPS01, KPS00] on top of the Oracle9i Server using PL/SQL for most of the computational main memory based programming. The evaluation of the *blobintersection* routines was delegated to a DLL written in C. All experiments were performed on a Pentium III/700 machine with IDE hard drives. The database block cache was set to 500 disk blocks with a block size of 8 KB and was used exclusively by one active session.

Test Datasets. The tests are based on three test datasets *CAR*, *PLANE* and *SEQUOIA*. Table 8.1 contains a short summary of the characteristics of all datasets.

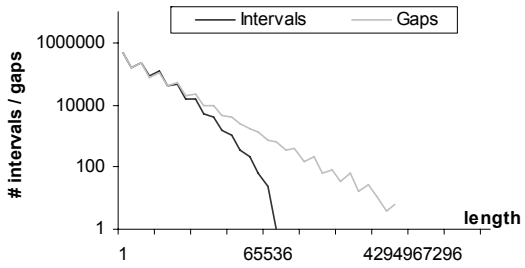
In all cases, the Z-curve was used as a space filling curve to enumerate the voxels. Figure 8.1 depicts the interval and gap histograms for our test data sets. All three datasets consist of many short intervals and short gaps, and only a few longer ones. We tested all compression techniques for both the $intersect$ and the $Xintersect$ predicate.



(a) *CAR* dataset



(b) *PLANE* dataset

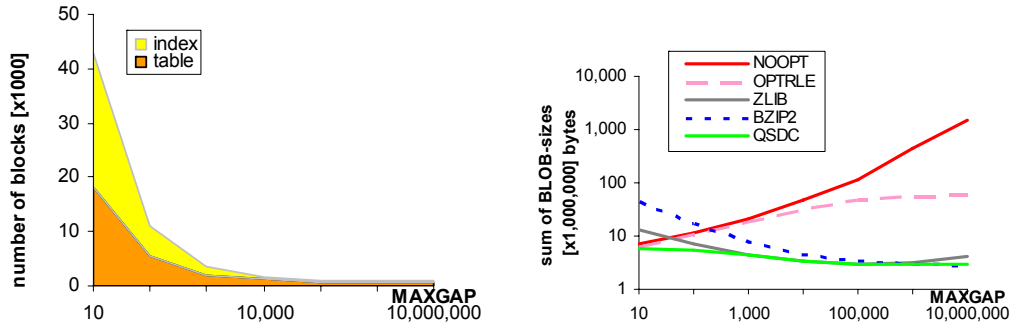


(c) *SEQUOIA* dataset

Figure 8.1: Histograms for intervals and gaps.

8.1 Storage Requirements

First we look at the storage requirements of the RI-tree on the PLANE dataset. In Figure 8.2(a), the storage requirements for the index, i.e. the two B+-trees underlying the RI-tree, as well as for the complete *IntervalGroups* table are depicted for the $MaxGap(QSDC)$ approach. In the case of small $MAXGAP$ parameters, the number of disk blocks used by the index dominates the number of disk blocks for the *IntervalGroups* table. With increasing $MAXGAP$ parameters, the number of disk blocks used by the index



(a) Index & BLOB for MaxGap(QSDC) (b) BLOB for MaxGap(DC)

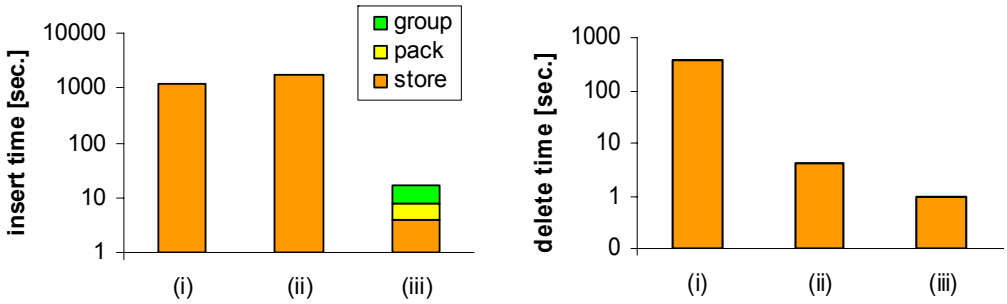
Figure 8.2: Storage requirements for the RI-tree (PLANE).

dramatically decreases hand in hand with the number of interval group sequences, and at high parameter values they yield no significant contributions any more to the overall sum of used disk blocks.

Figure 8.2(b) shows the different storage requirements for the BLOB with respect to the different data compression techniques. Due to an enormous overhead, the ZLIB and BZIP2 approaches occupy a lot of secondary storage space for small $MAXGAP$ values. On the other hand, for high $MAXGAP$ values they yield very high compression rates. For the PLANE dataset the BZIP2 approach yields a compression rate of more than 1:500 and is at least 20 times more efficient than the approach used in [KPPS03b]. The QSDC approach yields good results over the full range of the $MAXGAP$ parameter. For high $MAXGAP$ values, the number of disk blocks used for the BLOBs corresponds to the number of disk blocks used overall. For these high $MAXGAP$ parameters, the $MaxGap(QSDC)$, $MaxGap(ZLIB)$ and $MaxGap(BZIP2)$ approach lead to a much better storage utilization than the $MaxGap(NOOPT)$ and the $MaxGap(OPTRLE)$ approach.

8.2 Update Operations

In this section we investigate the time needed for updating complex spatial objects in the database. For most of the investigated application ranges, it



(a) Insert operation (b) Delete operation

Figure 8.3: Update operation cost for the RI-tree (CAR) (i) numerous intervals, (ii) one interval group, (iii) interval groups generated by $GroupInt(QSDC)$.

is enough to confine ourselves to insert and delete operations, as updates are usually carried out by deleting the object out of the database and inserting the altered object again. Figure 8.3(a) shows that inserting all objects into the database takes very long if we store the numerous intervals in the RI-tree (i) or if we store one value approximations of the unpacked object in the RI-tree (ii). On the other hand, using our $GroupInt(QSDC)$ approach (iii) accelerates the insert operations by almost two orders of magnitude. The time spent for grouping and packing pays off, if we take into consideration that we save a lot of time for storing grouped and packed objects in the database.

Obviously, the delete operations are also carried out much faster for our $GroupInt(QSDC)$ approach as we have to delete much less disk blocks (cf. Figure 8.3(b)).

8.3 Query Processing

In this section, we want to turn our attention to the query processing by examining different kinds of collision queries. The figures presented in this paragraph depict the average result obtained from collision queries where we have taken every part from the CAR dataset and the 100 largest parts from

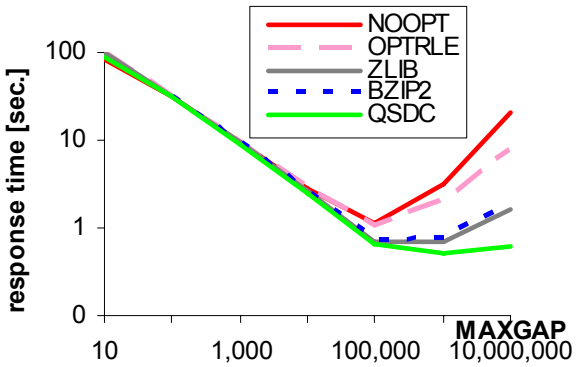


Figure 8.4: $MaxGap(DC)$ evaluated for boolean intersection queries on the RI-tree (PLANE).

the PLANE dataset as query objects.

8.3.1 MaxGap

In Figure 8.4 it is shown in which way the overall response time for boolean intersection queries based on the RI-tree depends on the $MAXGAP$ parameter. If we use small $MAXGAP$ parameters, we need a lot of time for the first filter step whereas the *blobintersection* test is relatively cheap. Therefore, the different $MaxGap(DC)$ approaches do not differ very much for small $MAXGAP$ values. For high MAXGAP values we can see that the $MaxGap(QSDC)$ approach performs best with respect to the overall runtime. The $MaxGap(QSDC)$ approach is rather insensitive against too large $MAXGAP$ parameters. Even for values where the first filter step is almost irrelevant, e.g. $MAXGAP = 107$, the $MaxGap(QSDC)$ approach still performs well. This is due to the fact that for large $MAXGAP$ values the $MaxGap(QSDC)$ approach needs much less physical reads, about 1% of the $MaxGap(NOOPT)$ approach. As a consequence, the query response time of the $MaxGap(QSDC)$ approach is approximately 1/35 of the query response time of the $MaxGap(NOOPT)$ approach.

In Figure 8.6 it is shown in what way the different data space resolutions influence the query response time. Generally, the higher the resolution, the

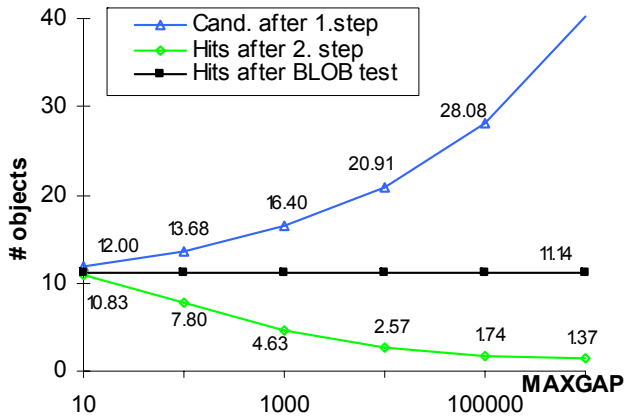
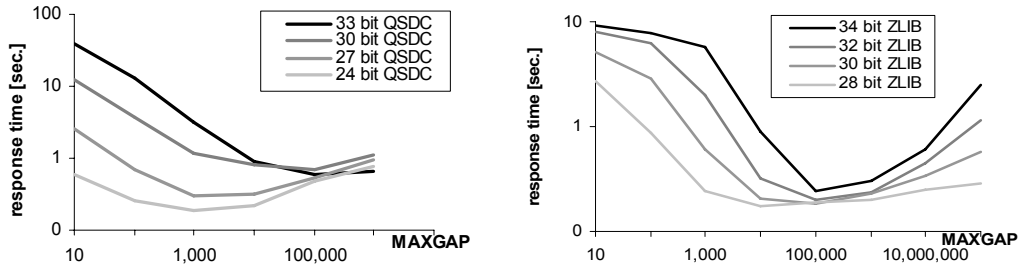


Figure 8.5: Filter quality for boolean intersection queries on the RI-tree (CAR).

slower is the query processing. Our $MaxGap(QSDC)$ is especially suitable for high resolutions, but also accelerates medium or low resolution spatial data. To sum up, the $MaxGap(QSDC)$ approach improves the response time of collision queries for varying index structures and resolutions by up to two orders of magnitude. In Figure 8.5, it is illustrated that at small $MAXGAP$ values the number of the different object IDs resulting from the first filter step is only marginally higher than the number of different IDs in the final result set. Likewise, the number of detected hits in the second filter step is only marginally smaller. With increasing $MAXGAP$ values the two curves diverge. To sum up, the optimizations presented in Section 7.4 are especially useful for small $MAXGAP$ parameters.

8.3.2 GroupInt

Figure 8.4 shows that for packed data, the optimum $MAXGAP$ value is higher than for unpacked data. Furthermore, Figure 8.6 shows that for increasing resolutions the optimum $MAXGAP$ also increases. We now experimentally show that the *GroupInt* algorithm produces object decompositions which yield almost optimum query response times for varying index structures, compression techniques and data space resolutions. Please note that the grouping of the 3D datasets has been performed without the help of statis-



(a) CAR dataset (b) SEQUOIA dataset

Figure 8.6: $MaxGap(QSDC)$ evaluated for boolean intersection queries for the RI-tree using different resolutions.

tics modeling the data distribution (cf. Section 7.1). For our 3D datasets, the initial grouping of the objects turned out to be quite stable and the enhanced grouping with the help of statistics showed only little differences.

Table 8.2 depicts the overall query response time for boolean and ranking intersection queries for the RI-tree based on the *GroupInt* algorithm.

We can see that for boolean intersection queries this grouping delivers results quite close to the minimum response times depicted in Figure 8.4. Furthermore, we notice that the $GroupCon(QSDC)$ approach outperforms the RI-tree [KPS01] by a factor of 180 for boolean intersection queries on the PLANE dataset. For ranking intersection queries the RI-tree [KPS00] is not applicable due to the enormous amount of generated join partners. On the other hand, the $GroupInt(QSDC)$ approach yields interactive response times even for such queries. The *GroupInt* algorithm adapts to the opti-

Table 8.2: $GroupInt(DC)$ evaluated for boolean* and ranking** intersection queries for the RI-tree (PLANE).

	NOOPT	BZIP2	QSDC	RI-tree [KPS01]* [KPS00]**
# interval groups	24,453	16,063	15,468	9,289,569
runtime* [sec]	1.35	0.71	0.55	135.01
runtime** [sec]	2.42	1.23	0.92	∞ (not applicable)

Table 8.3: $GroupInt(ZLIB)$ evaluated for boolean intersection queries for the RI-tree with different resolutions (SEQUOIA).

	34 bit	32 bit	30 bit	28 bit
runtime [sec]	0.29	0.24	0.21	0.19

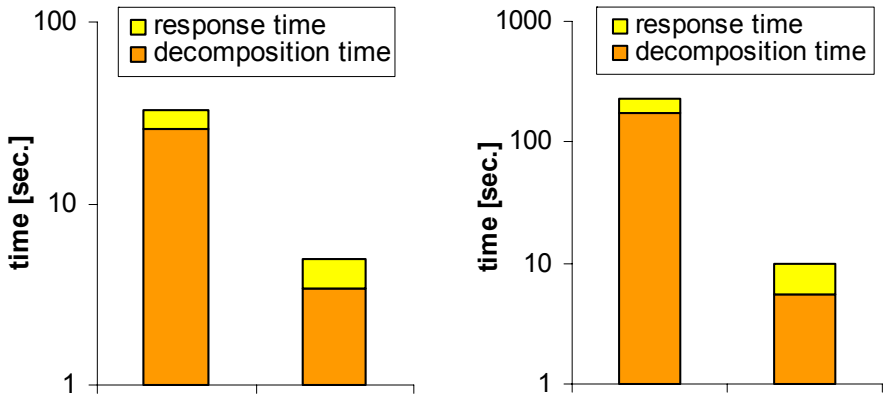
mum $MAXGAP$ parameter for varying compression techniques, by allowing greater gaps for packed data, i.e the number of generated interval group objects is smaller in the case of packed data.

In Table 8.3 it is shown that the query response times resulting from the $GroupInt$ algorithm for varying resolutions, are almost identical to the ones resulting from a grouping based on an optimum $MAXGAP$ parameter (cf. Figure 8.6)

To sum up, the $GroupInt$ algorithm produces object decompositions which yield almost optimum query response times for varying compression techniques and data space resolutions.

8.3.3 Window Queries

In a last experiment, we carried out different window queries. Figure 8.7 depicts the average runtime for window queries, where we moved each window to 10 different locations. As shown in Figure 8.7(a) and 8.7(b), our statistic-based decomposition approach of the query object can improve the query response behavior by more than one order of magnitude, compared to the granularity-bound decompositioning approach where we decompose the query objects into normal intervals. This speed-up is mainly due to the reduced decomposition time resulting from the fact that we do not decompose the interval groups completely into normal intervals, but take the actual data distribution into account to guide the decompositioning process. If our selectivity estimation indicates low selectivity for the actual interval group, we do not further decompose the interval group but use it directly as query interval, where the actual window coordinates are stored in the corresponding BLOB. The experiments show that the query response time does not suffer



(a) Window size equals 0.00001% of the data space yielding approx. 0.01% selectivity

(b) Window size equals 0.01% of the data space yielding approx. 1.0% selectivity

Figure 8.7: Window queries (SEQUOIA).

from the fact that we did not decompose the windows with the maximum possible accuracy. The time we need for the additional refinement step to filter out false hits is compensated by the much smaller number of query intervals resulting from a coarser decomposition of the query window. To sum up, our statistic-based decomposition approach is especially useful for commonly used window queries.

Part III

Analysis of Temporal Data

Chapter 9

Introduction

In this part, we introduce the new concept of threshold-based similarity search for time series databases. In Part II we used interval group sequences as basic representation of complex spatial objects. Now, we propose to use interval sequences as basic representation of complex shaped time series objects. Certainly, one advantage for this type of representation is, similar to that of representing complex spatial objects, that interval sequences are easier to handle than the original object representations. However, in case of time series there is another very important advantage. Based on this new type of time series representation, we can define a novel but very promising similarity measure for time series. This measure does not only provide new prospects in data mining in time series databases but also allows to develop efficient methods for searching in very large databases comprising large and complex time series objects.

In particular, we present efficient and effective similarity search algorithms for time series. As mentioned above, we use sequences of intervals in order to approximate our objects. In opposite to Part II, we consider intervals with a temporal instead of a spacial dimension. A particular interval sequence representation of a time series object is associated with a given amplitude threshold.

9.1 Preliminaries

Time series are sequences, discrete or continuous, of quantitative data assigned to specific moments in time. Formally, we define a time series as follows:

Definition 9.1 (Time Series).
A 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 $x_i \in \mathbb{R}$ denotes the data corresponding to time t_i. Naturally, we assume that the sequence is sorted w.r.t. the time slots, i.e. $\forall i \in 1, .., N-1 : t_i < t_{i+1}$.

Time series often represent continuously changing attributes and their values are sampled at discrete time slots. In case of missing values, i.e. nonexistent values between two measurements, we assume that the gaps are filled by means of interpolation. There exists a large range of appropriate solutions for time series interpolation. Because linear interpolation is the most prevalent interpolation method for time series, we assume that the time series curves are supplemented by linear interpolation, if required. Throughout the rest of this thesis, $x(t) \in \mathbb{R}$ denotes the (interpolated) time series value of time series X at time $t \in T$.

9.2 Threshold Based Similarity Measure

Time series are usually very large, containing several hundreds or even thousands of values per sequence. Consequently, the comparison of two time series can be very expensive, particularly when considering the entire sequence of values of the compared objects. There are a lot of data mining applications where the mining process does not need the entire course of the time series. Vague qualitative information about the shape like "above" or "below" a certain threshold may often be sufficient and even desired in some applications.

In this thesis, we introduce a novel type of similarity measure for time series called *threshold distance* or *threshold similarity*. The corresponding similarity query on time series databases is called *threshold query* (TQ). Threshold queries enable the analysis of time series tightly focused on specific amplitude values, which are important and significant for the analysis goal.

In many application areas it could be beneficial if the analysis of time series data would be concentrated on certain amplitudes (or amplitude spectra). A sample application from medical analysis is visualized in Figure 9.1 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 after the systole which indicates a low risk. For the examination of times series w.r.t. 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 displayed below the ECG-curves. Each time interval sequence belongs to one time series. They correspond to the time frames within a time series that exceed the specified threshold τ. We can observe that the time interval sequences derived from $T2$ and $T3$ differ only marginally. In contrast, the time series $T1$ shows quite different characteristics caused by the ECG-aberration which indicates the heart disease.

The applicability of threshold based time series analysis can be also demonstrated by the example depicted in Figure 9.2. There are four time series from the real time series dataset *Trace* depicted, each representing a class of several time series which are hidden for clarity reasons. A detailed description of this dataset is given in Section 2.3. Basically, the four classes differ significantly at two certain positions. The time series may vary slightly in time. For a good classification, it seems promising to concentrate the similarity search on the significant amplitude instead of taking the entire time series into account. In fact, we observed in our experiments that we achieve the best classification accuracy for this dataset when considering only the

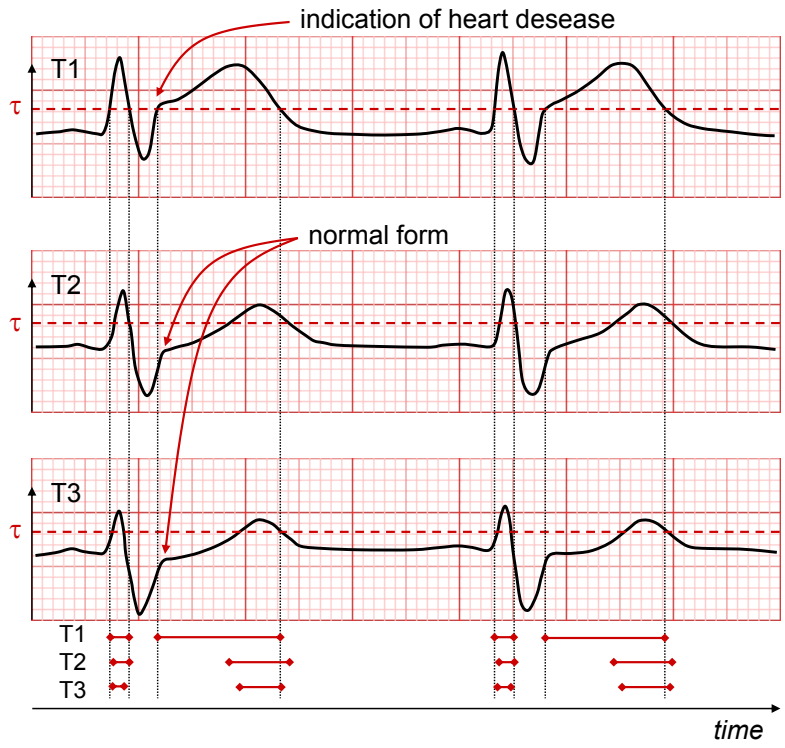


Figure 9.1: Threshold-based detection of risk patients for heart diseases.

significant amplitude value. The achieved classification accuracy by far outperforms the traditional time-similarity measure Euclidean distance w.r.t. the classification accuracy.

9.2.1 General Idea

The general concept of threshold based similarity search is as follows: given two time series X and Y, and an amplitude threshold τ. 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 are above or below the given threshold τ at similar time intervals. 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

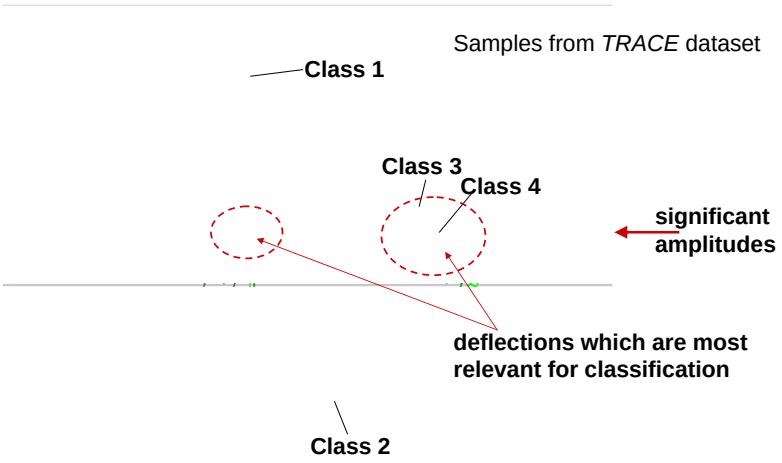


Figure 9.2: Threshold-based classification of time series.

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 similar time" is reduced to compare sequences of time intervals. Usually, the number of intervals is much less than the number of ozone values per ozone sequence 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.

9.2.2 Threshold Based Representation vs. Dimensionality Reduction

Even though dimensionality reduction techniques are generally very important for many similarity search problems, they are not very adequate for threshold queries. The reason is that dimensionality reduction techniques naturally aggregate time series values over time. In contrast, the threshold based representation of time series, i.e. the set of time intervals indicating that the time series is above a given threshold, aggregates time series over the amplitude spectrum. The advantage of threshold queries is that they pre-

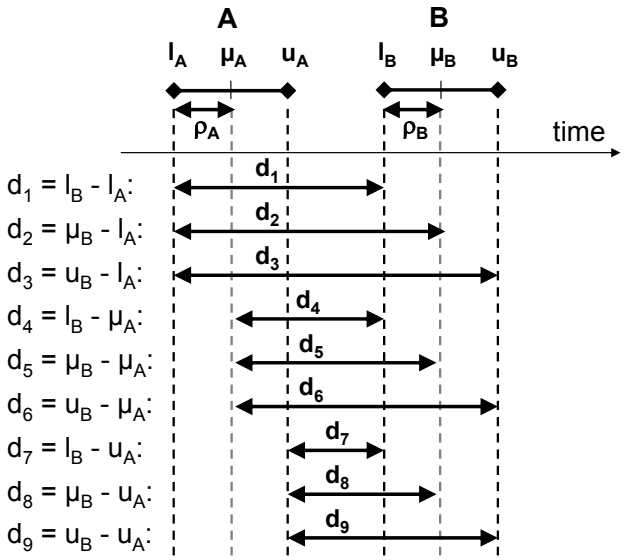


Figure 9.3: The nine basic distances between two intervals A and B.

serve the original time dimension. In addition, threshold queries are designed to suppress certain amplitude spectra which would interfere the results. Dimensionality reduction techniques cannot be directly used for this purposes because they still represent the exact course of the time series rather than intervals of values above a threshold. We can apply data reduction techniques in order to compress the threshold based representation of time series as proposed in [RKBL05]. However, the compressed information does not support the computation of the threshold-based similarity. The compressed representations have to be decompressed before we are able to compute the similarities.

9.2.3 Similarity-Distance Measures for Intervals

It is obvious that the similarity between two points in time can be measured by their distance in time. However, the similarity between two time intervals is harder to model.

Figure 9.3 shows two intervals A and B, each described by four attributes which might be relevant for the similarity measure, the two end points *lower*

l and *upper* u, one midpoint μ and the latitude $\rho = \mu - l$. Since the latitude ρ is not a point in time, usually only the first three attributes l, u and μ are considered. Based on these three attributes, nine basic distances between the two intervals can be defined [Joh06]. A major disadvantage of these distance measures is that none fulfills all metric properties, i.e. positive definite, symmetry and the triangle inequality.

A well-known class of distance measures is the Minkowski-class of metrics. We can use their metric properties for our purpose, in particular for accelerating the query process (cf. Chapter 11). The triangle inequality allows us to apply spatial access methods, like the R^*-tree, in order to speed up similarity queries on intervals. Another advantage is that both parameters that completely define an interval are taken into consideration for similarity computation and not just only one parameter.

The Minkowski metric is a class of models where the distance is given by:

$$d_{AB} = \sqrt[r]{\sum_{i=1}^{n} d_{AB_j}^r},$$

where d_{AB} is the overall distance between A and B, d_{AB_j} is the distance between these two points on the j-th of a set of mutually orthogonal axes in n-dimensional space, and the exponent r $(1 \leq r \leq \infty)$ is the parameter which determines the particular metric.

Since we are using intervals, there are just two mutually independent (orthogonal) axes so that the Minkowski metric can be simplified to

$$d_{AB} = \sqrt[r]{d_{AB_1}^r + d_{AB_2}^r}.$$

The parameter r may be considered as parameter of component weight. If $r = 1$, all components are weighted equally in their effect on the overall distance measure. If r increases, the components become increasingly differentially weighted according to the differences on individual components. If r approaches infinity, the largest of the component distances completely dominates the overall distance measure. In the following, we consider the three most common Minkowski-metric parameter values $r = 1$, 2 and ∞. The case

where $r = 1$ is referred to as the *Manhattan metric*. A value of $r = 2$ provides the standard distance formula for *Euclidean space*. The third metric takes an r-value of ∞, and has been referred to as the *maximum metric*. Under this model, only the largest of the component distances contributes to the overall distance.

9.2.4 Contributions and Outline

The main contributions of this part can be summarized as follows:

- We introduce and formalize the novel concept of threshold-based similarity and define *threshold queries* on time series databases.
- We present a novel data representation of time series and an access method which support threshold queries efficiently.
- We introduce a selective pruning strategy and propose an efficient algorithm for threshold queries based on the new time series representation.

In a broad experimental evaluation, we show that the new type of query yields important information and therefore is useful for several application fields. Furthermore, performance tests show that our proposed algorithms achieve a high speed-up of threshold queries.

Chapter 10 formally introduces our novel similarity measure and proposes a new query type called threshold query. In Chapter 11 we show how time series can be represented in order to support threshold queries for arbitrary threshold values efficiently. An efficient query algorithm based on the proposed representation is described in Chapter 12.

Chapter 10

Threshold Based Similarity Search

In this chapter, we formally introduce the novel concept of threshold queries. We consider one-dimensional (univariate) time series represented by a sequences of N value-time pairs $\langle (x_1, t_1), .., (x_N, t_N) \rangle$ as defined in Section 9.1.

10.1 Threshold-Crossing Time Intervals

We start with the definition of *threshold-crossing time intervals*, an aggregated information of time series used to compute the threshold similarity.

Definition 10.1 (Threshold-Crossing Time Intervals).
Let $X = \langle (x_i, t_i) \in \mathbb{R} \times T : i = 1..N \rangle$ *be a time series and* $\tau \in \mathbb{R}$ *be a threshold value. Then the* threshold-crossing time intervals *of* X *with respect to* τ *is a sequence* $S_{\tau,X} = \langle (l_j, u_j) \in T^2 : j \in \{1, .., M\}, M \leq N \rangle$ *of time intervals such that*

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

Note that we shortly write S_X for threshold-crossing time intervals of a time series Object X if no threshold parameter is specified.

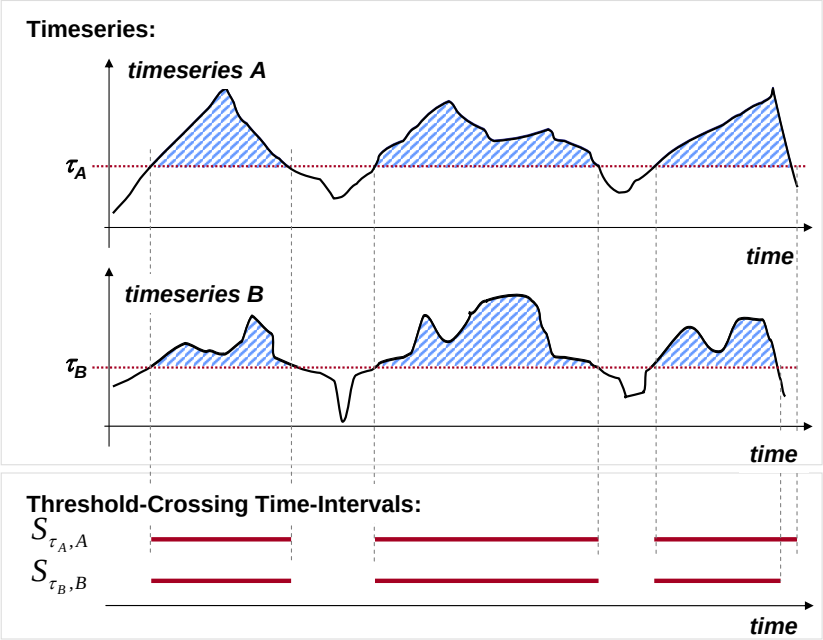


Figure 10.1: Threshold-Crossing Time Intervals.

The example shown in Figure 10.1 depicts the threshold-crossing time intervals $S_{\tau_A,A}$ and $S_{\tau_B,B}$ of the time series A and B respectively. After we have defined which aggregated information we extract from the time series and how we can represent this information, we now present our similarity model based on this representation. In the following, we choose a suitable similarity distance between single intervals which are the basic components of our representation.

10.2 Similarity Model for Time Intervals

There are lot of different possibilities to compute distances between intervals, commenced with the nine basic distance measures depicted in Figure 9.3. Following the discussion in Chapter 9.2.3, we choose the Euclidean distance as it seams the most intuitive one, i.e. two time intervals are defined to be similar if they have "similar" starting and end points. Our approach works with the other Minkowski metrics as well, but we use the Euclidean distance throughout this thesis. An empirical comparison of the different similarity

measures is given in Chapter 13.

Definition 10.2 (Similarity 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. The (dis)similarity between two time intervals is expressed by the distance function* $d_{int} : (T \times T) \times (T \times T) \rightarrow \mathbb{R}$ *which is defined as follows:*

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

10.3 Similarity Model for Threshold-Crossing Time Intervals

For a certain threshold τ a time series object is represented by a sequence of time intervals. Consequently, we need a similarity distance measure for sequences of intervals. For any time series X, the order of the threshold-crossing time intervals of X is naturally given by the interval parameters and the fact that all intervals are disjunctive. For this reason, we can define the threshold-crossing time intervals as a set of intervals without loss of generality. Now, we have to employ a distance measure suitable to set based objects. Several distance measures for set based objects have been introduced in the literature [EM97]. In our approach, we employ the Sum of Minimum Distances (*SMD*). The *SMD* tries to find the best match of each entity of one set with any entity of the other set. This matching of one entity is done independently of the other entities of the same set. If we adopt this to our time series representation, each threshold-crossing of one time series will be matched with the best fitting threshold-crossing time interval of the other time series. Hence, the *SMD* most adequately reflects the intuitive notion of similarity between two threshold-crossing time intervals. As our time interval sets have different cardinalities, we slightly modify the *SMD* by normalizing the distance with the cardinalities of the interval sets. The *threshold-distance* d_{TS} based on the normalized *SMD* [KS04] is defined as follows:

Definition 10.3 (Threshold-Distance).
Let X *and* Y *be two time series and* S_X *and* S_Y *be the corresponding threshold-*

crossing time intervals. Then the threshold distance is defined as follows:

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

As mentioned above, 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 set-valued distance measure has further advantages. Time series having similar shapes, i.e. showing a similar behavior, may be transformed into threshold-crossing time intervals of different cardinality. Since the above distance measure does not take the cardinalities of the interval sequences into account, it is adequate for our purpose. Another advantage is that the distance measure mainly considers local similarity. This means that for each time interval of one time series only its closest counterpart of the other time series is taken into account. Consequently, local similarity related to the threshold crossing will be detected and incorporated into the overall similarity distance.

The threshold-distance according to a certain threshold τ is also called "*τ-similarity*". We use these both expressions alternately in the remainder of this thesis.

10.4 Similarity Queries Based on Threshold Similarity

Finally, based on our new similarity model introduced above, we can define novel similarity queries for time series. The most prominent similarity queries are the *distance-range query* and the *k-nearest-neighbor query*. The distance-range query reports for a given query object Q and a specific range $\varepsilon \in \mathbb{R}_0^+$ those objects of which their similarity distance to Q is smaller or equal to ε. The *k-nearest-neighbor query* reports for a given query object Q and a specific parameter $k \in \mathbb{N}^+$ the k most closest objects to Q according to the used similarity distance measure. Applying our similarity model, we can incorporate these two similarity queries as follows:

Definition 10.4 (Threshold-Based ε-Range Query).
Let $\mathcal{D}$ denote the domain of time series objects. The threshold-based ε-range query *consists of a query time series $Q \in \mathcal{D}$, a query threshold $\tau \in \mathbb{R}$ and a parameter $\varepsilon \in \mathbb{R}_0^+$. It reports the set $TQ_\varepsilon^{\varepsilon-range}(Q,\tau) \subseteq \mathcal{D}$ of time series from $\mathcal{D}$ such that*

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

Similar to the distance-range similarity query we can adopt the nearest-neighbor similarity query as well.

Definition 10.5 (Threshold-Based k-Nearest-Neighbor Query).
Let $\mathcal{D}$ be the domain of time series objects. The threshold-based k-nearest-neighbor query *consists of a query time series $Q \in \mathcal{D}$, a query threshold $\tau \in \mathbb{R}$ and a parameter $k \in \mathbb{N}^+$. It reports the smallest set $TQ_k^{k-NN}(Q,\tau) \subseteq \mathcal{D}$ of time series objects that contains at least k objects from $\mathcal{D}$ such that*

$$\forall X \in TQ_k^{k-NN}(Q,\tau), \forall Y \in \mathcal{D} \backslash TQ_k^{k-NN}(Q,\tau) :$$

$$d_{TS}(S_{\tau,X}, S_{\tau,Q}) < d_{TS}(S_{\tau,Y}, S_{\tau,Q}).$$

Note that we call both similarity queries defined above simply *Threshold Query* if the specific query type (ε-range or k-nearest-neighbor) does not make any difference in the context. Furthermore, if there is no specification of the parameter k for the threshold-based k-nearest-neighbor query, we assume that $k = 1$.

Chapter 11

Threshold Based Indexing

A straightforward approach for executing a threshold query is to read sequentially each time series X from the database. Then compute the corresponding threshold-crossing time interval sequence $S_{\tau,X}$ which is used to compute the threshold-similarity distance $d_{TS}(S_{\tau,X}, S_{\tau,Q})$. Finally, we report those time series whose distance $d_{TS}(S_{\tau,X}, S_{\tau,Q})$ fulfill the query predicate. However, if the time series database contains a large number of objects and the time series are reasonably large, then this type of performing the query becomes unacceptably expensive. As a solution, in this chapter we introduce an access method which is convenient for the proposed time series representation. In particular, it allows an efficient access to the threshold-crossing time intervals of the time series.

We present two approaches for the management of time series data, both of which efficiently support threshold queries. The key point of the proposed 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 sufficient to compute the query results. At query time, we only need the information at which time frames the time series exceed and fall below the specified threshold. The ability to access only the relevant parts of the time series at query time would save a lot of I/O cost. The basic concept of our approach is to pre-compute the threshold-crossing time intervals $S_{\tau,X}$ for each time series object X and store them on disk in such a way that it can

be accessed efficiently.

For the sake of clarity, we first present a straightforward approach, assuming that the threshold value τ is constant for all queries and known in advance. Afterwards, we present the general approach which supports arbitrary choice of τ at query time.

11.1 Managing Threshold-Crossing Time Intervals with Fixed τ

Let us assume that the query threshold τ is fixed for all queries. Then, we can compute the corresponding $S_{\tau,X}$ for every time series X in advance. 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 [KPS01]. However, like the other interval access methods, the RI-tree supports intersection queries on interval data very well but does not efficiently support the computation of similarity distances between intervals or sequence of intervals according to our similarity model. Moreover, the existing approaches cannot be applied to our general approach where we assume that τ is not fixed. Contrary, we propose a simple solution which efficiently supports similarity queries on intervals (or more generally sequences of intervals) and which can be easily extended to support queries with arbitrary τ.

Time intervals can also be considered as points in a two-dimensional plane [GG98]. In the following, we refer to this plane as *time-interval plane*. The one-dimensional intervals (*native space*) are mapped to the time-interval plane by taking their start and end points as two-dimensional coordinates. An example is depicted in Figure 11.1. This representation has some advantages for the efficient management of intervals:

- First, the distances between intervals are preserved.
- Second, the position of large intervals which are located within the

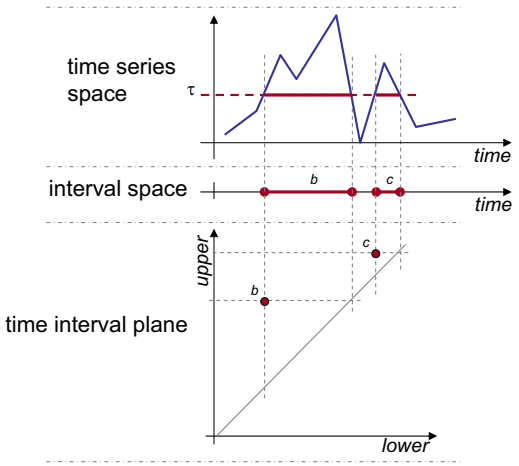


Figure 11.1: Mapping of Time Intervals to the Time Interval Plane.

upper-left region substantially differs from the position of small intervals (located near the diagonal).

- However, the most important advantage is that the Euclidean distance in this plane corresponds to the similarity of intervals according to Definition 10.2.

The complete set of threshold-crossing time intervals of a time series is represented by a set of two-dimensional points in the time interval plane. The transformation chain from the original time series to the point set in the time interval plane is depicted in Figure 11.1.

In order to efficiently manage the point sets of all time series objects, we can use any index structure which can handle point data, e.g. the R^*-tree [BKSS90]. In particular, the R^*-tree is very suitable for managing points in low-dimensional spaces which are not equally distributed. Additionally, it well supports the distance-range and nearest-neighbor query which will be required to perform the threshold queries efficiently. Note that since each object is represented by a set of points in the time interval plane, it is referenced by the index structure multiple times. This property has to be taken into account for the query process. Details of our query algorithm are be presented later in Chapter 12.

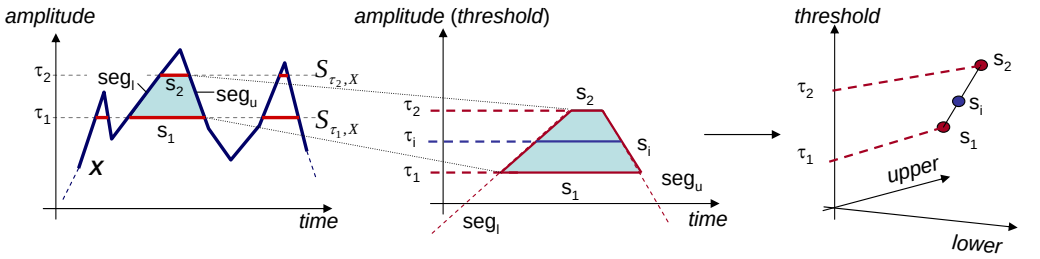


Figure 11.2: Time Intervals in Parameter Space for Arbitrary Threshold.

11.2 Managing Threshold-Crossing Time Intervals for Arbitrary τ

In contrast to the first approach, we now describe how to manage efficiently threshold queries for arbitrary threshold values τ. First, we have to extend the transformation task of the simple approach that the time-interval plane representations of the threshold-crossing time intervals of the time series are available for all possible threshold values τ. Therefore, we extend the time interval plane by one additional dimension which indicates the corresponding threshold values. In the following, we call the extended time-interval space "*parameter space*". A two-dimensional plane in the parameter space spanned by the two interval-endpoint dimensions at a certain threshold τ is called *time-interval plane* of threshold τ.

In the following, we assume that the time series objects are linearly interpolated, i.e. consecutive time series values (x_i, t_i) and (x_{i+1}, t_{i+1}) are connected by a two-dimensional segment $((x_i, t_i), (x_{i+1}, t_{i+1}))$ in the time-amplitude space (native space). Hence, the time series consists of a sequence of segments which starting and end points are defined by the time series values and the corresponding time slots (cf. Figure 11.2(a)).

Lemma 11.1.
Let $X \in \mathcal{D}$ be a time series and $S_{\tau_1,X}$ and $S_{\tau_2,X}$ be two threshold-crossing time intervals of X, where w.l.o.g. $\tau_1 < \tau_2$. Let $s_1 \in S_{\tau_1,X}$ and $s_2 \in S_{\tau_2,X}$ be two time intervals which start points lie on one segment seg_l of the linearly

interpolated time series and the end points lie on another segment seg_u*. Then all threshold-crossing time-intervals* $S_{\tau_i,X}$ *with* $\tau_1 \leq \tau_i \leq \tau_2$ *contains 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

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

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

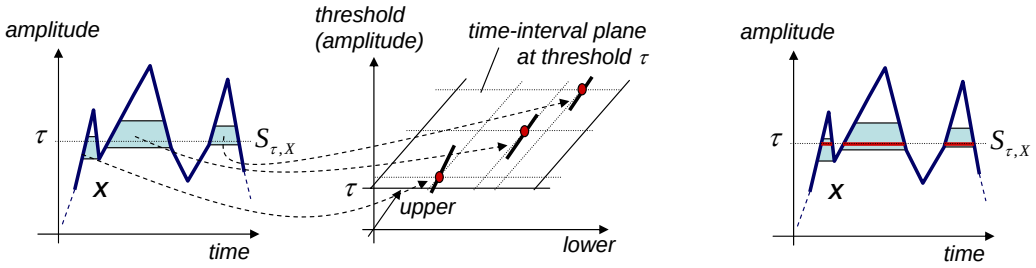
and

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

□

Let us consider the following example shown in Figure 11.2 in order to clarify Lemma 11.1. Figure 11.2(a) shows a linearly interpolated time series X. Let the time interval s_1 be an entity of the threshold-crossing time intervals $S_{\tau_1,X}$ and s_2 be an entity of $S_{\tau_2,X}$. Both 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 intervals $S_{\tau_i,X}$ which are between $S_{\tau_1,X}$ and $S_{\tau_2,X}$, 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 11.2(b). Given the time intervals s_1 and s_2 transformed into 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 11.2(c).

Following Lemma 11.1, all time intervals which are bounded by the same time series segments can be transformed into one segment in the parameter



a) building time interval groups (native space)

b) determination of $S_{\tau,X}$ in parameter space

c) threshold-crossing time intervals (native space)

Figure 11.3: Determination of threshold-crossing time intervals from parameter space.

space. In order to represent all threshold-crossing time intervals of one time series in the parameter space, we have to identify all sets of time intervals where each set contains those time intervals which are bounded by the same time series segment in the native space (cf. Figure 11.3(a)). Each set is then transformed into a three-dimensional segment in the parameter space (cf. Figure 11.3(b)).

The entire 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 one threshold-crossing time interval $S_{\tau,X}$ can be retrieved by intersecting the parameter-space segments which correspond to X with the two-dimensional time-interval plane at threshold *tau* (cf. Figure 11.3(b)). The resulting intersection points correspond to the time intervals of $S_{\tau,X}$ as depicted in Figure 11.3(c).

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

- We try to represent the entire set of threshold-crossing time intervals by the smallest number of segments in the parameter space.

- Then, 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.

11.3 Trapezoid Decomposition of Time Series

If we consider the time intervals from all possible threshold-crossing time intervals, the following observation can be made:

Lemma 11.2.
All intervals from 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. *Due to Definition 10.1, all values of X within a time interval from threshold-crossing time intervals $S_{\tau,X}$ are greater than the corresponding threshold value τ. Let us assume that the time series segment seg_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 lower than τ which contradicts definition 10.1. The validity of Lemma 11.2 w.r.t. the right bounding segment can be shown analogously.* □

Due to Lemma 11.2, 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 which 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 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:

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

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

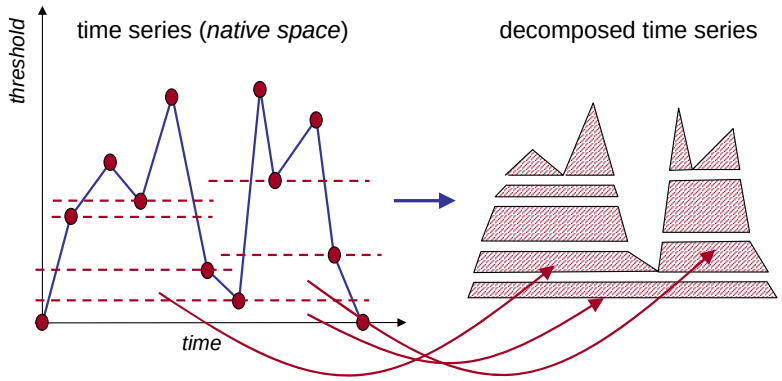


Figure 11.4: Time Series Decomposition.

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 space below the time series into a set of disjunctive trapezoids. A time series object can be considered as half-open uni-monotone polygon in the time-amplitude plane (native space). In the area of computational geometry, there are several sweep-line based polygon-to-trapezoid decomposition algorithms known [FM84] which can be processed in $O(n \cdot logn)$ time w.r.t. the number of vertices. According to the sweep-line based decomposition algorithms, we can develop an algorithm which decomposes a time series into the desired set of trapezoids. Figure 11.4 shows an example of a time series which is decomposed into the set of trapezoids. Since the time series values naturally are already sorted by time, our decomposition algorithm can be processed in linear time w.r.t. the length of the sequence. Our decomposition algorithm is depicted in Figure 11.6.

Let us illustrate the decomposition algorithm by means of the following example depicted in Figure 11.5. In the *for*-loop we sequentially process the time series segments s_1,..,s_{11}. As s_1 and s_2 have positive slopes, i.e. they open trapezoids, we push them on the stack. Next, we consider the segment $next\_seg = s_3$ which has a negative slope, i.e. we can close the first trapezoids. At this time (see step (1)), the stack contains the segments s_2, s_1. We pop s_2 from the stack and compute and output the first trapezoid T_1 by means of

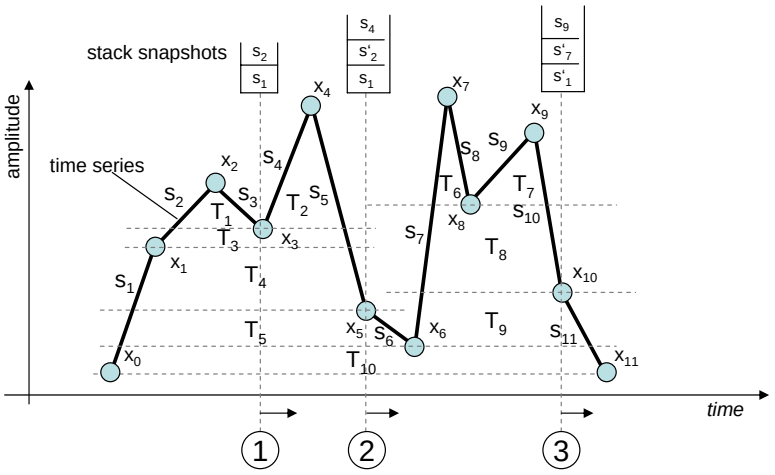


Figure 11.5: Time Series Decomposition Example.

the procedure *compute_trapezoid*(s_2,s_3). Then we cut the segment s_2 at the amplitude value $s_3.x_e = x_3$ and push the cut 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, cut s_1 w.r.t. x_5 and push the cut segment s_5' back on the stack. The algorithm continues with processing segment s_6 and so on.

11.4 Parameter Space Indexing

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 points and rectangles, we represent the three-dimensional segments by rectangles where the segments correspond to one of the diagonals of the rectangles.

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

```
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, ⊥, t_0, x_0)); //left time series border on stack
  TS.append((t_max,⊥)); //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 ⇒ open trapezoid
      stack.push(next_seg);
    else if (x_{i+1} > x_i), then //segment with negative slope ⇒ close trapezoids
      while (stack.top.x_s ≥ 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;
}

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 τ_bottom = max(seg1.x_s,seg2.x_e);
  float τ_top = min(seg1.x_e,seg2.x_s);
  Time t_s^bottom = intersect(seg1,τ_bottom);
  Time t_e^bottom = intersect(seg2,τ_bottom);
  Time t_s^top = intersect(seg1,τ_top);
  Time t_e^top = intersect(seg2,τ_top);
  output(Trapezoid(t_s^bottom,t_e^bottom,τ_bottom,t_s^top,t_e^top,τ_top));
}
```

Figure 11.6: Linear time series decomposition.

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 suffice 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 $((x_l, y_u, z_l), (x_u, y_l, z_u))$.

Chapter 12

Threshold Based Query Processing

In this chapter, we present an efficient algorithm for our two threshold queries, the threshold-based ε-range query and the threshold-based k-nearest-neighbor query. Both consist of a query time series Q and a query threshold τ, as well as the query type specific parameters ε and k (cf. Definition 10.4 and 10.5).

Given the query threshold τ, the first step of the query process is to extract the threshold-crossing time intervals $S_{\tau,Q}$ from the query time series Q. This can be done by one single scan through the query object Q. Next, we have to find those time series objects X from the database which fulfill the query predicate, i.e. the threshold distance $d_{TS}(S_{\tau,Q}, S_{\tau,X}) \leq \varepsilon$ in case of the threshold-based ε-range query or the objects belong to the k closest objects from Q w.r.t. $d_{TS}(S_{\tau,Q}, S_{\tau,X})$ in case of the threshold-based k-nearest-neighbor query.

A straightforward approach for the query process would be as follows: 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 and evaluate the query predicate in order to build the result set. 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.

12.1 Preliminaries

In the following, we assume that each time series object X is represented by its threshold-crossing time intervals $S_X = S_{\tau,X} = x_1, .., x_N$ which correspond to a set of points lying on the time-interval plane $\mathcal{P}$ within the parameter space at query threshold τ. Hence, S_X denotes a set of two-dimensional points[1]. Furthermore, let $\mathcal{D}$ denote the set of all time series objects and $\mathcal{S}$ 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 proposal, we need the 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 12.1 (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 12.2 (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

[1] For the description of the threshold-crossing time intervals in the native space (set of time intervals) and in the parameter space (set of points) we use the same notion S_X.

Table 12.1: Notations and operations on time interval sets.

$\mathcal{D}$	Set of all time series objects (database).
$\mathcal{P}$	Time-interval plane along the lower-upper dimensions at 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 12.1).
$NN_S(q)$	The nearest neighbor of q in S (cf. Definition 12.2).
$NN_{k,S}(q)$	The k^{th} nearest neighbor of q in S (cf. Definition 12.2).
$kNN_S(q)$	The k nearest neighbors of q in S (cf. Definition 12.2).

time intervals and $k \in \mathbb{N}^+$ *be the ranking parameter. The* k-nearest-neighbor $NN_{k,S}(q) \in \mathcal{P}$ $(k \leq N)$ *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.

In Table 12.1 we summarized the most important parameters required throughout the following sections.

12.2 Pruning Strategy for Threshold Queries

In this section, we show that we do not need to access all the time intervals in $\mathcal{S}$ in order to compute the threshold queries for any time series object in $\mathcal{D}$. That means that we can prune some objects without accessing them. The solution for our pruning strategy is based on the following two observations:

Lemma 12.1.
Let $S_Q = \{q_1, .., q_{M_Q}\} \subseteq \mathcal{P}$ *be the set of points which correspond to the*

query object Q. Then, each database object $X \in \mathcal{D}$ represented by $S_X = \{x_1, .., x_{M_X}\} \subseteq \mathcal{P}$ 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 $TQ_\varepsilon^{\varepsilon-range}(Q, \tau)$, formally:

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

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:

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

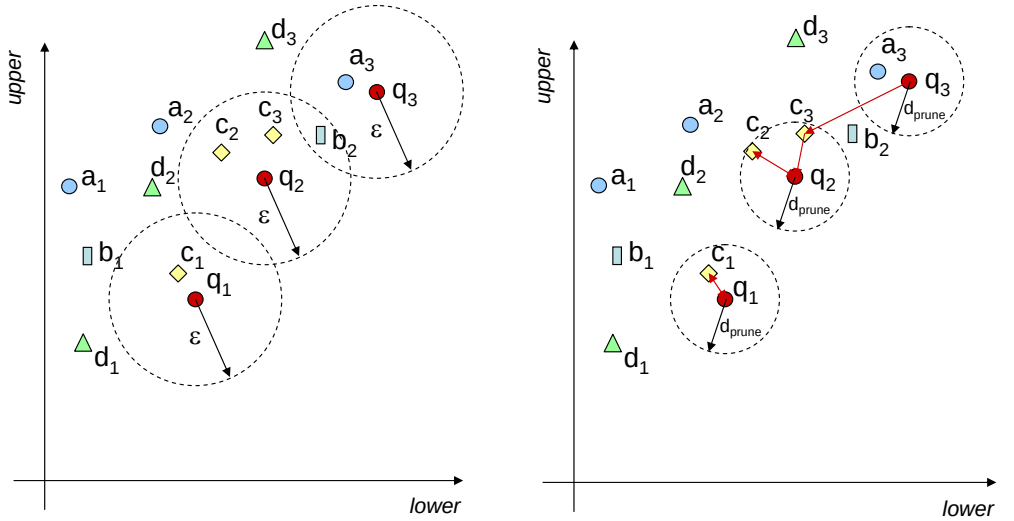
$$> \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot \sum_{q \in S_Q} \varepsilon + \frac{1}{|S_X|} \cdot \sum_{s \in S_X} \varepsilon \right) = \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot |S_Q| \cdot \varepsilon + \frac{1}{|S_X|} \cdot |S_X| \cdot \varepsilon \right) = \varepsilon.$$

□

An example is depicted in Figure 12.1(a) which shows the threshold-crossing time intervals $S_Q = \{q_1, q_2, q_3\}$ for the query object Q and the threshold-crossing time intervals $S_A = \{a_1, a_2, a_3\}$, $S_B = \{b_1, b_2\}$, $S_C = \{c_1, c_2, c_3\}$ and $S_D = \{d_1, d_2, d_3\}$ of the four database objects A, B, C and D, respectively. Due to Lemma 12.1, object D cannot be in the result set of $TQ_\varepsilon^{\varepsilon-range}(Q, \tau)$. The same holds for all the other objects having no time interval instance within the three ε-range circles.

Similar to the ε-range query, we can also identify pruning candidates for the k-nearest-neighbor query with the following observation. Here, w.l.o.g. we assume that the ranking parameter k is set to 1.

Lemma 12.2.
Let $S_Q = \{q_1, .., q_{M_Q}\} \subseteq \mathcal{P}$ be the set of points which corresponds to the query object Q. Furthermore, let d_{prune} be the threshold distance $d_{TS}(S_Q, S_X)$



(a) threshold based ε-range query

(b) threshold based k-nearest-neighbor query

Figure 12.1: Properties of threshold queries w.r.t. object pruning.

between Q and any database object X. Then each database object $Y \in \mathcal{D}$ represented by $S_Y = \{x_1, .., x_{M_X}\} \subseteq \mathcal{P}$ 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 $TQ_1^{kNN}(Q, \tau)$, formally:

$$\forall s \in S_Y, \forall q \in S_Q : s \notin Q_{d_{prune}}^{\varepsilon-range}(q) \Rightarrow Y \notin TQ_1^{kNN}(Q, \tau).$$

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 10.5, Y cannot be in the result set $TQ_1^{kNN}(Q, \tau)$.

□

Let us illustrate Lemma 12.2 by means of our example depicted in Figure 12.1(b). Let d_{prune} be the threshold distance $d_{TS}(S_Q, S_X)$ between Q and X. Then object B cannot be a result of $TQ_1^{kNN}(Q, \tau)$, because all distances $d_{int}(q, b)$ between any time interval q of S_Q and any time interval b of S_B exceeds d_{prune}, and thus, the overall threshold distance $d_{TS}(S_Q, S_B)$ must be greater than $d_{prune} = d_{TS}(S_Q, S_X)$. The same holds for object D. All the other objects have no time interval instance within the three d_{prune}-range circles.

Based on the two lemmas above, we can develop efficient threshold queries using the R^*-tree for the efficient organization of the threshold-crossing time intervals in the parameter space. The proposed algorithms for our two threshold queries aim at keeping the cost required for the expensive set-valued distance computation as low as possible. For this reason, both algorithms follow the multi-step query paradigm. They first try to reduce conservatively the result-set candidates by a cheap filter step and afterwards retrieve the exact result by performing the expensive refinement step on the reduced candidate set.

12.3 Threshold-Based ε-Range Query Algorithm

The algorithm for the threshold-based ε-range query is depicted in Figure 12.2. 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 efficiently 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

```
ALGORITHM TQ^{ε-range}(S_Q, ε, 𝒟, 𝒮)
BEGIN
  result_set := ∅;
  candidate_set := ∅;
  FOR EACH q ∈ S_Q DO
    candidate_set := candidate_set ∪{X ∈ 𝒟 | S_X ∩ R_{ε,𝒮}(q) ≠ ∅}; // filter step
  END FOR;
  FOR EACH X ∈ candidate_set DO
    IF d_TS(S_Q, S_X) ≤ ε THEN // refinement step
      result_set := result_set ∪X;
    END IF;
  END FOR;
  report result_set;
END
```

Figure 12.2: Threshold-based ε-range query algorithm.

refine each candidate X by computing the threshold distance to Q and put them into the result set if $d_{TS}(S_Q, S_X) \leq \varepsilon$.

12.4 Filter Distance for the Threshold Similarity

Before we start with the algorithm for the threshold-based k-nearest-neighbor query, we have to develop a suitable filter distance for our pruning strategy in order to reduce the expensive refinements as much as possible. The performance of our algorithm mainly depends on the quality of our filter. Primarily, the filter step should be at least faster than the refinement step. Furthermore, for our purpose the filter should:

- retrieve the most promising threshold-based k-nearest-neighbor first.
- be conservative but as accurate as possible w.r.t. the threshold similarity distance.

Both properties aim at pruning many false candidates very early. The first one enables an early detection of a suitable pruning distance d_{prune} which should be as low as possible, while the second property avoids false dismissals and aims at detecting the true drops as early as possible. Since the filter should be conservative, we require a lower bound criterion for the threshold distance.

12.4.1 Lower Bounding Threshold Distance

Now, we introduce a lower bound criterion for the threshold distance d_{TS} on the basis of 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, i.e. only partial information of the false candidates suffices to prune the corresponding object from the candidate list. 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 threshold-crossing time intervals corresponding to the query object and $S_X \subseteq \mathcal{P}$ corresponding to 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 the parts of the threshold distance which can be written now as follows:

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

We use two auxiliary functions $\kappa_k(q_i)$ and $\bar{\kappa}_k(S_Q)$ which help us to divide our database objects into two sets. $\kappa_k(q_i) \subseteq \mathcal{D}$ denotes the set of all objects X

which has at least one entity $x \in S_X$ 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 12.3.
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 12.2 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 following lemma is a generalization of Lemma 12.3 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 12.4.
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 estimated by the following formula:

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

Proof. *Let $X \in \mathcal{D}$ be any database object and Q be the query object. According to Definition 12.2 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, $d_1^{min}(Q, X) \leq \sum_{q \in S_Q} d_{int}(q, NN_X(q)) = D_1(S_Q, S_X)$. □

Furthermore, we can also lower bound the distance $D_2(S_Q, S_X)$ as follows:

Lemma 12.5.
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 12.2 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.

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

□

12.4.2 Pruning Based on Lower Bounding Distance

In this section, we show which objects can be pruned, based on the information retrieved so far, i.e. without accessing the complete object information.

Let us assume that the object $Q \in \mathcal{D}$ 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 is not refined, yet. Then we can prune Y for the threshold query $TQ_1^{k-NN}(Q, \tau)$ iff

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

If we now apply Lemma 12.4 and 12.5, we can prune Y iff

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

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

From the computational point of view, we should distinguish the objects in $\bar{\kappa}_k(S_Q)$ from the other objects. Now we can infer the two statements below which directly follows from Lemma 12.4 and 12.5:

Lemma 12.6.
All objects Y which are in the set $\bar{\kappa}_k(S_Q)$, i.e. $Y \in \bar{\kappa}_k(S_Q)$, can be pruned iff

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

Lemma 12.7.
All objects Y which are <u>not</u> in the set $\bar{\kappa}_k(S_Q)$, i.e. $Y \notin \bar{\kappa}_k(S_Q)$, can be pruned iff

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

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 which we have retrieved at least one time interval so far. These objects are organized in a list which will be possibly expanded in each iteration. We call this list *object list*. Now, we can compute the lower bounding distance for all objects in the object list and prune them according to Lemma 12.7. 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 12.6). As soon as we have found a pruning distance d_{prune} for which Lemma 12.6 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 next to be refined is found. In doing so, 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 have to prune all objects $Y \in \mathcal{D} \backslash \{X\}$ from the object list according to Lemma 12.7. The pruned objects can be omitted from the remaining search steps. 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

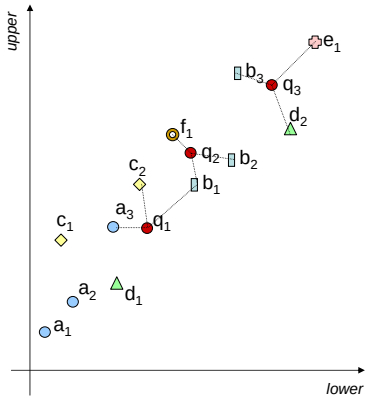
for the most similar one (in case of the threshold-based 1^{st}-nearest-neighbor query), have been pruned from the object list.

A demonstrative example for the pruning process is included in the next section which presents the threshold-based nearest-neighbor query algorithm.

12.5 Threshold-Based Nearest-Neighbor Query Algorithm

The query algorithm of the TQ_1^{k-NN} query is depicted in Figure 12.5. 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]. We maintain for each $q \in S_Q$ a priority queue, each storing the accessed R^*-tree nodes in ascending order of their distances to the corresponding query point q. Note that the R^*-tree indexes the three-dimensional segments in the parameter space, but 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 12.6.

Furthermore, we update the object list *object_distList* which keeps for each object X accessed so far an array which stores for each $q \in S_Q$ the nearest-neighbor distance $NN_X(q)$ in case this information is already available. For this reason, we search for each time interval s retrieved by $NN_{k,\mathcal{S}}(q)$ the corresponding object in the object list *object_distList* and store the distance $d_{int}(s, q)$, iff, w.r.t. q and X there is no distance available from earlier iterations. As soon as we retrieved for an object X all $NN_X(q)$-distances



(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

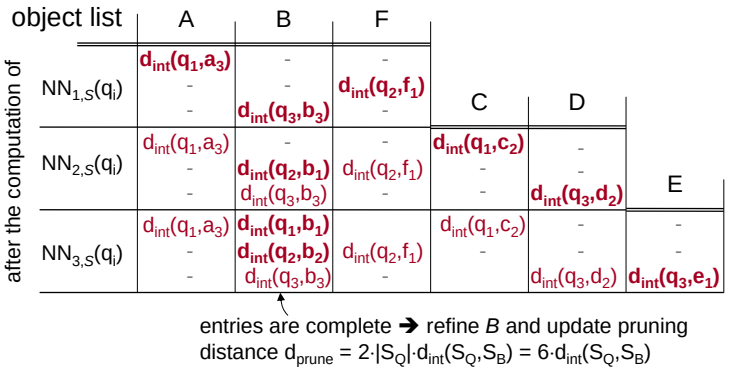
Figure 12.3: Example of the threshold-based nearest-neighbor query.

for all $q \in S_Q$, 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 update the pruning distance d_{prune} and remove X from the object list. By means of the new pruning distance, we decide whether the previous *result* has to keep, and then prune X. If X is closer than the previous *result* according to the threshold distance, we replace the previous *result* with X.

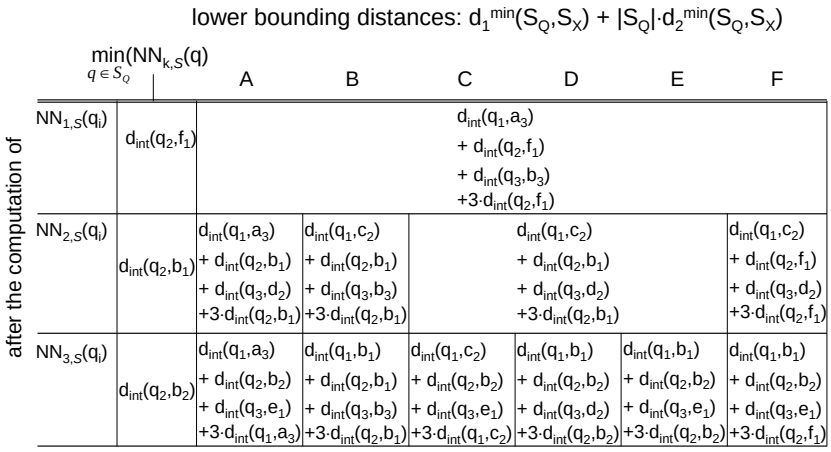
Next, we compute the lower-bounding distance *lb_dist* for each object in the object list and prune those objects for which *lb_dist*$\geq d_{prune}$ holds.

As long as the object list is not empty, we repeat this procedure in the next iterations. Finally, we get 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. Except these little modifications, the algorithm for threshold-based k-nearest-neighbor queries hardly differs from our threshold-based nearest-neighbor-query algorithm.



(a) Object list



(b) Lower-bounding distance computation

Figure 12.4: Step-wise lower-bounding distance computation of the threshold-based nearest-neighbor query example.

We now step through our algorithm with the query example depicted in Figure 12.3. In our example, the query consists of three time-interval plane points $S_Q = \{q_1, q_2, q_3\}$. Figure 12.3(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 $S_A = \{a_1, a_2, a_3\}$, $S_B = \{b_1, b_2, b_3\}$, $S_C = \{c_1, c_2\}$, $S_D = \{d_1, d_2\}$, $S_E = \{e_1\}$ and $S_F = \{f_1\}$. The table depicted in Figure 12.3(b) shows the results of the first three query iterations of the incremental k-nearest-neighbor queries $NN_{1,S}(q_i)$, $NN_{2,S}(q_i)$ and $NN_{3,S}(q_i)$. The state of the corresponding object list *object_distList* after each iteration is shown

in Figure 12.4(a). Figure 12.4(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 12.7. In this case, all database objects have equal lower bounding distances as depicted in Figure 12.4(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 in order to check if something can be pruned.

Next, we retrieve b_1, b_2 and e_1. After updating the object list, we detect that its entries are complete for object B, i.e. we have found for each query time-interval the corresponding nearest neighbor w.r.t. object 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 12.7 and 12.6.

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 experiments (cf. Section 13.3) 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.

```
TYPE Q_ARRAY[N] := ARRAY[N] of DOUBLE;
ALGORITHM TQ_1^{k-NN}(S_Q, D, 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 := +∞
  k := 0;
  REPEAT
    k := k + 1;
    act_kNN = fetch-next(S_Q,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 ∈ object_distList DO /*refinement step*/
      IF (obj.DIST.complete() = true) THEN
        d'_prune = 2 · |S_Q| · 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 ∈ object_distList DO
      lb_dist := d_1^min(S_Q, S_Y) + |S_Q| · d_2^min(S_Q, S_Y);
      IF (lb_dist ≥ 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 12.5: Threshold-based nearest-neighbor query algorithm.

Chapter 13

Experimental Evaluation

In this chapter, we present the results of experiments performed on a broad selection of different real-world and artificial time series datasets.

First, we demonstrate in Section 13.3 that threshold queries can be efficiently performed by using our proposed time series decomposition and query concept. In particular, we evaluated our time series decomposition approach (cf. Chapter 11) and our query processing strategy (cf. Chapter 12) by measuring the selectivity, execution time and pruning power of similarity queries. Furthermore, we experimentally show in Section 13.4 that our novel threshold-based similarity measure can be very valuable for mining tasks like the classification of time series. We demonstrate that our approach achieves a high classification accuracy on a wide range of different time series datasets which are well established in the time series mining community. However, our approach can be processed significantly faster than the DTW [BC94] approach. For some datasets our approach even outperforms dynamic time warping with respect to classification accuracy.

13.1 System Environment

All experiments were performed on a workstation featuring a 1.8 GHz Opteron CPU and 8 GB RAM. We used a disk with a transfer rate of 100 MB/sec, a

Table 13.1: Summary of the Temporal Test Datasets.

Dataset	# time series	length	# classes
AUDIO	$7 \cdot 10^5$	300	-
SCIEN_ENV	-	48	-
SCIEN_GEX	$6 \cdot 10^3$	24	-
GunX	200	150	2
Trace	200	275	4
CBF	150	127	3
SynCtrl	600	6000	6

seek time of 3 ms and a latency delay of 2 ms. Furthermore, we set the disk block size to 8 KB. Performance is presented in terms of the elapsed time including I/O time and CPU time.

13.2 Datasets

We used several real-world and synthetic datasets in our evaluation, one audio dataset (*AUDIO*), two scientific datasets (*SCIENTIFIC*) and a set of well-established test datasets often used as a benchmark (*STANDARD*). We applied our approach to both scientific datasets and on the four standard datasets in order to show the effectiveness of threshold queries. A short summary of the characteristics of all datasets is given in Table 13.1.

13.3 Performance Results

We compared the efficiency of our proposed approach, in the following denoted by 'R_{Par}', for answering threshold queries, using one of the following techniques.

The first competing approach, denoted by 'Seq_{Nat}', works on the native time series data. It corresponds to a sequential processing of the native data.

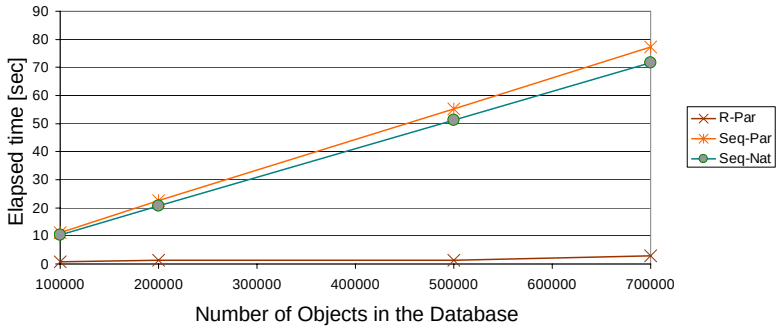
The threshold-crossing time intervals of each time series was computed at query time.

The second competitor, denoted by 'Seq_{Par}', works on the parameter space rather than on the native data. It assumes that all time series objects are already transformed into the parameter space, but without using any index structure. At query time, this method requires a sequential scan over all segments of the parameter space.

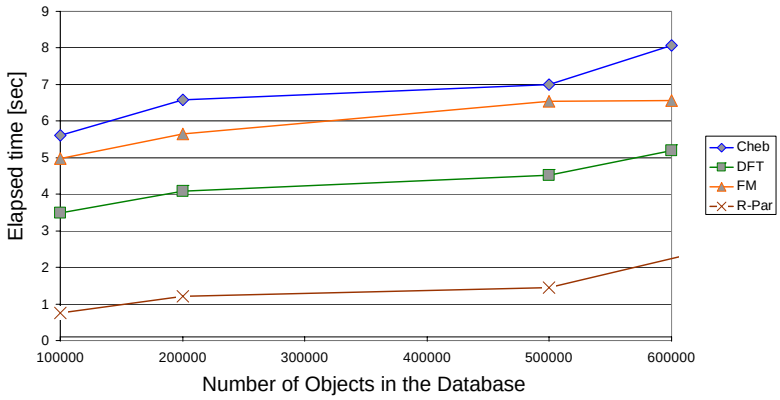
We further compare the performance of our approach to traditional similarity search approaches based on the following dimension reduction methods: Chebyshev Polynomials (*Cheb*) [CN04], Discrete Fourier Transformation (*DFT*) [AFS93] and Fast Map (*FM*) [FL95]. In particular, we implemented the algorithm proposed by Seidl and Kriegel in [SHP98] which adapts the GEMINI framework (cf. Section 2.3) 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-based nearest-neighbor queries. The presented results are the average results of these queries.

First, we performed threshold queries against database instances of different sizes to measure the influence of the database size to the overall query time. The elements of the databases are time series of fixed length $l = 50$. Figure 13.1 exhibits the performance results for each database. In Figure 13.1(a) it is shown that the performance of both approaches Seq_{Nat} and Seq_{Par} significantly decreases with increasing database size, whereas our approach R_{Par} scales very well, even for large databases. Furthermore, our approach shows similar scalability behavior than the three dimensionality reduction approaches *Cheb*, *DFT* and *FM* as depicted in Figure 13.1(b). Yet, our approach even outperforms them by a factor of 4 to 5.



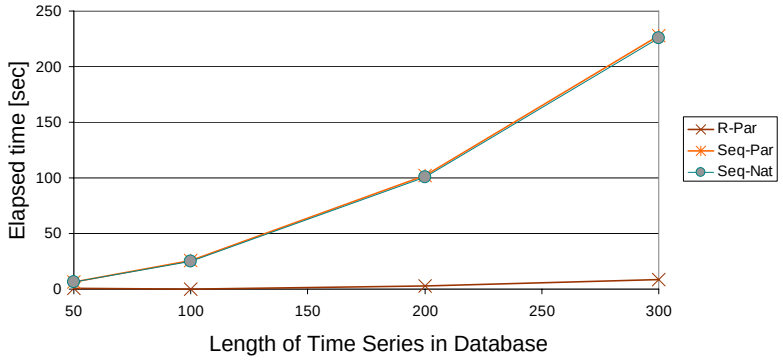
(a)



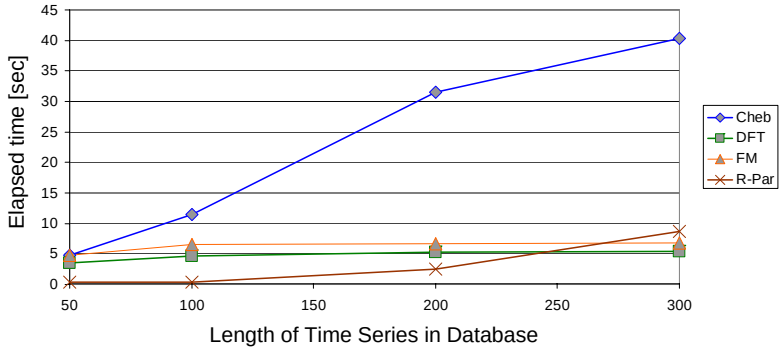
(b)

Figure 13.1: Scalability of the *threshold-query* algorithm against database size.

Second, we explored the impact of the length of the query object and the time series in the database. The results are shown in Figure 13.2. Again, our technique outperforms the competing approaches Seq_{Nat} and Seq_{Par} whose cost increase very quickly due to the expensive distance computations (cf. Figure 13.2(a)). 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 13.2(b). If the length of the time series objects exceeds 200, then both approaches *DFT* and *FM* scale better than 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



(a)

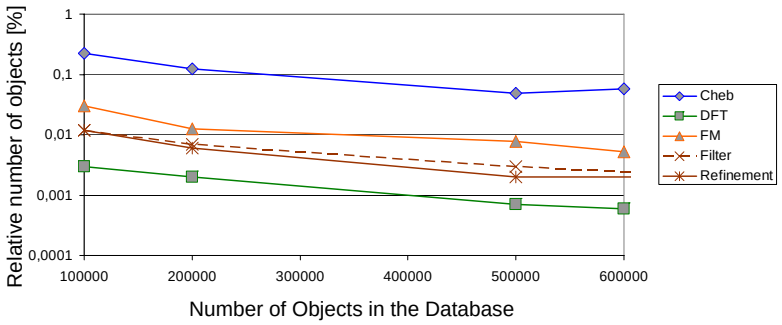


(b)

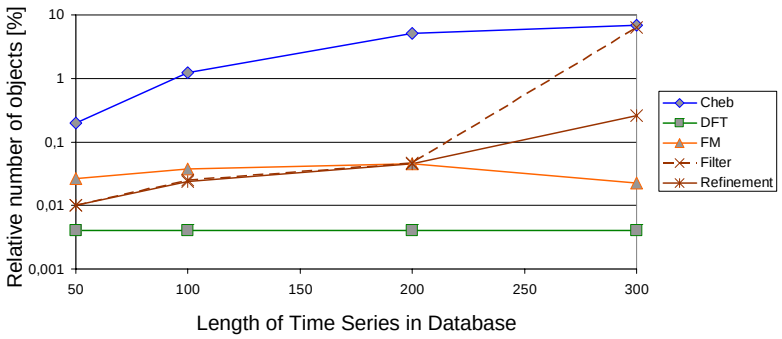
Figure 13.2: Scalability of the *threshold-query* algorithm against time series length.

length for constant approximation quality. Obviously, the cardinality of our time series representations increases linear with the time series length.

In the next experiment, we demonstrate 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 has to be refined finally, denoted by '*Refinement*'. We will again compare our approach to the three dimensionality reduction methods *Cheb*, *DFT* and *FM*. Figure 13.3(a) and Figure 13.3(b) show the results relatively to the database size and length of the time series objects. Generally, only a very small portion of the candidates has to be refined to report the result. Similar to the dimension reduction



(a) Pruning power for varying database size.



(b) Pruning power for varying time series length.

Figure 13.3: Pruning Power of the threshold-based nearest-neighbor algorithm.

methods, our approach scales well for large databases. For small time series, our approach has a lightly better pruning power than *Cheb* and *FM*. We can observe that the pruning power of our approach decreases with increasing time series length. An interesting point is that the number of candidates to be accessed in the filter step increases faster for larger time series than the number of finally refined candidates. Yet, for the *AUDIO* dataset the *DFT* method shows the best results w.r.t. the pruning power.

Furthermore, we examined the number of nearest-neighbor search iterations of 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 w.r.t. the database size. Nevertheless,

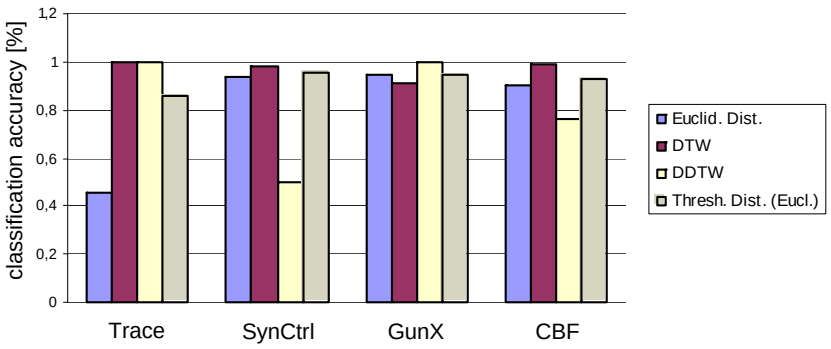


Figure 13.4: Comparison to Traditional Distance Measures.

only a few iterations are required to report the result.

The bottom line is that, with respect to query performance, our approach obviously outperforms by far both competing approaches Seq_{Nat} and Seq_{Par}. However, it is comparable to the three Euclidean distance-based dimensionality reduction methods *Cheb*, *DFT* and *FM*. For small to medium large time series, our approach slightly outperforms the three dimensionality reduction methods. In the next experiments, we demonstrate that, for some applications, our approach is also more effective for data mining tasks than Euclidean distance-based similarity measures.

13.4 Evaluation of the Threshold Based Similarity Measure

In this section, we experimentally evaluate the effectiveness of threshold queries. In particular, we prove the suitability of our similarity model against other approaches by using threshold queries for classification tasks performed on well established test datasets. The quality of our similarity model is expressed by the classification accuracy using a k-nearest-neighbor classifier (k = 5) with 10-fold cross validation. In order to achieve a large variety of different data characteristics in our test bed, we apply the *STANDARD* test datasets for the following experiments.

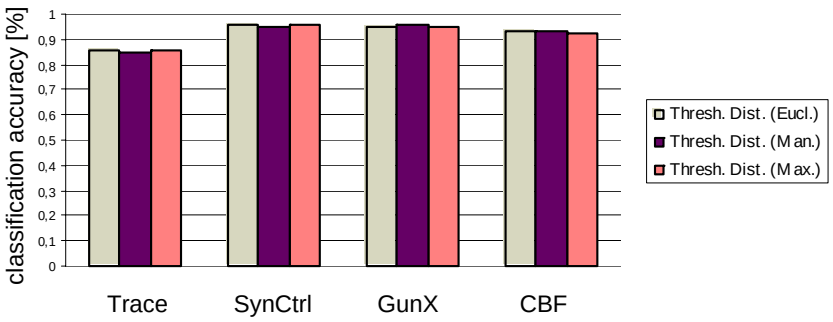


Figure 13.5: Comparison of Different Interval Similarity Distances.

13.4.1 Comparison to Traditional Distance Measures

In the first experiment (cf. Figure 13.4), we compare our approach to competing similarity measures traditionally used for time series data, the Euclidean distance (*Euclid. Dist.*), Dynamic Time Warping (DTW) and Derivative Dynamic Time Warping ($DDTW$) [KP01]. Our approach achieves a good classification quality for all four datasets. For the dataset *Trace* the Euclidean distance achieves only an accuracy of about 45% while our approach achieves about 86%. With the *GunX* dataset our approach even outperforms the DTW distance measure.

13.4.2 Comparison of Different Similarity Distances for Time Intervals

First, we examine different L_p-norms ($p = 1, 2, \infty$) applied to the interval-similarity distance measure d_{int}. Figure 13.5 depicts the results of the classification accuracy achieved, respectively. All three L_p-norms show similar behavior w.r.t. the classification accuracy.

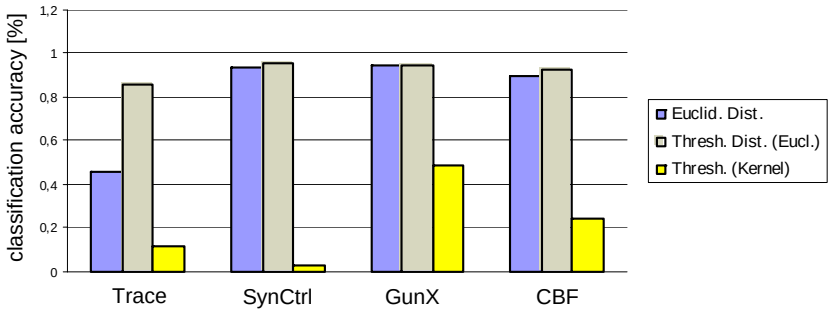


Figure 13.6: Comparison of Different Set Similarity Distances.

13.4.3 Comparison of Different Similarity Distances for Sets of Time Intervals

In the next experiment, we evaluate the effectiveness of the Sum of Minimum Distance (SMD) for measuring the threshold similarity between two time series. For comparison, we used the set kernel function as proposed in [GFKS02]:

$$d^{set-kernel}(S_X, S_Y) := \frac{\sum_{x \in S_X, y \in S_Y} e^{\frac{-d_{int}(x,y)^2}{\sigma}}}{|S_X| \cdot |S_Y|},$$

where σ is a parameter which can be used to adjust the sensitivity of the similarity. For low σ values, large interval distances have only little influence on the similarity. Figure 13.6 shows the results of the achieved classification accuracy for all four datasets. Additionally, we again depict the results of the simple Euclidean distance (*Euclid. Dist.*). The set-kernel distance measure (*Thresh. (kernel)*) falls by far below our proposed SMD-based distance measure (*Thresh. Dist. (Eucl.)*) for all four datasets. The problem of the set-kernel distance measure is that one time interval of one time series is matched to all time intervals of the other time series.

13.4.4 Results on Scientific Datasets

Now we evaluate the results on the air pollution dataset *SCIEN_ENV*. We performed 10-nearest neighbor threshold queries with randomly chosen query objects. Interestingly, when we choose time series as query objects that were

derived from rural sensor stations representing particulate matter parameters (PM_{10}), we obtained only time series representing the same parameters that were also measured at rural stations. This confirms that the pollution by particle components in the cities in fact differs considerably from the pollution in rural regions. A second interesting result was produced when we used PM_{10} time series of working days as queries. The resulting time series were also derived from working days representing PM_{10} values.

The results on the gene expression dataset were also very interesting. Our task was to find the most similar gene with $\tau = 0$ to a given query gene. The idea behind this task is to find a gene that is functionally related to the query gene. We posed several randomized queries to this dataset with $\tau = 0$ and evaluated the results w.r.t. biological significance, using the SGD database[1]. 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 participate in the protein biosynthesis. In particular, threshold queries can be used to predict the function of genes whose biological role is not resolved yet.

To summarize, the results on the real-world datasets suggest the practical relevance of threshold queries for important real-world applications.

[1] http://www.yeastgenome.org/

Part IV

Analysis using Reverse Nearest Neighbor Queries

Chapter 14

Introduction

Although the reverse k-nearest neighbor problem is the complement of the k-nearest neighbor problem, the relationship between kNN and RkNN is not symmetric and the number of the reverse k-nearest neighbors of a query object is not known in advance. A naive solution of the RkNN problem requires $O(n^2)$ time, as the k-nearest neighbors of all of the n objects in the dataset have to be found. Thus, more efficient algorithms are required.

As we discussed in Section 2.4, the well-known methods for reverse k-nearest neighbor search can be categorized into two classes, the hypersphere-approaches and the Voronoi-approaches. All these approaches are only designed for Euclidean vector data but cannot be applied to general metric objects. Hypersphere-approaches such as [YL01] extend a multidimensional index structure to store each object along with its nearest neighbor distance and, thus, actually store hyperspheres rather than points. In contrast, the Voronoi-approaches such as [TPL04] store the objects in conventional multi-dimensional index structures without any extension and compute a Voronoi cell during query processing. In principle, the possible performance gain of the search operation is much higher in the hypersphere approaches while only Voronoi-approaches can be extended to the reverse k-nearest neighbor problem with an arbitrary $k > 1$ in a straightforward way. However, this approach is not extendable to general metric spaces since it relies on explicitly computing Voronoi hyperplanes which are complex to compute in arbitrary

metric spaces.

In this part, we propose an efficient solution based on the hypersphere approach for the RkNN problem with an arbitrary k not exceeding a given threshold parameter k_{max} for general metric objects. The idea is not to store the true nearest neighbor distances for each k of every object separately but rather to use suitable approximations of the set of nearest neighbor distances. This way, we approximate both, the kNN distances of a single object stored in the database as well as the k-nearest neighbor distances of the set of all objects stored in a given subtree of our metric index structure. To ensure the completeness of our result set (i.e. to guarantee no false dismissals) we need a conservative approximation which never under-estimates any k-nearest neighbor distance but also approximates the true k-nearest neighbor distances of a single object or a set of objects with minimal approximation error (in a least squares sense). To reduce the number of candidates that need to be refined, we additionally store a progressive approximation of the kNN distances which is always lower or equal to the real kNN distances. Thus, only objects that have a kNN distance to a given query object which is between the lower bound (progressive approximation) and the upper bound (conservative approximation) need to be refined. We demonstrate in Chapter 15 that the k-nearest neighbor distances follow a power law which can be exploited to efficiently determine such approximations. Our solution requires a negligible storage overhead of only two additional floating point values per approximated object. The resulting index structure called MRkNNCoP (Metric reverse kNN with conservative and progressive approximations)-Tree can be based on any hierarchically organized, tree-like index structure for metric spaces. In addition, it can also be used for Euclidean data by using a hierarchically organized, tree-like index structure for Euclidean data.

14.1 Contributions

Most recent methods for the reverse k-nearest neighbor search suffer from the fact that they are only applicable to $k = 1$ or at least a fixed value of

k. The most generic approach is that of using Voronoi cells [TPL04]. Since it does not rely on precomputed kNN distances, it can handle queries with arbitrary values for k. However, the approach proposed in [TPL04] relies on the computation of the Voronoi (hyper-)plane which only exists in Euclidean vector spaces. In general metric spaces, the hyperplanes that separate the Voronoi cells are hard to compute. So far, there exist only methods for R-tree like index structures. Thus, these approaches cannot be extended for metric databases.

To the best of our knowledge, this approach is the first contribution to solve the generalized RkNN search problem for arbitrary metric objects. In particular, our method provides the following new features:

1. It can be applied to general metric objects, i.e. databases containing any type of complex objects as long as a metric distance function is defined on these objects.
2. It is applicable to the generalized RkNN problem where the value of k is specified at query time.

The remainder of this part is organized as follows: Section 14.2 introduces preliminary definitions. In Chapter 15 we introduce our new index structure, the MRkNNCoP-Tree, for efficient reverse k-nearest neighbor search in general metric spaces in detail. Chapter 17 contains an extensive experimental evaluation and concludes this part of the thesis.

14.2 Problem Definition

Since we focus on the traditional reverse k-nearest neighbor problem, we do not consider recent approaches for related or specialized reverse nearest neighbor tasks such as the bichromatic case, mobile objects, etc.

In the following, we assume that $\mathcal{D}$ is a database of n metric objects, $k \leq n$, and *dist* is a metric distance function on the objects in $\mathcal{D}$. The set of *k-nearest neighbors* of an object q is the smallest set $NN_k(q) \subseteq \mathcal{D}$ that

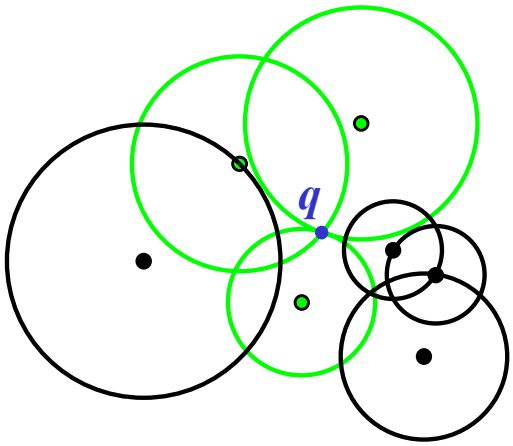


Figure 14.1: Example for a RkNN query with $k = 1$.

contains at least k objects from $\mathcal{D}$ such that

$$\forall o \in NN_k(q), \forall \hat{o} \in \mathcal{D} - NN_k(q) : dist(q, o) < dist(q, \hat{o}).$$

The object $p \in NN_k(q)$ with the highest distance to q is called the *k-nearest neighbor* (kNN) of q. The distance $dist(q, p)$ is called *k-nearest neighbor* distance (kNN distance) of q, denoted by $nndist_k(q)$.

The set of *reverse k-nearest neighbors* (RkNN) of an object q is then defined as

$$RNN_k(q) = \{p \in \mathcal{D} \mid q \in NN_k(p)\}.$$

In Figure 14.1, an example RkNN query for query object q and $k = 1$ is depicted. The result objects and their 1NN radii are shown in light color.

The naive solution to compute the reverse k-nearest neighbor of a query object q is rather expensive. For each object $p \in \mathcal{D}$, the k-nearest neighbors of p are computed. If the k-nearest neighbor list of p contains the query object q, i.e. $q \in NN_k(p)$, object p is a reverse k-nearest neighbor of q. The runtime complexity of one query is $O(n^2)$. It can be reduced to an average of $O(n \log n)$ if an index such as the M-tree [CPZ97] (or, if the objects are feature vectors, the R-tree [Gut84] or the R*-tree [BKSS90]) is used to speed-up the nearest neighbor queries.

Chapter 15

kNN Distance Approximations for RkNN Search

As discussed above, the Voronoi approach is not applicable to general metric spaces because in these spaces, the explicit computation of hyperplanes is not possible. As a consequence, we want to base our generic index for reverse kNN search on the ideas of the RdNN-Tree which is only applicable for Euclidean vector data and the specialized case of $k = 1$. The generalizations to enable the application of metric objects is rather straightforward. Instead of using an Euclidean index structure such as the R-Tree [Gut84] or R*-tree [BKSS90] we can use a metric index structure such as the M-Tree [CPZ97]. However, in order to solve the generalized RkNN problem where the value for k could be different from query to query is much more challenging. The key idea of the RdNN-Tree is to precompute the kNN distance for an *a priori* fixed value[1] of k for each object. If a query object q has a larger distance to an object o than the kNN distance of o, we can safely drop o. Otherwise, object o is a true hit. However, as indicated above, this approach requires to specify the value of k in advance.

In general, there are two possibilities to generalize the RdNN-Tree in order to become independent of k. First, we could store the kNN distances

[1] originally for $k = 1$

of each object for each $k \leq k_{max}$, where k_{max} is the maximum value of k depending on the application. Obviously, this approach results in high storage costs and, as a consequence, in a very large directory of the index tree. Thus, the response times will be significantly enlarged. Second, we can conservatively approximate the kNN distances of each object by one distance, i.e. the k_{max}NN distances. However, if k_{max} is considerably higher than the k needed in a given query, the pruning power of the k_{max}NN distance in order to identify true drops will obviously be decreased dramatically. As a consequence, the response time will also be significantly increased due to this bad kNN distance estimation and the resulting poor pruning. In addition, for each object o where $dist(o,q) \leq dist_{k_{max}}(o)$ a refinement is needed since it cannot be ensured that $dist(o,q) < dist_k(o)$, i.e. for each of these objects o another kNN query needs to be launched.

Here, we propose a novel index structure that overcomes these problems by approximating the kNN distances of each object for all $k \leq k_{max}$ using a function. We store for each object $o \in \mathcal{D}$ such a function that conservatively approximates the kNN distances of o for any $k \leq k_{max}$. The approximation is always greater or equal to the real distances. This conservative approximation allows to identify objects that can be safely dropped because they cannot be true hits. For the remaining objects, we need a refinement step inducing a kNN query for each candidate.

In fact, we can even further reduce the number of candidates, i.e. the number of necessary kNN queries for refinement by storing also a progressive approximation of the kNN distances of each $o \in \mathcal{D}$ for any $k \leq k_{max}$. The progressive approximation is always lower or equal to the real distances and, thus, enables to identify true hits. Only objects that have a distance to the query q less or equal than the conservative approximation and greater than the progressive approximation need to be refined, i.e. induce a kNN query.

The idea of using conservative and progressive approximations is illustrated in Figure 15.1. If q_1 is the query object, p is a true hit and need not to be refined because the progressive approximation ensures that $dist(p,q_1) < nndist_k(p)$. If q_3 is the query object, p can be dropped safely because the

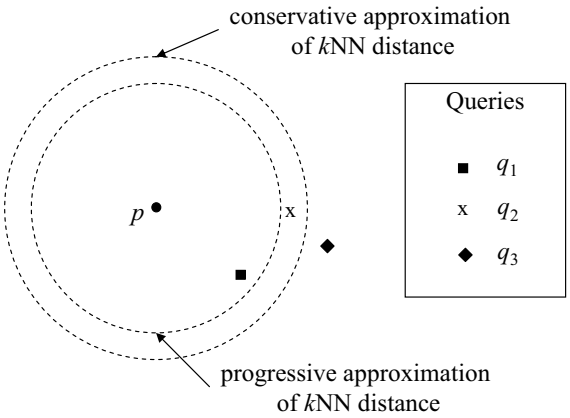


Figure 15.1: Using conservative and progressive approximation for RkNN search.

progressive approximation ensures that $dist(p, q_3) > nndist_k(p)$. If q_2 is the query object, p is a candidate that needs refinement, i.e. we must launch an exact kNN query around p.

Thus, for each object, instead of the kNN distance(s) of a given value of k or all possible values of k, we simply store two approximation functions. We can use an extended M-Tree, that aggregates for each node the maximum of all conservative approximations and the minimum of all progressive approximations of all child nodes or data objects contained in that node. These approximations are again represented as functions. At runtime, we can estimate the maximum kNN distance for each node using the approximation in order to prune nodes analogously to the way we can prune objects. The resulting candidate objects can be identified as true hits or candidates that need further pruning using the progressive approximation.

In the following, we explain how to compute a conservative approximation of the kNN distances for arbitrary $k \leq k_{max}$ (cf. Section 15.1). We then sketch how a progressive approximation can be generated analogously (cf. Section 15.6). After that, we describe how these approximations can be integrated into an M-Tree. The resulting structure is called MRkNNCoP-Tree (Metric RkNN with conservative and progressive approximations — cf. Section 15.7). At the end of this section, we outline our novel RkNN search algorithm (cf.

Section 16).

15.1 Conservative Approximation of k-NN Distances

As discussed above, a conservative approximation of the kNN distances of each data object is needed in order to determine those objects that can be safely dropped because they cannot be part of the final result.

First, we have to address the problem to select a suitable model function for the conservative approximation of our k-nearest neighbor distances for every $k \leq k_{max}$. In our case, the distances of the neighbors of an object o are given as a sequence

$$NNdist(o) = \langle nndist_1(o), nndist_2(o), ...nndist_{k_{max}}(o) \rangle$$

and this sequence is ordered by increasing k. Due to monotonicity, we also know that $i < j \Rightarrow nndist_i(o) \leq nndist_j(o)$. Our task here is to describe the discrete sequence of values by some function $appx_o : \mathbb{N} \to \mathbb{R}$ with $appx_o(k) \approx nndist_k(o)$. As the approximation function is required to be conservative we have the additional constraint $appx_o(k) \geq nndist_k(o)$.

From the theory of self-similarity [Sch91] it is well-known that in most datasets the relationship between the number of objects enclosed in an arbitrary hypersphere and the scaling factor (radius) of the hypersphere (the same is valid for other solids such as hypercubes) approximately follows a power law:

$$encl(\varepsilon) \propto \varepsilon^{d_f}$$

where ε is the scaling factor, $encl(\varepsilon)$ is the number of enclosed objects and d_f is the fractal dimension. The fractal dimension is often (but not here) assumed to be a constant which characterizes a given dataset. Our k-nearest neighbor sphere can be understood to be such a scaled hypersphere where the distance of the k-nearest neighbor is the scaling factor and k is the number

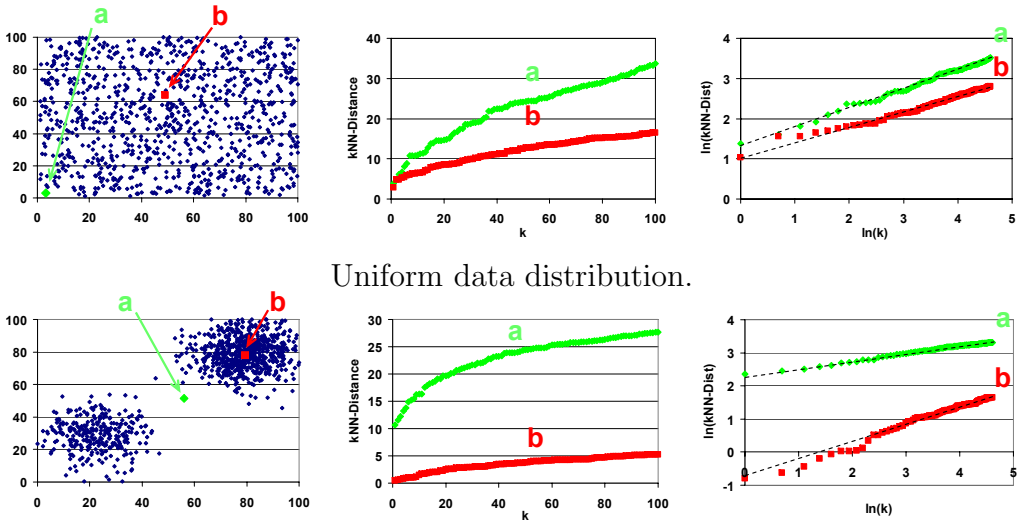


Figure 15.2: Illustration of the relationships between k and the kNN distance for different synthetic data distributions.

of enclosed objects. Thus, it can be assumed that the k-nearest neighbor distances also follow the power law and form approximately a line in log-log space (for an arbitrary logarithmic basis) [Sch91], i.e.:

$$\log(nndist_k(o)) \propto \frac{\log(k)}{d_f}.$$

This linear relationship between k and the kNN distance in log-log space is illustrated for different synthetic sample data distributions in Figure 15.2. Obviously this linear relationship is not perfect. However, as it can be anticipated from Figure 15.2, the relationship between $\log(k)$ and $\log(nndist_k(o))$ for any object o in a database of arbitrary distribution, exhibits a clear linear tendency.

From this observation, it follows that it is generally sensible to use a model function which is linear in log-log space — corresponding to a parabola in non-logarithmic space — for the approximation. Obviously, computing and storing a linear function needs considerable less overhead than a higher order function. Since we focus in this section on the approximation of the

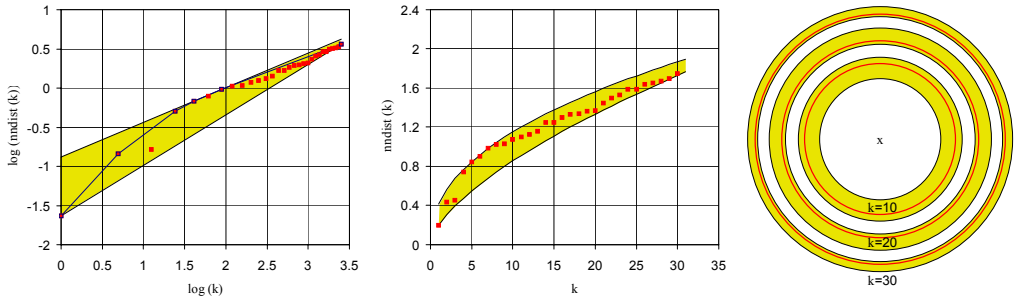


Figure 15.3: Visualizations of the conservative and progressive approximations of the k-nearest neighbor distances of a sample object for different values of k.

values of the k-nearest neighbor distance over varying k in a log-log sense, we consider the pairs $(\log(k), \log(nndist_k(o))$ as points of a two-dimensional vector space (x_k, y_k). These points are not to be confused with the objects stored in the database (e.g. the object o the nearest neighbors of which are considered here) which are general metric objects. Whenever we speak of *points* (x, y) or *lines* $((x_1, y_1), (x_2, y_2))$ we mean points in the two-dimensional log-log space where $\log(k)$ is plotted along the x-axis and $\log(nndist_k(o))$ for a given general metric object $o \in \mathcal{D}$ is plotted along the y-axis.

In most other applications of the theory of self-similarity it is necessary to determine a classical regression line without any constraint conditions, approximating the true values of $nndist_k(o)$ with least square error. A conventional regression line would find the parameters (m, t) of the linear function $y = m \cdot x + t$ minimizing least square error:

$$\sum_{1 \leq k \leq k_{max}} ((m \cdot x_k + t) - y_k)^2 = \min$$

which evaluates the well known formula of a regression line in 2D space. This line, however, is not a conservative approximation of a point set. In order to guarantee no false dismissals we need here a line which minimizes the above condition while observing the constraint that all actual y_k-values, i.e. $\log(nndist_k)$, are less or equal than the line, i.e. $y_k \leq m \cdot x_k + t$, and we derive a method with a linear time complexity in the number k_{max} of k-

nearest neighbor distances to be approximated (provided that the distances are ordered according to increasing k). We formally state this optimization problem:

Optimization Goal. The optimal conservative approximation of a sequence $NN_k(o)$ of k-nearest neighbor distances of an object o is a line

$$L_{opt}(o) = (m_{opt}, t_{opt}) : y = m_{opt} \cdot x + t_{opt}$$

in the log-log space. This line defines the following approximation function:

$$\log(appx_o(k)) = m_{opt} \cdot \log(k) + t_{opt}$$

with the following constraints:

C1. $L_{opt}(o)$ is a conservative approximation of y_k, i.e.

$$y_k \leq m_{opt} \cdot x_k + t_{opt} \ (\forall k : 1 \leq k \leq k_{max}).$$

C2. $L_{opt}(o)$ minimizes the mean square error, i.e.

$$\sum_{1 \leq k \leq k_{max}} (m_{opt} \cdot x_k + t_{opt} - y_k)^2 = min$$

An example is visualized in Figure 15.3. Here we have a sequence of k-nearest neighbor distances for $1 \leq k \leq k_{max} = 30$ which are depicted as squares in the left diagram in a log-log space and in the middle diagram in non-logarithmic space. The corresponding visualization of the conservative and progressive approximations in the data space for $k = 10, 20, 30$ are depicted on the right hand side in Figure 15.3. In the left diagram the optimal line for the conservative approximation is the upper limit of the shaded area. In Section 15.6, we also introduce the progressive approximation which is the lower limit of the shaded area. These two limiting lines correspond to the parabolic functions depicted in the middle diagram where we can directly determine the conservative and progressive approximations for a given k. On the right hand side in Figure 15.3, we have an object x together with its actual k-nearest neighbor distance for $k = \{10, 20, 30\}$. The three shaded shapes mark the conservative and progressive approximations for these three k-values.

We develop our method to determine the approximation function in three steps. At first, we show that the line must interpolate (pass through) either two neighboring points or one single point of the upper part of the convex hull of the set to be approximated. Second, we show how to optimize a regression line (m_A, t_A) under the constraint that it has to interpolate one given *anchor point* A (in our method, only points of the upper convex hull have to be considered as anchor points). As a third step, we show that the line obtained in the second step is the optimal line according to our optimization goal if and only if both the successor and the predecessor of A in the upper convex hull are under the line (m_A, t_A).

Moreover, we show the following: If the predecessor exceeds the line, then the optimum line cannot pass through one of the successors of A. *Vice versa*, if the successor exceeds line (m_A, t_A) the optimum line cannot pass through one of the predecessors of A. This statement gives us the justification to search for a suitable anchor point using a bisection search (in contrast to linear search the bisection search improves the total runtime but not the complexity of the overall method which will be shown to be linear in k_{max}). Finally, we show that the global optimum is either found directly by the bisection search or interpolates the last two points considered in the bisection search.

15.2 Optimization Step 1

First, we show that the line must interpolate either two neighboring points or one single point of the upper convex hull of the approximated point set. The upper convex hull (UCH) is a sequence

$$UCH = \langle (x_{k_1}, y_{k_1}), (x_{k_2}, y_{k_2}), ..., (x_{k_u}, y_{k_u}) \rangle$$

composed from u points ($2 \leq u \leq k_{max}$). UCH always starts with the first point and ends with the last point to be approximated, i.e. $k_1 = 1; k_u = k_{max}$. It contains the remaining points in ascending order, i.e. $k_i < k_j \Leftrightarrow i < j$, and due to monotonicity we know also $x_{k_i} < x_{k_j}$ and $y_{k_i} < y_{k_j}$. UCH forms a poly-line composed from one or more (exactly $u - 1$) line

segments $S_i = ((x_{k_i}, y_{k_i}), (x_{k_{i+1}}, y_{k_{i+1}}))$ where the most important property is that the slope $s(S_i) = (y_{k_{i+1}} - y_{k_i})/(x_{k_{i+1}} - x_{k_i})$ of the line segments S_i is strictly monotonically decreasing ($i < j \Leftrightarrow s(S_i) > s(S_j)$), i.e. the segments form a right turn. UCH is the maximal sequence with this property, and, therefore, the composed UCH line forms an upper limit of the points to be approximated. The upper convex hull has some nice properties which are important to solve our optimization problem. First, as we prove in Lemma 15.1, the optimal conservative approximation must interpolate one or two points of the convex hull. Second, and more importantly for the complexity of our method, it facilitates the test whether or not the constraint is fulfilled that all approximated points are below a given line. We will see later that only two points of UCH have to be tested. In contrast, if we do not know the UCH, all the points $(x_k, y_k), 1 \leq k \leq k_{max}$ of the complete approximated set need to be tested.

It is easy to see that an optimal line must interpolate at least one point of the approximated set:

Lemma 15.1.
The optimal line L_{opt} must interpolate at least one point of the set of conservatively approximated points.

Proof. *Assume L_{opt} is above all points of the dataset but does not touch one of the points of the dataset. Then we could move the line downwards (leaving it parallel to the original line) until it touches one of the points without violating the constraints. When moving downwards, the distance of the line to every point decreases. Therefore the original line cannot be optimal.* □

This is visualized in Figure 15.4a. In our next lemma, we show that it is not possible that the approximating line interpolates only points which are not in UCH. In other words, it cannot happen that none of the points that are interpolated by L_{opt} are not in UCH.

Lemma 15.2.
Any line L interpolating a point $(x_k, y_k), 1 \leq k \leq k_{max}$ which fulfills the

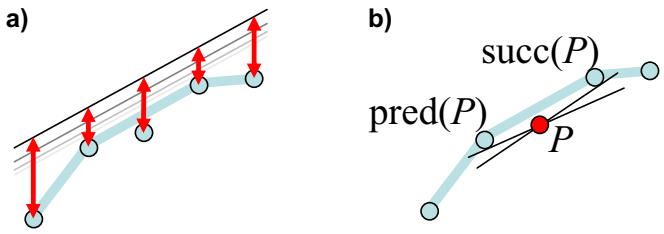


Figure 15.4: Constraints on the optimal line: Line intersects at least a) one point of the approximated set (Lemma 15.1) b) one point of the UCH (Lemma 15.2)

constraint that all approximated points are upper bound by it must interpolate at least one point $P \in UCH$.

Proof. *Let us assume, that L interpolates only one point $P \notin UCH$. Let $pred(P)$ be the last point before P which is member of UCH and $succ(P)$ be the first point after P in UCH. According to the definition of UCH, the two segments $S_1 = (pred(P), P)$ and $S_2 = (P, succ(P))$ form a left turn (otherwise, P would have to be member of UCH), i.e. $s(S_1) < s(S_2)$. The slope of L must be less than the slope of S_1 and greater than the slope of S_2 to upper bound both $pred(P)$ and $succ(P)$. This is not possible as the slope of S_1 is less than that of S_2 (left turn).* □

The idea of the proof is visualized in Figure 15.4b. From Lemma 15.2 and the obvious observation that no straight line can interpolate more than two points of UCH (which form a right turn), we know that L_{opt} must interpolate one or two points of UCH. We show later that both cases occur, indeed. In Figure 15.4, the 4 points of UCH are marked with darker frames and additionally connected by lines. This example also visualizes how the conservative approximation interpolates two points of UCH.

UCH can be determined from the ordered set of points $(x_k, y_k), 1 \leq k \leq k_{max}$ in linear time by sequentially putting the points onto a stack and deleting the second point after the top-of-stack whenever the slope is increasing among the three points on the top-of-stack. Obviously, the determination of UCH can be done in $O(k_{max})$.

15.3 Optimization Step 2

From Step 1 we know that our optimal line interpolates at least one point of UCH. Our next building block for the generation of the conservative approximation is the derivation of a regression line under the constraint that one given point is interpolated. We call this point the *anchor point*. Note that, until now, we have not yet shown how to select this anchor point among the points in UCH. For the moment, we can assume that we do this optimization for every point in UCH, but we show later how the anchor can be determined in a more directed way using bisection search. Furthermore, note that the optimization described in this step does not necessarily yield a line which fulfills all constraints. Our final method will do so, but the method here need not yet meet this requirement.

Given an anchor point $A = (x_A, y_A)$ and an approximated point set S, we call a line $L_A = (m_A, t_A)$ *anchor-optimal* w.r.t. A and S if it interpolates A and approximates S with least square error. As we know that our anchor point (x_A, y_A) is interpolated, in our optimization problem (to select the optimal line $y = m \cdot x + t$) we have only one variable (say m) as degree of freedom, because t is fixed by the constraint $y_A = m \cdot x_A + t$. Therefore, we can integrate the constraint into our line formula:

$$y = mx - mx_A + y_A$$

We search for that line which minimizes the sum of squared distances from the actual points to be approximated:

$$\sum_{1 \le k \le k_{max}} (mx_k - mx_A + y_A - y_k)^2 = min$$

An example is visualized in Figure 15.5 where we can see the anchor point as well as the distances from an arbitrary line which interpolates the anchor point. A necessary condition for a minimum is that the derivative vanishes:

$$\frac{\partial}{\partial m}\Big(\sum_{1 \le k \le k_{max}} (mx_k - mx_A + y_A - y_k)^2\Big) = 0$$

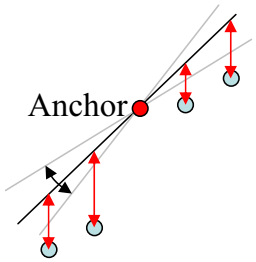


Figure 15.5: Illustration of an anchor point.

There exists only one solution $m = m_A$ to this equation with

$$m_A = \frac{\sum_k (y_k - y_A)(x_k - x_A)}{\sum_k (x_k - x_A)^2} = \\ = \frac{k_{max} x_A y_A - y_A(\sum_k x_k) + (\sum_k x_k y_k) - x_A(\sum_k y_k)}{k_{max} x_A^2 + (\sum_k x_k^2) - 2x_A(\sum_k x_k)}$$

This local optimum is a minimum, and there are no further local minima or maxima (except $m = \pm\infty$). Although the second formula for m_A looks complex at the first glance, it is very efficient to evaluate: the "expensive" sum-terms are independent from the anchor point A and need, thus, to be evaluated only once. If m_A is determined for more than one anchor point, those terms which need evaluation of all x_k and y_k must be evaluated only once. We show in the next section that, in general, we need to evaluate m_A for a number of anchor points which is logarithmic in the number of points in the convex hull, and, therefore, is in the worst case also in $O(\log k_{max})$.

15.4 Optimization Step 3

So far, we know that our optimal line interpolates at least one point of the UCH (Step 1). Together with Step 2, we know how to compute the optimal line: we need to construct an anchor optimal line given the right anchor point. We also know that this anchor point is part of the UCH. The remaining task is to find this right anchor point of the UCH. In the third step, we show that the correct anchor point and, thus, the global optimum approximation

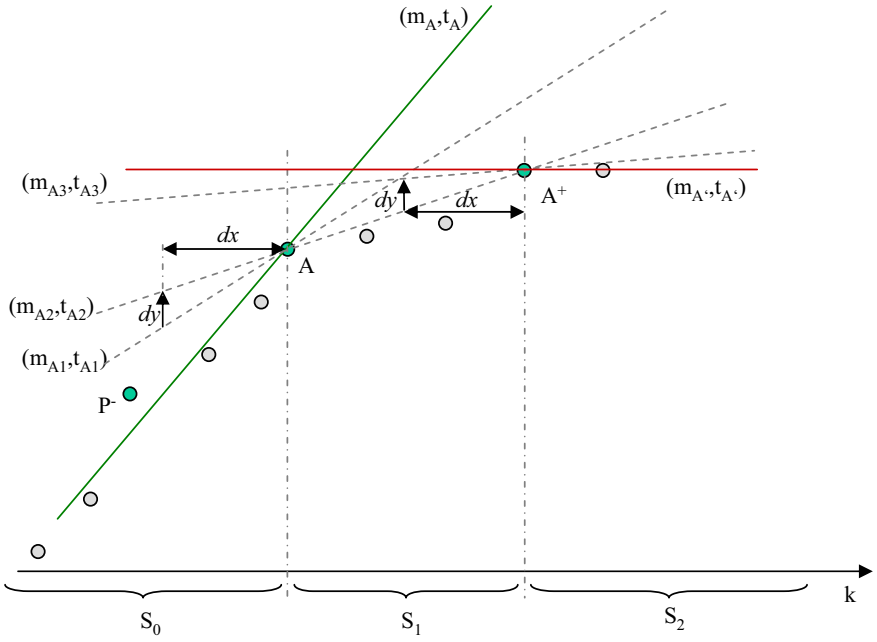


Figure 15.6: Illustration of the proof of Lemma 15.3: Monotonicity of the error of the conservative approximation of kNN distances

can be found efficiently by means of bisection search. The goal is to find that point $(x_{opt}, y_{opt}) \in UCH$ which is intersected by the global-optimum approximation-line. The search starts by the median $M_{UCH} = (x_{k_m}, y_{k_m})$ of the UCH which we first take as anchor point and compute its local optimum line (m_{k_m}, t_{k_m}). By means of the location of (m_{k_m}, t_{k_m}), we decide in which direction the bisection search has to proceed to find the correct anchor point. Thereby we distinguish three cases:

1. Both the predecessor and successor of M_{UCH} in the UCH, i.e. $pred(M_{UCH})$ and $succ(M_{UCH})$ are not above (m_{k_m}, t_{k_m}) (i.e. below or exactly on it).
2. The predecessor $pred(M_{UCH})$ is above (m_{k_m}, t_{k_m}).
3. The successor $succ(M_{UCH})$ is above (m_{k_m}, t_{k_m}).

As state above, the search starts at the median of the UCH, i.e. at M_{UCH}. At each step of the bisection search, we examine the three cases for

the current point $(x_{k_m}, y_{k_m}) \in UCH$ with its local optimum line (m_{k_m}, t_{k_m}). As we see in the next lemma, in the first case we can stop our search, because (m_{k_m}, t_{k_m}) is the global optimum. In the second case we proceed the search with the corresponding predecessor (x_{k_p}, y_{k_p}) of (x_{k_m}, y_{k_m}). If the corresponding predecessor has been already considered during the bisection search we can stop the search and the global optimum line passes through (x_{k_p}, y_{k_p}) and (x_{k_m}, y_{k_m}). In the third case we proceed the search with the successor of (x_{k_m}, y_{k_m}) analogous to the second case. In this way, the global optimum approximation can be found due to the following lemma:

Lemma 15.3.
Let $A \in UCH$ be the anchor point of an anchor optimal line (m_A, t_A), i.e. the line interpolate A and approximates the points with least square error. Furthermore, let (m_{A2}, t_{A2}) be another line which also passes through A and additionally passes through any successor $A^+ \in UCH$ of A, whereas $m_{A2} < m_A$. Then the global optimum line passes through A or any predecessors $A^- \in UCH$ of A.

Proof. *Let $A \in UCH$ be the anchor point of a line (m_A, t_A) which is anchor optimal w.r.t. A. Furthermore, let (m_{A2}, t_{A2}) be another line which pass through two points A and any successor $A^+ \in UCH$ of A, whereas $m_{A2} < m_A$. In the following we assume, that the global optimum line pass through A^+ but pass not through A. Let this line be $(m_{A'}, t_{A'})$, as depicted in Figure 15.6. Following the above assumption, the sum of squared distances from the actual points to line (m_{A2}, t_{A2}) must be greater than the squared distances to line $(m_{A'}, t_{A'})$. In the following, we show that this assumption does not hold.*

Let us first group all points into three sets S_0, S_1 and S_2. Whereas, S_0 denotes the set of all preceding points of A, S_1 denotes the set of all points succeeding A (A included) but preceding A_+ and S_2 denotes the set of all points succeeding A_+ (A_+ included). In the following we consider two additional lines (m_{A1}, t_{A1}) and (m_{A3}, t_{A3}), such that (m_{A1}, t_{A1}) pass through A and (m_{A3}, t_{A3}) pass through A_+. Furthermore, without loss of generality, let the slopes of (m_{A1}, t_{A1}) and (m_{A3}, t_{A3}) be in such a way, that $m_{A1} \leq m_A$

and $m_{A1} - m_{A2} = m_{A2} - m_{A3}$ *and* $m_A \geq m_{A1} \geq m_{A2} \geq m_{A3} \geq m_{A'}$ *(cf. Figure 15.6).*

Let I_i^A *be the sum of all squared distances from the points in* S_i *to the line* (m_A, t_A)*, i.e.*

$$I_i^A = \Sigma_{(x,y)\in S_i} dist((x,y),(m_A,t_A)),$$

where

$$dist((x,y),(m_A,t_A)) = (m_A x - m_A x_A + y_A - y)^2.$$

Then, the sum of the squared distances between all points and a line (m_A, t_A) *is equal to the sum* $I_0^A + I_1^A + I_2^A$*. If we assume that* (m_A, t_A) *is local optimal according to its anchor point* A *and* $m_A \geq m_{A1} \geq m_{A2}$*, then*

$$I_0^A + I_1^A + I_2^A \leq I_0^{A1} + I_1^{A1} + I_2^{A1} \leq I_0^{A2} + I_1^{A2} + I_2^{A2}.$$

The latter inequality is equivalent to:

$$I_0^{A2} - I_0^{A1} \geq I_1^{A1} - I_1^{A2} + I_2^{A1} - I_2^{A2}$$

By means of the theorems on intersecting lines and the assumption that $m_{A1} - m_{A2} = m_{A2} - m_{A3}$*, the following statements hold:*

1. $I_0^{A3} - I_0^{A2} \geq I_0^{A2} - I_0^{A1}$
2. $I_0^{A3} - I_0^{A2} \geq I_0^{A2} - I_0^{A1}$
3. $I_1^{A1} - I_1^{A2} \geq 0$
4. $I_1^{A3} - I_1^{A2} \geq 0$

With 1) the following statement holds:

$$I_0^{A3} - I_0^{A2} \geq I_1^{A1} - I_1^{A2} + I_2^{A1} - I_2^{A2}$$

With 2) this leads to

$$I_0^{A3} - I_0^{A2} \geq I_1^{A1} - I_1^{A2} + I_2^{A2} - I_2^{A3}$$

Due to 3) and 4), the inequality can be resolved to:

$$I_0^{A3} - I_0^{A2} + I_1^{A3} - I_1^{A2} \geq I_2^{A2} - I_2^{A3}$$

which is equivalent to

$$I_0^{A2} + I_1^{A2} + I_2^{A2} \leq I_0^{A3} + I_1^{A3} + I_2^{A3} \leq I_0^{A'} + I_1^{A'} + I_2^{A'}$$

This means, that the sum of squared distances from the actual points to line (m_{A2}, t_{A2}) *is lower than the squared distances to line* (m_{A3}, t_{A3}) *which contradicts the assumption.* □

15.5 Summary: The Optimization Algorithm

The algorithm which computes the optimal conservative approximation line is depicted in Figure 15.7. It requires as input an object o, the sequence $NNdist(o)$ of kNN distances of o which should be approximated and the upper limit k_{max}. The algorithm reports the line $L_{opt} = (m, t)$ corresponding to the line $y = m \cdot x + t$ which denotes the optimal conservative approximation line. The algorithm shown in Figure 15.7 consists of two main parts.

In the first part we compute the UCH of the kNN distances in $NBNdist(o)$ in the log-log space, i.e. of the points $(\log k, \log nndist_k(o))$. For this task, we apply a modification of Graham's scan algorithm for the convex hull [And79] which parses all kNN-distances from $k = 1$ to k_{max} and buffers references to those kNN-distances within a stack which build a conservative right-curved sequence denoting the upper convex hull of the points $(\log k, \log nndist_k(o))$ in log-log space. This step is performed because we have shown that at least one point of the convex hull must be interpolated by L_{opt} (cf. Lemma 15.2).

In the second main part we perform a bisection search for the optimum approximation line. Our algorithm starts with the complete UCH. It selects the median point of the UCH as the first anchor point and computes its anchor optimal line (aol). By inspecting its direct predecessor and successor, respectively, it distinguishes between 3 cases: (1) Both neighbor points are below aol: Then the global optimum is reached. (2) The right neighbor point (successor) is above the aol: We proceed recursively with the right half of the UCH (this time considering the median of the right half). Case (3),

ALGORITHM optimize(o, $NNdist(o)$, k_{max})
BEGIN
  // *First Part*
  compute upper convex hull UCH of the points $(\log k, \log nndist_k(o))$
    according to [And79];
  // *Second Part*
  WHILE UCH still contains unmarked points **DO**
    (x_a, y_a) = median of UCH;
    compute anchor optimal line (m_a, t_a) w.r.t. anchor point (x_a, y_a);
    mark (x_a, y_a);
    $(x_p, y_p) = pred((x_a, y_a))$; // *predecessor in* UCH
    $(x_s, y_s) = succ((x_a, y_a))$; // *successor in* UCH
    IF $y_p \leq m_a \cdot x_p + t_a$ **AND** $y_s \leq m_a \cdot x_s + t_a$ **THEN**
      // *global optimum found*
      RETURN (m_a, t_a);
    ELSE IF $y_p > m_a \cdot x_p + t_a$ **THEN**
      // *examine predecessor*
      IF (x_p, y_p) is already marked **THEN**
        $m_p = (y_a - y_p)/(x_a - x_p)$; $t_p = y_p - x_p \cdot m_p$;
        RETURN (m_p, t_p);
      ELSE // *proceed with left side of UCH*
        UCH = left side of UCH;
    ELSE IF $y_s > m_a \cdot x_s + t_a$ **THEN**
      // *examine successor*
      IF (x_s, y_s) is already marked **THEN**
        $m_s = (y_a - y_s)/(x_a - x_s)$; $t_s = y_s - x_s \cdot m_s$;
        RETURN (m_s, t_s);
      ELSE // *proceed with right side of UCH*
        UCH = right side of UCH;
  END WHILE
END

Figure 15.7: Finding the optimal approximation.

left neighbor above *aol* is handled analogously. The slope of the computed line is used to identify the search space of the subsequent search step (as substantiated by Lemma 15.3). In each step of this algorithm, the problem size is divided by two. Finally, the parameters (m, t) of the computed line are reported.

15.6 Progressive Approximation of kNN Distances

As discussed above, a progressive approximation of the kNN distances of each data object can be used to determine true hits. Analogously to the optimal conservative approximation, the optimal progressive approximation of a sequence $NN_k(o)$ of k-nearest neighbor distances of an object o is a line $L_{opt}(o)$ in log-log space. The only difference is that the line $L_{opt}(o)$ must satisfy a *progressive* constraint, i.e. constraint C1 in Section 15.1 is changed as follows:

$$y_k \geq m_{opt} \cdot x_k + t_{opt} \ (\forall 1 \leq k \leq k_{max}).$$

We can generate the progressive approximation analogously as described in Section 15.1. The difference is that we have to consider the lower convex hull instead of the upper convex hull. Obviously, the resulting progressive approximation line must be below all real kNN distances.

15.7 Aggregating the Approximations

So far, we have shown how to generate a conservative and progressive approximation for each object of the database. However, the conservative and progressive approximations can also be used for the nodes of the index to prune irrelevant sub-trees. Similar to the RdNN-Tree, we need to aggregate for each data node the maximum kNN distances of the objects within that node. For this aggregation, the conservative approximations must be used. In addition, we could aggregate the minimum kNN distance of the

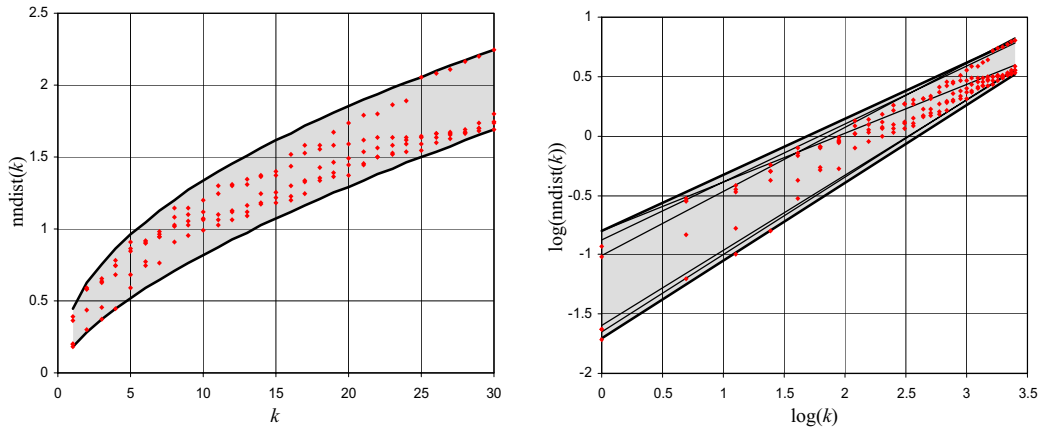


Figure 15.8: Aggregated approximation lines.

objects within the node in order to detect true hits. For this aggregation, the progressive approximation should be used. However, in most cases the progressive aggregation does not pay off because for a node it is not selective enough.

We build the conservative approximation of a data node N by conservatively approximating the conservative approximation lines $L_{opt}(o_i) = (m_i, t_i)$ of all data objects $o_i \in N$. The resulting approximation line is defined by the points $(\log(1), y_1)$ and $(\log k_{max}, y_{k_{max}})$, where y_1 is the maximum of all lines at $k = 1$ and $y_{k_{max}}$ is the maximum of all lines at k_{max}, formally

$$y_1 = \max_{o_i \in N} m_i \cdot \log 1 + t_i = \max_{o_i \in N} t_i,$$

$$y_{k_{max}} = \max_{o_i \in N} m_i \cdot \log k_{max} + t_i.$$

The progressive approximation can be determined analogously. Figure 15.8 illustrates both concepts.

The approximation information can be propagated to parent, i.e. directory, nodes in a straightforward way. The resulting storage overhead is negligible because the additional information stored at each node is limited to two values for the conservative approximation. Nevertheless, the kNN distance for *any* value of k can be estimated at each node.

We call the resulting index structure MRkNNCoP (Metric reverse kNN with conservative and progressive approximations) - Tree. The original concepts of the MRkNNCoP-Tree presented here can be incorporated within any hierarchically organized index for metric objects. Let us note that the concepts can obviously also be used for RkNN search in Euclidean data. In that case, we need to integrated the proposed approximations into Euclidean index structures such as the R-tree [Gut84], the R*-tree [BKSS90], or the X-tree [BKK96].

Chapter 16

RkNN Search Algorithm

The search algorithm for RkNN queries on our MRkNNCoP-Tree is similar to that of the RdNN-Tree. However, our index structure can answer RkNN queries for any k specified at query time and, due to the use of a metric index structure, is applicable to general metric objects.

The pseudo code of the query algorithm is depicted in Figure 16.1. A query q is processed by starting at the root of the index. The index is traversed such that the nodes having a smaller mindist to q than their aggregated kNN distance approximations are refined. Those nodes, where the mindist to q is larger than their aggregated kNN distance approximation are pruned. A node N of a M-Tree is represented by its routing object N_o and the covering radius N_r. All objects represented by N have a distance less than N_r to N_o. The mindist of a node N and a query point q, denoted by $mindist(N, q)$, is the maximum of the distance of q to N_o minus the covering radius N_r (if $dist(q, N_o) > N_r$) and zero (if $dist(q, N_o) \leq N_r$), formally:

$$mindist(N, q) = \max\{dist(q, N_o) - N_r, 0\}.$$

The aggregated kNN distance of a node N, denoted by $k\text{NN}_{agg}(N)$ can be determined from the conservative approximation $L_{opt}(N) = (m_N, t_N)$ of N by

$k\text{NN}_{agg}(N) = m_N \cdot \log k + t_N$.

Thus, we can prune a node N with approximation $L_{opt}(N) = (m_N, t_N)$ if

```
ALGORITHM RkNN_query(𝒟, q, k)
BEGIN    // Assumption: 𝒟 is organized as MRkNNCoP
  queue := new Queue;
  insert root of 𝒟 into queue;
  WHILE NOT queue.isEmpty()
    N := queue.getFirst();
    IF N is node THEN {
      IF mindist(N,q) ≤ m_N · log k + t_N THEN
        insert all elements of N into queue;}
    ELSE // N is a point
      IF dist(N,q) < m^p_N · log k + t^p_N THEN
        add N to result set;
      ELSE IF m^c_N · log k + t^c_N > dist(N,q)
        add N to candidate set;
  END WHILE
END
```

Figure 16.1: The RkNN search algorithm.

$mindist(N,q) > m_N \cdot \log k + t_N$.

The traversal ends up at a data node. Then, all points p_i inside this node are tested using their conservative approximation $L_{con}(p_i) = (m^c_{p_i}, t^c_{p_i})$ and their progressive approximation $L_{prog}(p_i) = (m^p_{p_i}, t^p_{p_i})$. A point p can be dropped if $dist(p,q) > m^c_p \cdot \log k + t^c_p$. Otherwise, if $dist(p,q) < m^p_p \cdot \log k + t^p_p$, point p can be added to the result. Last but not least, if $m^c_p \cdot \log k + t^c_p > dist(p,q) > m^p_p \cdot \log k + t^p_p$, point p is a candidate that need an exact kNN query as refinement. Using this strategy, we get a set of true hits and a set of candidates from the RkNNCoP-Tree. The set of candidates are refined using a batch kNN search as proposed in [YL01].

Chapter 17

Experimental Evaluation

All experiments have been performed on Windows workstations with a 32-bit 3.2 GHz CPU and 2 GB main memory. We used a disk with a transfer rate of 50 MB/s, a seek time of 6 ms and a latency delay of 2 ms. In each experiment we applied 100 randomly selected RkNN queries to the particular dataset and reported the overall result. The runtime is presented in terms of the elapsed query time including I/O and CPU-time. All evaluated methods have been implemented in Java. We applied our approach on metric as well as Euclidean datasets. The main characteristics of all datasets are summarized in Table 17.1.

17.1 Metric RkNN Search

Since there is no recent approach for RkNN search in general metric spaces, we compared our MRkNNCoP-Tree with two already discussed variants. The first variant, denoted by "MRkNN-Max", stores for each object of the database the kNN distance for one arbitrary $k_{max} \geq 1$. Obviously, this approach has less storage overhead than the MRkNNCoP-Tree but needs expensive refinement steps if the parameter k of the query differs from the precomputed k_{max} value. The second variant, denoted by "MRkNN-Tab", stores all kNN distances for $k = 1 \ldots, k_{max}$ in a table for each data object

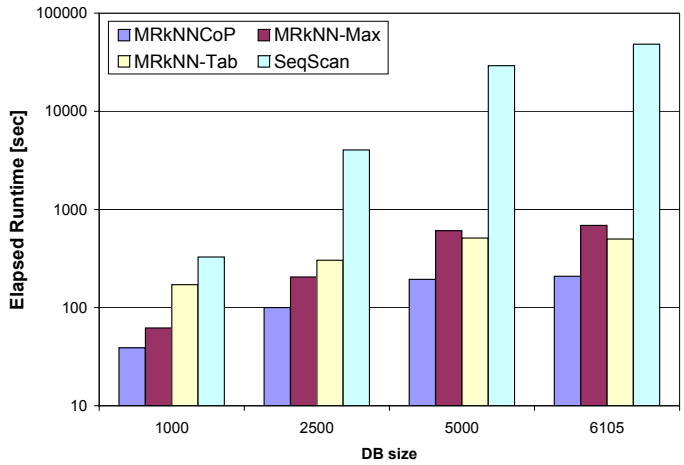


Figure 17.1: Runtime w.r.t. database size on Oldenburg dataset.

and each node of the tree, respectively. The advantage of this approach is that only true hits are computed, i.e. we do not need any refinement step. However, the distance table becomes quite large for increasing k_{max} values. This leads to a smaller branching factor of the tree nodes. Thus, the tree is higher suffering from large directory traversal overhead. A third competitor of our MRkNNCoP-Tree is the sequential scan, denoted as "SeqScan".

Our experiments were performed using a real-world road network data set derived from the city of *Oldenburg*. The nodes of the network graph were taken as database objects from which subsets of different size were selected to form the test dataset. For the distance computation we used the shortest-path distance computed by means of the Djikstra algorithm.

Table 17.1: Summary of the RkNN Test Datasets.

Dataset	Distance measure	dimensions
Oldenburg	Dijkstra	-
Sequoia	Euclidean	5
Color moments	Euclidean	9
Color texture	Euclidean	16

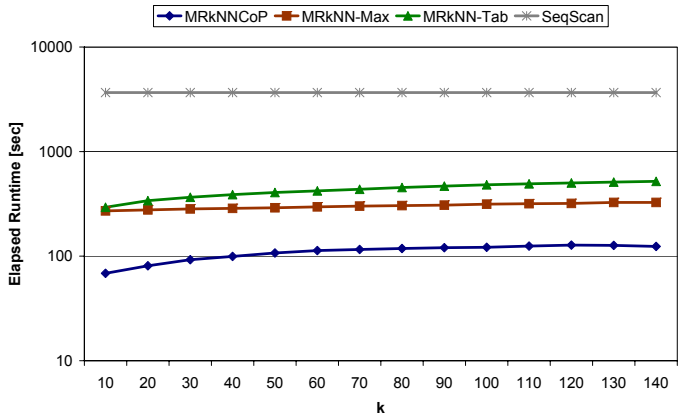


Figure 17.2: Runtime w.r.t. parameter k on Oldenburg dataset (2.500 data objects).

17.1.1 Runtime w.r.t. database size

In Figure 17.1 the runtime of the competitive algorithms w.r.t. varying data base size is illustrated in a log-scale. The parameter k was set to $k = 50$, while $k_{max} = 100$. It can be observed that our MRkNNCoP approach clearly outperforms the simple "MRkNN-Max" and "MRkNN-Tab" approaches. This is due to the already discussed shortcomings of the two naive approaches. "MRkNN-Max" suffers from a poor pruning capability, whereas "MRkNN-Tab" suffers from a large directory and, thus, from a costly tree traversal.

17.1.2 Runtime w.r.t. parameter k

A similar result can be observed when comparing the runtime of MRkNNCoP and its competitors w.r.t. varying parameter k (cf. Figure 17.2). In this experiment we set $k_{max} = 150$. Again, the runtime axis is in log-scale to visualize the sequential scan. Obviously, the sequential scan is almost independent from k and lies significantly above the runtime of the other techniques. Again, the MRkNNCoP-Tree performs significantly better than the naive approaches.

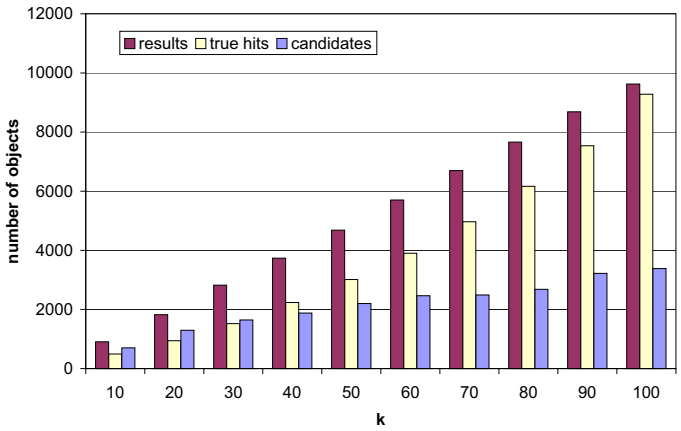


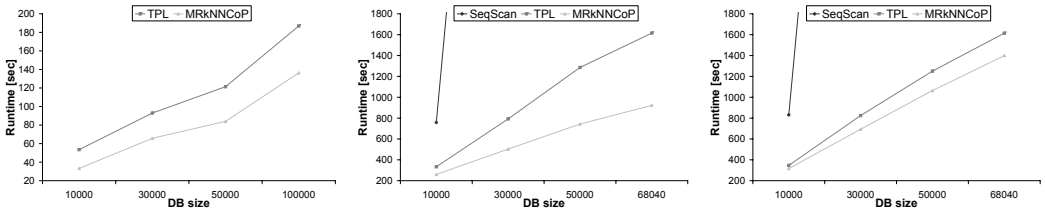
Figure 17.3: Pruning capability w.r.t. parameter k on Oldenburg data set (5.000 data objects).

17.1.3 Pruning capabilities

Figure 17.3 shows the pruning capability of MRkNNCoP w.r.t. k on the Oldenburg dataset. Compared with the size of the result set only a small number of candidates has to be refined, i.e. the conservative approximation yields a sound upper bound. Furthermore, the number of true hits we get from our progressive approximation increases with increasing k. For example, for $k \geq 70$ the pruned true hits are more than 75% of the result set. For these objects no expensive refinement step is necessary, thus our progressive approximation provides a very efficient lower bound for the kNN distance.

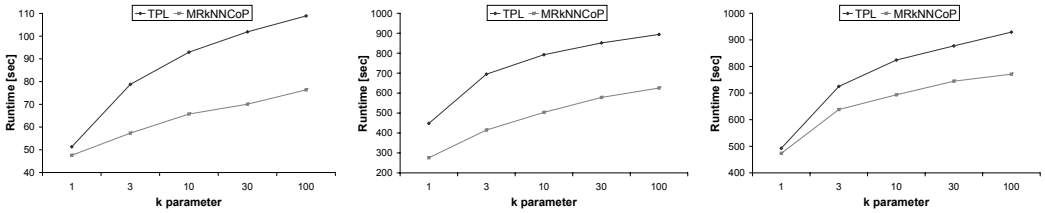
17.2 Euclidean RkNN Search

We also integrated our concepts into an X-tree [BKK96] in order to support RkNN search in Euclidean data. We made the same experiments as presented already above for metric data using three real-world datasets. It turned out that our approach even outperforms recent RkNN search methods that are specialized for Euclidean data. The used datasets include a set of 5-dimensional vectors generated from the well-known SEQUOIA 2000 bench-



(a) Sequoia (5-dimensional) (b) Color moments (9-dimensional) (c) Color texture (16-dimensional)

Figure 17.4: Comparison of runtime w.r.t. database size.



(a) Sequoia (5-dimensional) (b) Color moments (9-dimensional) (c) Color texture (16-dimensional)

Figure 17.5: Comparison of runtime w.r.t. parameter k.

mark data set and two "Corel Image Features" benchmark datasets from the UCI KDD Archive, one contains 9 values for each image, the other data set contains 16-dimensional texture values. The underlying X-tree had a node size of 2 KByte.

17.2.1 Naive approaches

Again, in our first experiment we compared our approach with the "MRkNN-Max" and the "MRkNN-Tab" approaches, both also based on an X-tree. The experiment was performed on the sequoia dataset consisting of 20.000 objects. The MRkNN-Max approach stores the k_{max}NN distance for $k_{max} = 100$. Again, it turns out that our approach has much better runtime performance than the other two techniques due to the already discussed shortcomings of these naive approaches. We observed similar results for the two other

datasets.

17.2.2 Runtime w.r.t. database size

We compared our approach with the Voronoi-based approach [TPL04] (short "TPL") and the sequential scan (short "SeqScan"). TPL is the only existing approach for the generalized RkNN search. In Figure 17.4 the runtime of the three algorithms w.r.t. varying data base size is illustrated. The parameter k was set to $k = 10$. For clearness reasons, the sequential scan is not visualized for all database sizes. It can be observed that our MRkNNCoP approach also outperforms the existing TPL approach. For example the performance gain is more than 40% for the 9-dimensional dataset. This is due to the fact that MRkNNCoP needs substantially less refinement steps than TPL.

17.2.3 Runtime w.r.t. parameter k

A comparison of the runtime of MRkNNCoP and TPL w.r.t. varying parameter k is depicted in Figure 17.5. In this experiment the database size was set to 30.000 objects. The runtime of the sequential scan is almost independent from k and lies significantly above the runtime of the two other techniques. Due to clearness reasons it is again not visualized. Again, the MRkNNCoP performs better than TPL. It is also worth noting that the performance gap between MRkNNCoP and TPL increases with increasing parameter k.

17.2.4 Pruning capabilities

Figure 17.6 shows the pruning capability of MRkNNCoP w.r.t. k on the 16-dimensional color texture dataset (30.000 objects). Compared with the size of the result set only a small number of candidates has to be refined, i.e. the conservative approximation yields a sound upper bound. Furthermore, the number of true hits we get from our progressive approximation is about 66% to 86% of the result set and increases with increasing k. For these objects no

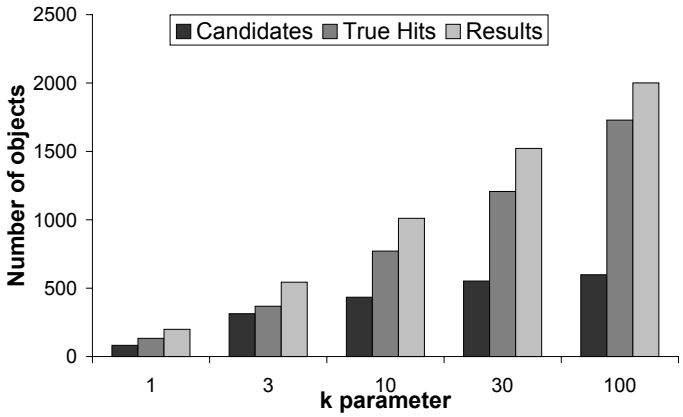


Figure 17.6: Pruning capability w.r.t. parameter k on color texture dataset.

expensive refinement step is necessary, thus our progressive approximation provides a very efficient lower bound for the kNN distance. We show only the color texture dataset, as the results on the two other datasets are similar.

Part V

Analysis of Video & Audio

Chapter 18

Introduction

Video and music clips are an important type of multimedia data. Due to recent technical advances, the amount of video and audio data that is available in digital formats as well as the possibility to access and display such files has increased enormously. Nowadays, it is possible to view complete movies on mobile phones and MP3 players. Another important aspect is that broadcasting streams over the Internet (e.g. in video podcasts or web radios) allows to distribute video and audio data to a large number of people while spending minimum effort and budget. Thus, new database techniques are needed which efficiently manage video and audio data.

Chapter 19

Video Retrieval

In the following, we propose a novel framework for video similarity search that takes the multi-represented nature of the data objects into account. In particular, our framework is able to integrate multiple representations such as audio and video features into the query processing. The most important issue for multi-represented similarity search is the weighting of each representation, i.e. the decision "how significant is a given representation for a given object". We propose methods for this task that can be applied to both types of summarization techniques, i.e. higher-order and first-order summarization, that are commonly used in multimedia similarity search. In addition, we propose a method for combining multiple representations for similarity search by weighting each representation. A broad experimental evaluation of our methods using a database of music videos demonstrates the benefit of our methods for similarity search in multimedia databases.

The remainder is organized as follows. Details of our novel method for multi-represented similarity search in multimedia databases are presented in Section 19.1, Section 19.4 we present our experimental evaluation.

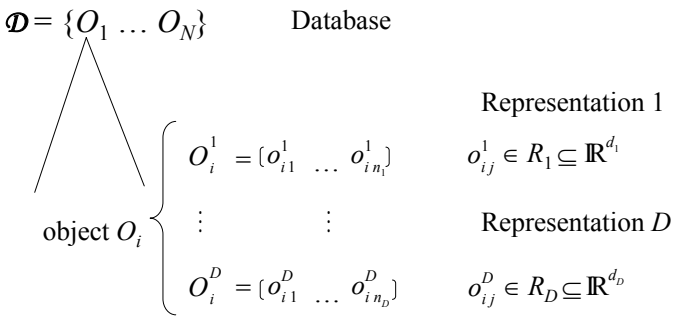


Figure 19.1: Basic notations.

19.1 Multi-represented Similarity Search in Multimedia Databases

From now on, we assume $\mathcal{D}$ to be a database of N multimedia objects. Each object $O_i \in \mathcal{D}$, $i = 1, \ldots, N$, is represented by a given set of D representations $R_1, \ldots, R_D$, where each representation is a feature space, i.e. $R_i \subseteq \mathbb{R}^{d_i}$, and $d_i \in \mathbb{N}$ denotes the dimensionality of the feature space of representation R_i $(1 \leq i \leq D)$. The j-th representation of O_i is denoted by O_i^j, i.e. $O_i = (O_i^1, \ldots, O_i^D)$. We further assume that each representation O_i^j of O_i consists of a series of feature vectors of length n_j, i.e. $O_i^j = (o_{i\,1}^j, \ldots, o_{i\,n_j}^j)$ with $o_{i\,l}^j \in R_i$. The definitions are summarized in Figure 19.1.

In addition, we assume that the distances within each representation are normalized sufficiently over all representations, e.g. using any of the methods of [SJL+03].

In order to combine multiple representations within the similarity evaluation, we have to determine for each object $O_i \in \mathcal{D}$ and for each of its representations O_i^j a weight for each of the n_j feature vectors $o_{i\,1}^j, \ldots, o_{i\,n_j}^j$.

Having weights for each feature vector of each representation of each object, we can use any common distance measure between sets of points such as the Hausdorff distance in order to compute a weighted distance between two multi-represented multimedia objects. We first introduce novel methods to determine the weights for a feature vector of a given representation and

then describe how these weights can be used to improve similarity search on multimedia objects.

19.2 Weighting Functions For Summarizations

As described above, a multimedia object usually consists of a large set of feature vectors per representation. For efficiency reasons, these large sets of feature vectors are usually summarized within each representation. The derived summarizations can be classified as first-order or higher-order summarizations (cf. Section 2.5). Thus, the feature vectors $o^j_{i\,1}, \ldots, o^j_{i\,n_i}$ of object $O_i \in \mathcal{D}$ of representation R_j are representative points of the derived summarizations $S^j_{i\,1}, \ldots, S^j_{i\,n_i}$. In the following, an original point p belongs to a summarization S if it is a member of the according cluster (in case of higher-order summarizations) or if the according representative of S is the representative with the lowest distance to p among the representatives of all summarizations.

Since different users may have a different notion of similarity among videos, it is desirable to consider this diversity in a best possible way when defining a similarity measure between multimedia objects. For our multi-represented approach, we have to take this diversity into account when we design a weighting function for the feature vectors of each representation. Thus, in the following, we present four methods to determine weights for representative feature vectors of a summarization that rates the significance of these summarization vectors in order to represent the according original feature vectors. The different weighting functions reflect different notions of similarity. Note that the weighting factor of each representative point is evaluated for each data object and each representation separately.

19.2.1 A Weighting Function Based on Support

The idea behind our first weighting function is that each summarization vector represents a given amount of original feature vectors. This amount

is a good indication on the significance of this representative, i.e. how good this summarization represents the original feature vectors. Thus, in our first approach, the weight of the l-th feature vector $o^j_{i\,l}$ of the j-th representation of object $O_i \in \mathcal{D}$, denoted by $\mathcal{W}_{\text{supp}}(o^j_{i\,l})$, is computed by the fraction of points that are represented by $o^j_{i\,l}$. Formally, if $|S^j_{i\,l}|$ denotes the fraction of original points that are summarized by $S^j_{i\,l}$, then the weight of the representative $o^j_{i\,l}$ is computed by

$$\mathcal{W}_{\text{supp}}(o^j_{i\,l}) = |S^j_{i\,l}|/n_j.$$

This weighting function is illustrated in Figure 19.2(a). The original points that contribute to the weight of the representative denoted by "△" are shaded in light gray, whereas the original points that contribute to the weight of the representative denoted by "x" are shaded in black. The weight for the representative denoted by "△" is simply computed by the fraction of gray points. The weight for the representative denoted by "x" is computed by the fraction of black points.

19.2.2 A Weighting Function Based on Specific Quality Measures

The first weighting function only considers the number of objects the given summarization vector represents. However, it does not take the distances to the representative object into account. For example, consider a representative point r_l representing l objects rather bad, i.e. the average distance of the l points to their representative r_l is significantly high, and a representative point r_k representing $k < l$ points significantly better, i.e. the average distance of the k points to their representative r_k is significantly low. Using our first weight function, r_l would be weighted higher than r_k (since $k < l$) although this contradicts the intuitive aim of our weighting function. A better idea might be to consider the distances of the original points within one summarization to their representative.

Usually, the summarization is generated optimizing a specific quality function. For example, for higher-order summarizations, the summariza-

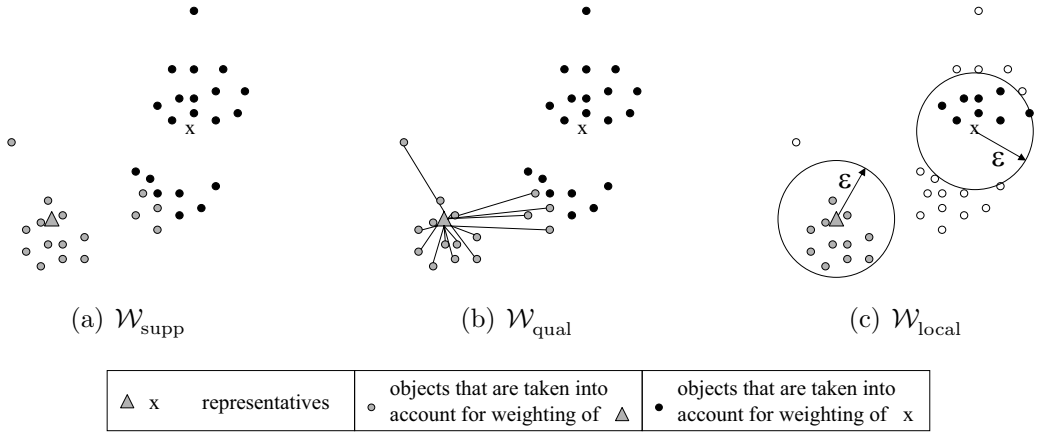


Figure 19.2: Illustration of three different weighting functions.

tion is derived from a clustering algorithm such as k-means or EM, which optimizes a clustering quality criterion (e.g. TD^2, log-likelihood). In case of first-order summarization techniques, we can e.g. use the method described in [CZ02] and the according quality function. Our second quality measure is based on the quality criterion upon which the summarization is generated. Intuitively, a summarization vector with high representative power should be weighted high.

Let $CQ(o^j_{i\,l})$ be the quality measure for the l-th summarization vector of the j-th representation of object $O_i \in \mathcal{D}$, based on which the summarization is generated, e.g. TD^2 in case of higher-order features generated by k-means. Then, the weight of $o^j_{i\,l}$ is computed by:

$$\mathcal{W}_{\text{qual}}(o^j_{i\,l}) = CQ(o^j_{i\,l}).$$

An example of this weighting function is visualized in Figure 19.2(b). The weight for the representative denoted by "△" is computed by e.g. the average distance of the original objects in its summarization to the representative.

19.2.3 A Weighting Function Based on Local Neighborhood

The second weighting function takes each original object into account when computing the weights for the derived summarizations. However, the original multimedia objects may contain some noise points, e.g. feature vectors that do not fit properly to any summarization, or — in case of an ineffective summarization procedure — one summarization may contain feature vectors of different clusters. In general, due to noisy original objects, the second weighting strategy may also fail.

In this cases, it would be more reliable to rate the weight of a representative point r based only on the original points in the local neighborhood of r.

Our third weighting function follows this idea. Let $\mathcal{N}_\varepsilon(r_i^j) = \{q_i^j | dist(r_i^j, q_i^j) \leq \varepsilon\}$ be the ε-neighborhood of a representative r_i^j of the i-th database object $O_i \in \mathcal{D}$ in the j-th representation R_j. Let us note that $\mathcal{N}_\varepsilon(r_i^j)$ only contains original feature vectors q_i^j of O_i in representation R_j. We define the weight of $o_{i\,l}^j$ by the number of objects in its local neighborhood, formally

$$\mathcal{W}_{\text{local}}(o_{i\,l}^j) = |\mathcal{N}_\varepsilon(o_{i\,l}^j)|/n_j.$$

This weighting function is illustrated in Figure 19.2(c). The original points that contribute to the weight of the representative denoted by "△" are again shaded in light gray, whereas the original points that contribute to the weight of the representative denoted by "x" are shaded in black. Original points that do not contribute to the weight of any summarization vector are shaded in white. The weights for both representatives are derived by the number of original points within their ε-neighborhood, normalized by n_j.

19.2.4 A Weighting Function Based on Entropy

The three weighting functions which we have introduced so far are rather local in the following sense: in order to compute the weight of a representative o

of a summarization S_o, they only consider the objects that are summarized by S_o, i.e. belong to S_o. However, it may be more appropriate to consider all original features of a given representation R_i in order to rate a summarization vector o^i of this representation. Our fourth weighting strategy follows this idea.

When computing the weight of a summarization vector o^i of a representation R_i, we want to take the distances of all original feature vectors $q_1^i, \ldots, q_m^i$ of representation R_i to o^i into account. In fact, the distances of q_l^i to o^i can be considered as a random variable x following a Gaussian distribution $G(x)$. The information content of such a random variable can be measured by its entropy. For example, if the entropy of the variable x equals 1, the distances $dist(q_l^i, o^i)$ are randomly distributed, whereas if the entropy of the variable x is considerably low, the distances $dist(q_l^i, o^i)$ are most likely densely packed around the mean value of x and thus, o^i is a good representation of the vectors $q_1^i, \ldots, q_m^i$. Figure 19.3 illustrates two Gaussians with different standard deviations derived from two summarizations of different quality. The Gaussian displayed in the upper part of Figure 19.3 has a lower deviation because the summarized original feature vectors are clustered. Its entropy will be considerably lower than the entropy of the Gaussians depicted in the lower part of Figure 19.3 which has a considerably higher standard deviation. This is due to the randomized distribution of the summarized feature vectors in the lower example.

Formally, let $x_{o^i} = \{dist(o^i, q_l^i) \mid 1 \leq l \leq m\}$ be a random variable. The Gaussian distribution $G(x_{o^i})$ of this random variable x_{o^i} is represented by the mean

$$\mu_{G(x_{o^i})} = \frac{\sum_{l=1}^{m} dist(o^i, q_l^i)}{m}$$

and the standard deviation

$$\sigma_{G(x_{o^i})} = \sqrt{\frac{1}{m} \cdot \sum_{j=1}^{m} (dist(o, q_j) - \mu_{G(x_{o^i})})^2}.$$

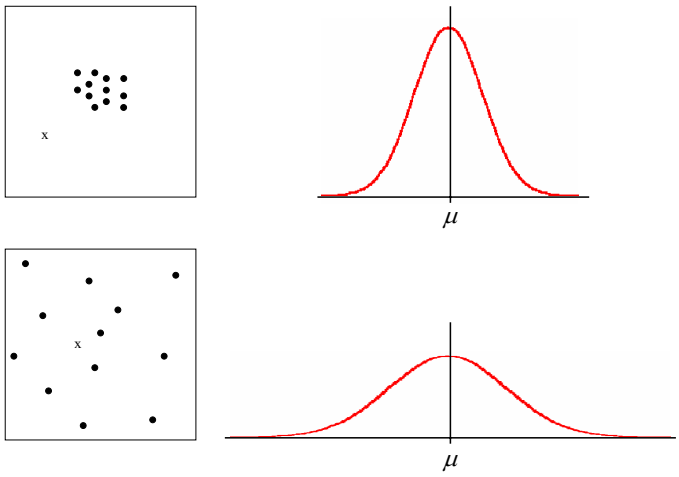


Figure 19.3: Different Gaussian distributions of distances from original objects to summarizations.

The entropy of x_{o^i} is then defined as

$$\mathcal{H}(x_{o^i}) = \int_{-\infty}^{+\infty} G(x_{o^i}) \cdot \log G(x_{o^i}) \; dx_{o^i}.$$

Let $o^j_{i\,l}$ be the l-th summarization vector of object $O_i \in \mathcal{D}$ in representation R_j and let $x_{o^j_{i\,l}}$ be the random variable built by the distances of the original features of O_i in representation R_j to $o^j_{i\,l}$ as defined above. The weight of $o^j_{i\,l}$ is defined as the entropy of the random variable $x_{o^j_{i\,l}}$, formally

$$\mathcal{W}_{\text{entropy}}(o^j_{i\,l}) = 1 - \mathcal{H}(x_{o^j_{i\,l}}).$$

The weighting function evaluates to zero if the entropy equals 1, i.e. the distances are distributed randomly. On the other hand, the weighting is near 1 if the distance distribution has a small standard deviation, i.e. the original feature vectors are considerably dense around the summarization vector.

Let us note, that we can efficiently calculate the entropy by using an appropriate five-order polynomial approximation that depends on the mean and standard deviation.

19.3 Combining Multiple Representations for Similarity Detection

Having defined a weighting function for each summarization vector for each representation of a database object, we can combine multiple representations for the process of similarity detection. The key step for efficient similarity search is the design of a dedicated distance measure that takes the weights of each summarization vector into account.

In general, we can adopt any distance measure that has been designed for multimedia objects to consider the weights of each feature vector. Let $O = (O^1, \ldots, O^D) \in \mathcal{D}$ be an arbitrary database object and let $Q = (Q^1, \ldots, Q^D)$ be the query object. Furthermore, let $dist^i$ be the distance function for comparing the i-th representation of O and Q, i.e. a^i and Q^i. Then, the distance between query object Q and a database object O can be computed by

$$dist(Q, O) = \sum_{i=1}^{D} \lambda^i \cdot dist^i(O^i, Q^i).$$

The most important part is to determine the weight λ^i of representation R_i. Obviously, λ^i should be derived from the weights of summarization vectors of the i-th representation of the query object Q, i.e. from $\mathcal{W}(q_1^i), \ldots, \mathcal{W}(q_{n_i}^i)$. The use of the weights of the query object Q only rather is more intuitive than using the weights of both Q and O because we want to ensure that we find database objects that are most similar to Q. Thus, the weights of Q are much more important than that of the database object O.

Regarding the distance function which should be used on the summarizations in each representation, we propose to distinguish between higher-order summarizations and first-order summarizations. Of course, we can combine representations of higher-order summarizations with representations of first-order representations.

19.3.1 Higher-order Summarizations

For higher-order summarizations, we use the Hausdorff distance which is an approved and frequently used distance measure in multimedia similaritry search to compute the similarity between a database object $O^i = \{o_1^i, \ldots, o_n^i\}$ and a query object $Q = \{q_1^i, \ldots, q_n^i\}$ w.r.t. a given representation R_i. Formally, the Hausdorff distance is defined as

$$H(Q^i, O^i) = \max(h(Q^i, O^i), h(O^i, Q^i)),$$

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} dist(a, b).$$

In fact, the Hausdorff distance relies on the distance of two specific summarizations, one from Q^i, say q_h^i, and one from O^i, say o_h^i. In other words, there are two summarizations $q_h^i \in Q^i$ and $o_h^i \in O^i$, such that $H(Q^i, O^i) = dist(q_h^i, o_h^i)$. Then the weight of the i-th representation λ^i is determined by the the weight of q_h^i, formally

$$\lambda^i = \mathcal{W}(q_h^i).$$

Let us note that the distance function $dist(a, b)$ between two summarization representatives a and b can be arbitrary. If the summarization representatives are feature vectors, e.g. derived by k-means clustering, any common distance measure such as the Euclidean distance can be used. If the summarization technique generates Gaussian distributions, e.g. using EM clustering, we use the Kullback-Leibler distance [IL00].

19.3.2 First-order Summarizations

For first-order summarizations, we use the distance function proposed in [CZ02] called ranked ViSig Similarity (VSS). This similarity measure relies on a set of distances between summarizations of the query Q and a database object O in each representation. Analogously to higher-order features, we weight each distance with the weight of the participating query summarization.

Table 19.1: Summary of the Video Test Dataset.

Dataset	# video clips	avg. duration (mins:secs)	# video features	# audio features
MUSICVIDEO	500	4:05	3	4

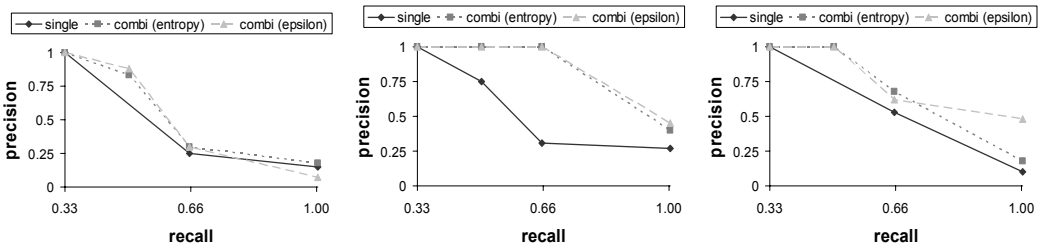
19.4 Experimental Evaluation

All experiments were performed on a workstation featuring a 1.8 GHz Opteron CPU and 8GB RAM. We evaluated our concepts using a database of music videos (*MUSICVIDEO*) collected by a focused web crawler. A short summary of the characteristics of the dataset is given in Table 19.1.

19.4.1 Multi-represented vs. Uni-represented Similarity Search

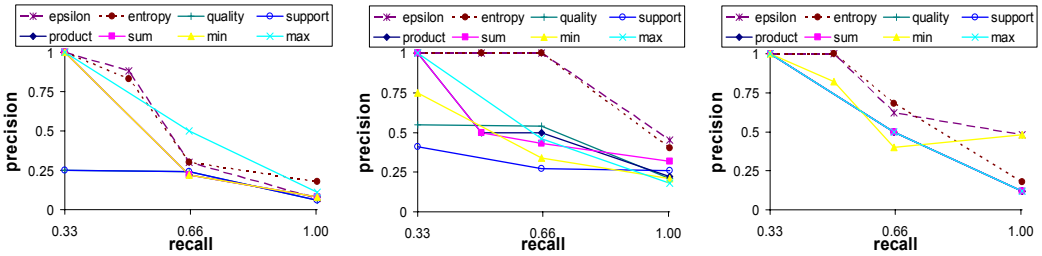
First, we show that multi-represented similarity search is usually more effective than similarity search using only one representation. In addition, we show in this subsection, that weighting the different representations yield a significant benefit compared to un-weighted multi-represented similarity detection.

In a first experiment, we performed video similarity search. As setup step, we picked 50 query videos from our database and manually selected a set of videos which are similar to the query videos. We compared recall and precision achieved on the best single representation to the query result computed by using the ε-neighborhood and entropy weighting functions. Furthermore, we investigated the performance of our weighting strategies on three summarization techniques, namely video signatures (ViSig), K-Means and expectation maximization (EM). The results of this comparison is depicted in Figure 19.4. For all evaluated summarization techniques, we observed a significant performance improvement when using multiple representations in comparison to the best single representation. Furthermore, our weighted approach leads to better results on all considered summarization techniques.



(a) ViSig (b) K-Means (c) EM

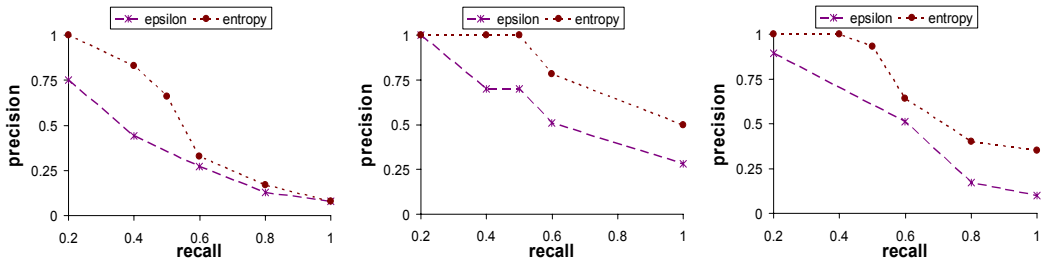
Figure 19.4: Precision vs recall for different summarization techniques on best single representation and two best weighting functions.



(a) ViSig (b) K-Means (c) EM

Figure 19.5: Precision vs recall for different summarization techniques on standard combination strategies and proposed weighted combination strategies.

Using the same test setup as described before, we compared different standard combination techniques for multi-represented objects to our weighted combination method is shown in Figure 19.5. We investigated the performance of commonly used standard combination techniques such as product, sum, minimum and maximum. In most cases, our weighted approach is more effective than the standard combination algorithms. Especially the ε-neighborhood and entropy weighting methods show good precision and recall values for all considered summarization strategies.



(a) ViSig (b) K-Means (c) EM

Figure 19.6: Precision vs recall for different weighting strategies when performing similarity search for videos of the same artist.

19.4.2 Multi-represented Similarity Search Applications

In the following, we identify two common applications that may pose different challenges to multimedia similarity search techniques and propose the most appropriate weighting functions for these tasks.

Application 1: Finding Similar Videos

Our first application addresses copyright issues. In order to detect plagiarism, we want to find videos that are similar to a given query video. We argue that in this application, similarity should be considered more locally because several representations are usually almost identical. This is the case if e.g. the image or audio part of a video is encoded in different resolutions or sampling rates. To distinguish these videos from the rest of the database, it is necessary to examine a small neighborhood. Otherwise, we would obtain results which are similar, but do not not violate the copyright.

The ε-neighborhood weighting function follows this idea and can successfully be applied for this task as shown in Figure 19.4 and Figure 19.5.

Application 2: Finding Videos of a Given Artist

In our second application, we address content-based multimedia retrieval in music video databases. Given a query video of a specific artist, we want all videos of this artist in our database. Obviously, in this application, a more global notion of similarity is necessary.

In order to demonstrate this idea, we selected a set of 20 query videos associated with different artists. For each video in our query set, we extracted all videos of the same artist from our database. The results of our artist search are depicted in Figure 19.6. In all experiments, the entropy-based weighting function outperforms the ε-neighborhood approach. This can be explained by the fact that the entropy weighting function takes all distances into account in opposite to the local character of the ε-neighborhood function.

Chapter 20

Audio Classification

The progress of computer hardware and software technology in recent years made it possible to manage large collections of digital music on an average desktop computer. Often meta information, such as artist, album or title, is available along with the audio file. However, the amount and quality of the available meta information in publicly accessible online databases, e.g. freedb.org, is often limited. This meta data is especially useful when searching for a specific piece of music in a large collection. To organize and structure a collection, additional information such as the genre would be very useful. Unfortunately, the genre information stored in online databases is often incorrect or does not meet the user's expectations.

In this chapter, a content-based hierarchical genre classification framework for digitized audio is presented as sketched in Figure 20.1. It is often problematic to assign a piece of music to exactly one class in a natural way. Genre assignment is a somewhat fuzzy concept and depends on the taste of the user. Therefore, our approach allows multi-assignment of one song to several classes. The classification is based on feature vectors obtained from three acoustic realms namely *timbre*, *rhythm* and *pitch*. Thus, each song is described by multiple representations, each of them containing a set of feature vectors, so called *multiple instances*.

Our main contributions are:

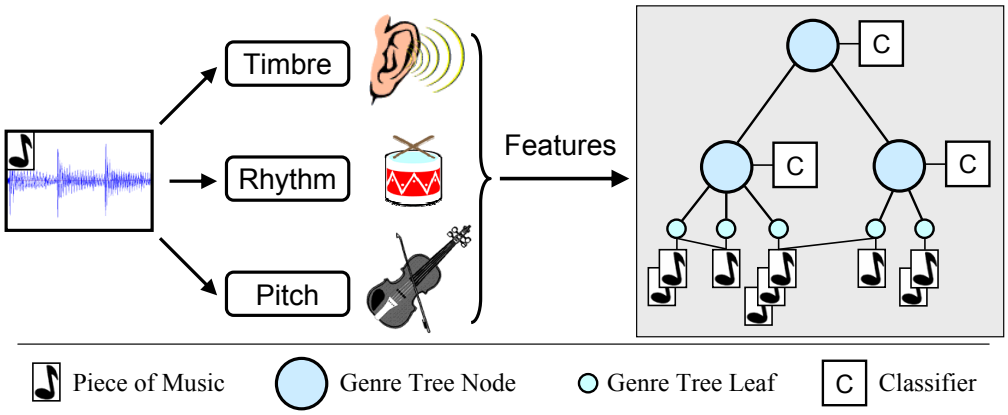


Figure 20.1: Architecture of the proposed framework.

1. a novel semi-supervised, hierarchical *instance reduction* technique which enables us to use only a small number of relevant features for each classifier.

2. An effective and efficient framework for *hierarchical genre classification* (HGC) of music pieces in a *multi-representation* and *multi-instance* setting.

Let us note that our framework can also be used for *genre classification* (GC) in flat class systems.

20.1 Efficient Hierarchical Genre Classification

In this section, we describe our approach for classifying large collections of music pieces in a genre taxonomy (cf. Figure 20.2). Since a music piece is described by a set of feature vectors, we first describe a novel hierarchical semi-supervised technique for instance reduction. The reduced descriptions are used afterwards for hierarchical classification of music pieces with SVMs. Furthermore, we use object adjusted weighting in order to take advantage from multiple representations.

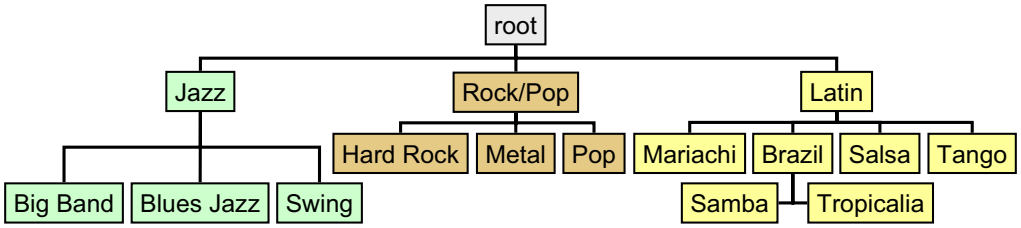


Figure 20.2: An example genre hierarchy.

20.1.1 Hierarchical Instance Reduction.

Let DB be a set of music objects. We argue that a multi-instance object $X = \{x_1, \ldots, x_n\} \in DB$ can be described by a vector $X_{reduced}$ containing minimal distances to a given set of so called *support objects* $S = \{s_1, \ldots, s_m\}$ where $m \ll n$. Formally,

$$X_{reduced} = (\min_{x_i \in X} dist(x_i, s_1), \ldots, \min_{x_i \in X} dist(x_i, s_m)).$$

The set S can either be calculated by a random selection of m instances from DB, or it is possible to choose each $s_i \in S$ as a centroid of a clustering that can be calculated on a small sample of instances from DB. An example for the instance reduction is illustrated in Figure 20.3.

The number of elements in $X_{reduced}$ may still be too large for solving the classification problem efficiently. Thus, we propose to exploit the hierarchical organization of classes and to select only a small subset $S_N \subseteq S$ for each inner node N of the genre taxonomy. The elements of S_N should be selected so that the subclasses C_N of N can be distinguished in the best possible way. Therefore, the subset of support objects is individual for each inner node N.

To calculate S_N we suggest to apply a semi-supervised method based on the *information gain criterion*. Let $T(C_N)$ be a set of all training objects belonging to C_N. The domains $D(s_i)$ are discretized by using the method described in [FI92]. After discretization the information gain criterion for each attribute can by calculated by

$$InfoGain(s_i, T(C_N)) = H(T(C_N)) - \sum_{t \in T(C_N)} \frac{|t|}{|T(C_N)|} \cdot H(t),$$

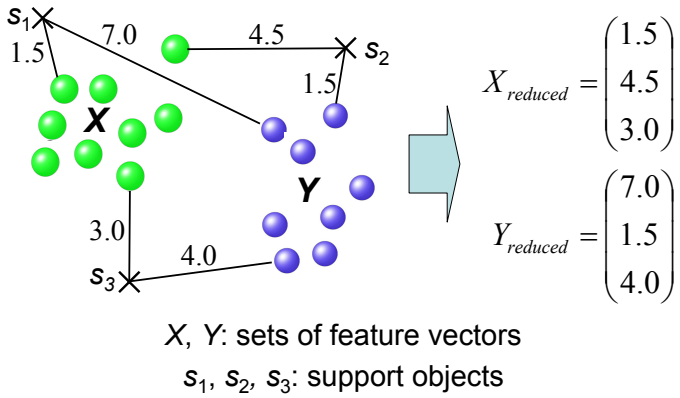


Figure 20.3: Instance reduction with help of support objects.

where $H(t)$ denotes the entropy. Finally, S_N is calculated as follows: $S_N = \{s_j \in S \,|\, |S_N| = k \wedge \forall s_j \in S_N \forall a \in S : InfoGain(a, T(C_N)) \leq InfoGain(s_j, T(C_N))\}$. After that, S_N is used for training and classification on the node N.

20.1.2 Hierarchical Genre Classification by Using Multiple Representations.

A *two layer classification process* (2LCP) handles the hierarchical classification problem on each inner node N of the genre taxonomy. This process acts as a guidepost for the hierarchical classification. We train SVMs in the first layer of the 2LCP that distinguishes only single classes C_{single} in each representation. Since standard SVMs are able to make only binary decisions we apply the so-called one-versus-one (OvO) approach (cf. Figure 20.4) in order to make a classification decision for more than two classes. We argue that for our application the OvO approach is best suitable because the voting vectors Φ_i provided by this method are a meaningful intermediate description that is useful for solving the multi-assignment problem in the second layer of our 2LCP. In order to perform the multi-assignment we take advantage of the class properties in our application domain. We limit the possible class combinations to a subset $C_{combi} \subset 2^{C_{single}}$ because there exist several combinations that do not make sense, e.g. a piece of music belonging to the class 'salsa'

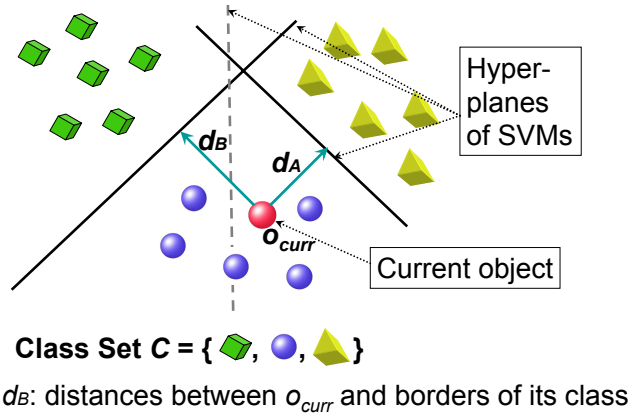


Figure 20.4: Border distance based derivation of weights for a multi-represented object.

is very implausible to be also in the class 'metal'. For this purpose, we only take those $c \in 2^{C_{single}}$ into account, which occur in the training set.

The SVM classifier in the second layer of the 2LPC uses an aggregation of the voting vectors Φ_i from the first layer of the 2LPC as input to assign an object to a class $c \in C_N = C_{single} \cup C_{combi}$. The second task that is handled by the classifier in the second layer is the aggregation of multiple representations. The voting vectors $\Phi_1, \ldots, \Phi_k$ provided by the first layer SVMs for each representation $R_1, \ldots, R_k \in R$ are aggregated by using a weighted linear combination $V = \sum_{i=1}^{k} \omega_i \Phi_i$. Then V is used as the input for the classifier in the second layer. The weights ω_i in the combination are calculated by using object adjusted weighting. The intuition behind the object adjusted weighting is that the current object o_{curr} used in training or to be classified needs to have a sufficient distance from any of the other classes. Furthermore, the closer surrounding of the hyperplane is treated in a more sensitive way. More formally, let c_j be the class of o_{curr} determined by majority vote in Φ_i, then $\omega_i = sigmoid(\min_{c_i \in C_{single} \wedge c_i \neq c_j} dist(o_{curr}, HyperPlane(c_j, c_i)))$, where $HyperPlane(c_j, c_i)$ denotes the maximum margin hyperplane separating the classes c_j and c_i, $sigmoid$ denotes sigmoid function defined as $sigmoid(x) = \frac{1}{1+exp(\alpha \times x + \beta)}$. A suitable optimization algorithm (e.g. Levenberg-Marquardt algorithm [Lev44]) is used to determine the parameters α and β that minimize the least squares error for the sigmoid target function $accuracy(o) =$

Table 20.1: Summary of the Audio Test Dataset.

Dataset	# music pieces	avg. duration (mins:secs)	# classes
SONGDATA	486	4:14	15

$\frac{1}{1+exp(\alpha \times x+\beta_j)}$ given the observed pairs of confidence ranges and classification accuracy. Figure 20.4 depicts an example of weight calculation where the weight ω should be set according to the value of d_A.

20.2 Experimental Evaluation

We implemented our approach in Java 1.5 and performed all experiments on a Pentium IV workstation equipped with 2 GB main memory. The genre hierarchy depicted in Figure 20.2 was used in all following experiments. A brief description of the *SONGDATA* dataset used for the evaluation can be found in Table 20.1.

We performed 10-fold cross-validation for evaluating the classification accuracy. In the following, we present the results of our experiments with particular emphasis to efficiency and effectiveness.

20.2.1 Effectiveness

In the first experiment, we compared the quality of GC on multiple, and HGC on single and multiple representations. Figure 20.5 depicts the experimental results. When working with multiple representations, our HGC approach (70.03%) achieves higher classification accuracy than using a single representation only. Furthermore, the classification accuracy of HGC is comparable to that of the flat GC approach (72.01%).

In the next experiment, we investigated how the classification accuracy of our approach is influenced by the number and the choice of the support

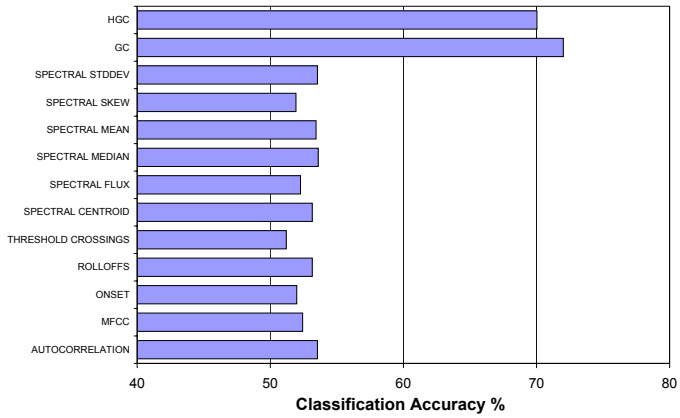


Figure 20.5: Accuracy for classification on single- and multi-representations.

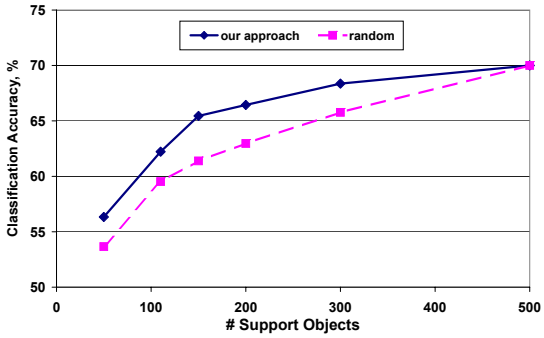


Figure 20.6: Accuracy for classification on single- and multi-representations.

objects. For choosing S_N, we either randomly picked the support objects or applied our strategy described in Section 20.1. The experimental results are depicted in Figure 20.6 and show that our approach always outperforms the random selection. For both approaches, the accuracy increases with an increasing number of support objects. However, especially for a low number of support objects, the random approach achieves a lower accuracy compared to our method. For a high number of support objects, both approaches yield a similar classification accuracy.

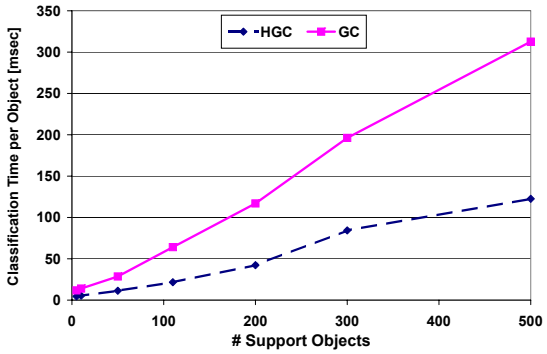


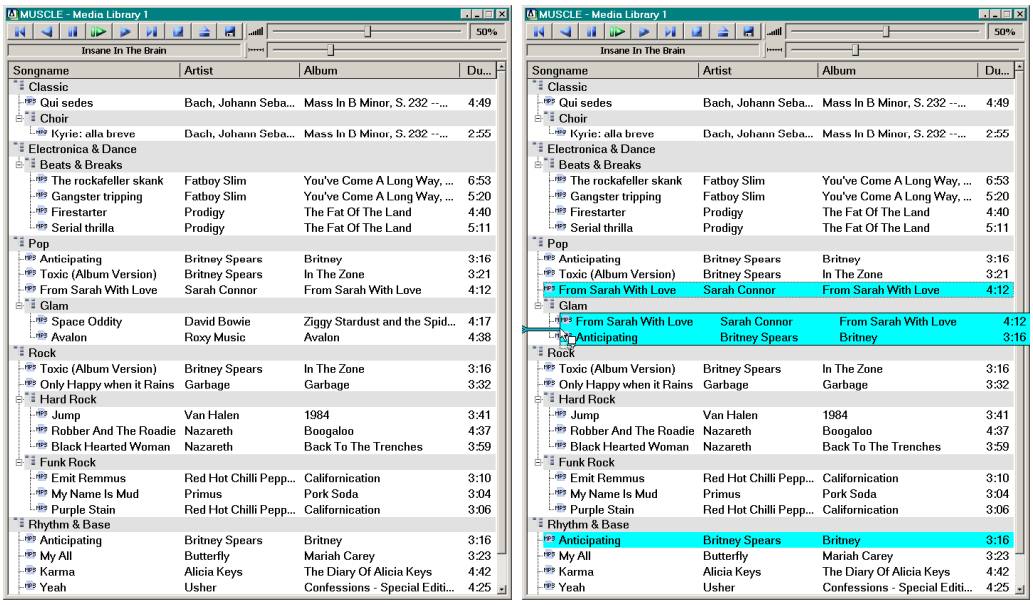
Figure 20.7: Classification time per object.

20.2.2 Efficiency

In a last experiment, we examined the runtime performance of GC and HCG for a varying number of support objects. As depicted in Figure 20.7, the runtime increases with an increasing number of support objects. The higher the number of support objects, the larger the runtime difference. Altogether, our approach achieves a good trade-off between the quality of the result and the required runtime when using 300 support objects.

20.3 Prototype

In the following, we present MUSCLE [BKK+06], a prototype of a powerful hierarchical genre classification tool for digitized audio. MUSCLE is based on the theoretical foundations introduced in this chapter and allows the user to organize large music collections in a genre taxonomy and to modify class assignments on the fly. It is often problematic to assign a piece of music to exactly one class in a natural way. Genre assignment is a somewhat fuzzy concept and depends on the taste of the user. Therefore, MUSCLE allows multi-assignments of one song to several classes. The classification is based on feature vectors obtained from three acoustic realms namely *timbre*, *rhythm* and *pitch*.



(a) Multi-Assignment of Songs (b) User Feedback

Figure 20.8: MUSCLE User Interface.

20.4 Practical Benefits

MUSCLE is implemented in C/C++ and runs on the Windows platform. Its hierarchical playlist acts as a jukebox. The installation archive of MUSCLE contains a default genre taxonomy including the necessary training data in the form of feature vectors for each song. This data is used in the demonstration. Using aggregated information such as feature vectors makes it possible to share the training data without having to distribute the underlying music data. Classes and training data in the genre taxonomy can be deleted, moved or added by the user. When the user commits the changes of the class hierarchy or of the corresponding training data, MUSCLE trains the affected classifiers. Note that usually only a small subset of the entire classifier hierarchy has to be trained because a modification at a node requires a partial adaptation of the node and all parent nodes only. It is also possible to start the training automatically after each modification or to run the training in the background. When the user is satisfied with the training setup, a folder to automatically classify all contained songs can be selected.

Fig. 20.8 illustrates MUSCLE's user interface. In the main window the playlist containing the classification result in form of a genre tree is displayed. An example for a multiple assignment of the song 'Anticipating' to the classes 'pop' and 'rhythm & base' can be seen in Fig. 20.8(a). In case the user wants to manually adjust the genre assignment of a song, entries can be re-arranged using drag & drop as shown in Fig. 20.8(b).

Part VI

Conclusions and Outlook

Chapter 21

Summary

The analysis of multimedia data is a challenging field of research, especially because the amount of data keeps growing rapidly due to advances in sensor, transmission and storage technology. This increases the need for solutions which can handle the enormous amount of data in an efficient way. The methods and concepts presented in this thesis contribute to the solution of novel analysis algorithms with a special emphasis on the efficiency.

This chapter summarizes the main contributions of this thesis. Directions for possible future research are given in the next chapter (Chapter 22).

21.1 Summary of Contributions

The rapidly increasing amount of data stored in multimedia databases requires efficient and effective analysis methods to make full use of the collected data. This thesis contributes in the field of analyzing spatial, temporal and video & audio data. It proposes new and original solutions for accelerating collision queries during the Digital Mock-Up process, proposes a new threshold-based similarity model for time series, shows how to efficiently perform reverse k-nearest neighbor search in arbitrary metric spaces, and deals with video retrieval and audio classification.

In the following, we give a detailed summary of these contributions.

21.1.1 Preliminaries (Part I)

The preliminaries in Part I provide some motivation and illustrate the topic and the background of this work. For each area, we describe typical problems and briefly sketch an efficient solution for these problems. We also review the related work and introduce the datasets which are used in the following parts to evaluate the new techniques.

21.1.2 Analysis of Spatial Data (Part II)

This part introduces a new generic approach for accelerating spatial query processing for the RI-tree. We use the promising concept of interval groups and show how we can efficiently store them by means of data compression techniques within ORDBMSs. In particular, we introduce a quick spatial data compressor QSDC in order to emphasize those packer characteristics which are important for efficient spatial query processing, i.e. good compression ratio and high unpack speed. Furthermore, we introduce a cost-based decompositioning algorithm for complex spatial objects, called *GroupInt*. *GroupInt* takes the decompression cost of the interval groups and their access probabilities into account. The approach is presented primarily for the

RI-tree, but it can be adapted to other relational index structures with only a few modifications. So the decompositioning algorithm is applicable for different spatial index structures, data space resolutions and compression algorithms. We show in a broad experimental evaluation that our new approach, i.e. the combination of *GroupInt* and QSDC, accelerates the RI-tree by up to two orders of magnitude.

21.1.3 Analysis of Temporal Data (Part III)

Part III proposes 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. We mentioned several practical application domains for such a query type. In addition, we presented a novel approach for managing time series data to efficiently support such threshold queries. Furthermore, we developed a scalable algorithm to answer *threshold queries* for arbitrary thresholds τ. A broad experimental evaluation demonstrates the importance of the new query type for several applications and shows the scalability and quality of our proposed technique in comparison to straightforward as well as existing approaches.

21.1.4 Analysis using Reverse Nearest Neighbor Queries (Part IV)

In Part IV, we propose the MRkNNCoP-Tree as the first index structure for reverse k-nearest neighbor (RkNN) search in general metric spaces where the value for k is specified at query time. Our index is based on the pruning power of the kNN distances of the database points. We propose to approximate these kNN distances conservatively and progressively by a simple function in order to avoid a significant storage overhead. We demonstrate how these approximation functions can efficiently be determined and how any tree-like

metric index structure can be built with this aggregated information. In our broad experimental evaluation, we illustrate that our MRkNN-CoP-Tree efficiently supports the generalized RkNN search in arbitrary metric spaces. In particular, our approach yields a significant increase in speed over the sequential scan and naive indexing solutions. In addition, we demonstrated that our proposed concepts are also applicable for Euclidean RkNN search. We show that our MRkNNCoP-Tree even outperforms the only existing solution for RkNN search for Euclidean vector data.

21.1.5 Analysis of Video & Audio Data (Part V)

Similarity search in multimedia databases can be improved by using multiple representations of the multimedia objects. For example, when searching for similar videos, music features such as rhythm and pitch as well as image features such as color histograms and textures can be used. Multiple representations and multiple instances can also be beneficially applied to data mining tasks, such as classification.

In the first half of Part V, we present a method for effective similarity search in multimedia databases that takes multiple representations of the database objects into account. In particular, we proposed several weighting functions for summarization vectors of different representations of each database object. Our concepts are independent of the underlying summarization method and compute a weight for each summarization vector of each representation for each object separately. Using these weighting factors, we further show how well-known distance measures for non-multi-represented multimedia objects can be adapted to multi-represented objects. In our experiments we evaluate the proposed methods and show the benefits of our approach. The second half of Part V introduces a framework for hierarchical music classification using multiple representations consisting of multiple instances. Our approach is able to handle multiple characteristics of music content and achieves a high classification accuracy efficiently, as shown in our experiments performed on a real-world dataset. We also developed a demonstration tool called MUSCLE which builds on this framework. While

approaches exist in the field of musical genre classification, none of them features a hierarchical classification in combination with interactive user feedback and a flexible multiple assignment of songs to classes. MUSCLE allows the user to organize large music collections in a genre taxonomy and to modify class assignments on the fly.

Chapter 22

Outlook

In this chapter, we point out potentials for future work.

22.1 Future Work

In conclusion, let us emphasize the potentials of the proposed methods for analyzing multimedia data.

- In Part II, we presented a cost-based decomposition approach for voxelized data on the basis of interval groups. The fact that the linearization of the voxels leads to patterns in the byte representation of a spatial object is currently exploited for efficient storage purposes only. A new spatial similarity model which treats the byte sequence as a string of characters could be developed from this representation. A promising approach might be to process the byte-string using the sequence-based similarity models from the field of bioinformatics which extract common subsequences. Another approach would be to transform the spatial objects into time series with the help of space-filling curves. Instead of interpreting the byte representation of an object as a time series consisting of 8-bit measurement points, it could be more promising to measure the density of the spatial object around each voxel. By count-

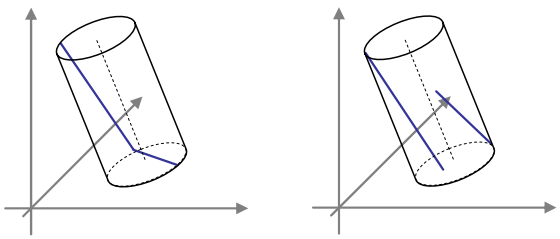


Figure 22.1: A cylinder as approximation of the segments in the parameter-space.

ing the number of voxels in an ε-environment for each position of the space-filling curve, an object could be transformed into a smooth time series. The similarity between objects could then be measured by using the threshold-based technique for time series introduced in Part III.

- Currently, the threshold-based approach in Part III does not perform any grouping of the segments into which a time series is transformed. However, grouping spatial primitives can dramatically improve the performance, as we have shown, by introducing the interval groups. Similar to the interval groups in Part II, spatially close segments in the parameter space could be grouped together before indexing them. A promising conservative approximation of the segments is a cylinder as illustrated in Figure 22.1. It can be described by an axis and the radius, i.e. it needs only one additional parameter compared to a segment.

- We use the Sum of Minimum Distances (*SMD*) as set-valued distance function for computing the threshold similarity of two time series in Part III. The *SMD* consists of two parts, the sum of the nearest-neighbor distances from the query object to the database object and the sum of the nearest-neighbor distances from the database object to the query object. The second part of the *SMD* could also be carried out as a $R1NN$ query. However, the $R1NN$ query has to be performed on a per-object basis and not on the entire database.

- The video retrieval and the audio classification presented in Part V of this thesis rely on low-level features only. Nowadays, there is a shift

of interest towards high-level features. High-level features describe the content of multimedia files on a semantic level, for example, by using keywords. Such an object description can be used to enhance the query process. For example, keywords can be used to formulate semantic queries for a retrieval processor. Another application of high-level features is a semantic analysis of the objects. It is possible, for example, to associate high-level semantics to low-level features with the help of relevance feedback or classification. In the area of image annotation, we could divide an image into several segments, semantically annotate a certain segment and then apply the same annotation to all other segments which have similar low-level feature values.

List of Figures

2.1 Multi-step query processing. 15
2.2 Virtual prototype of a car. 16
2.3 Common queries on spatial data. 17
2.4 Scan conversion on a triangulated surface. 19
2.5 Filling a closed voxelized surface. 20
2.6 Different space-filling curves in a two-dimensional space. . . . 21
2.7 Conversion from a triangulated surface into an interval sequence. 22
2.8 Patterns contained in a linearized object. 24
2.9 Euclidean distance between time series. 29
2.10 Feature based dimensionality reduction (GEMINI approach). . 30
2.11 Patients heart rate and systolic blood pressure after drug treatment. 32
2.12 Detection of associations between different environmental and climatical attributes. 33
2.13 Example time series taken from the *GunX* dataset. 36
2.14 Example time series taken from the *Trace Data* dataset. . . . 37
2.15 Example time series taken from the *Cylinder-Bell-Funnel* dataset. 37
2.16 Example time series taken from the *Synthetic Control* dataset. 38
2.17 Applications for RkNN queries. 40
2.18 Road network graph of the city of Oldenburg. 43
2.19 An image described by multiple representations. 46
2.20 Screenshots of similar music videos. 49

2.21 Conical representation of the HSV color space. 50
2.22 Basic idea of a Support Vector Machine (SVM). 51

4.1 Voxel Linearization Process Chain 62
4.2 Intervals and interval groups. 63

5.1 Pattern derivation by linearizing a voxelized object. 68
5.2 Flow diagram of QSDC compression algorithm. 69

6.1 Query distribution functions $QDF_i(x, y)$. 74
6.2 Computation of the access probability of interval groups. . . . 75
6.3 Evaluation cost $cost_eval$ for different data compressors. . . . 76
6.4 Decomposition Algorithm $GroupInt$. 77

7.1 Analogous architectures for the object-relational embedding. . 80
7.2 SQL statements for spatial object intersection, based on interval group sequences. 84
7.3 Intersection optimizations for interval groups. 86

8.1 Histograms for intervals and gaps. 91
8.2 Storage requirements for the RI-tree (PLANE). 92
8.3 Update operation cost for the RI-tree (CAR)
(i) numerous intervals, (ii) one interval group, (iii) interval groups generated by $GroupInt(QSDC)$. 93
8.4 $MaxGap(DC)$ evaluated for boolean intersection queries on the RI-tree (PLANE). 94
8.5 Filter quality for boolean intersection queries on the RI-tree (CAR). 95
8.6 $MaxGap(QSDC)$ evaluated for boolean intersection queries for the RI-tree using different resolutions. 96
8.7 Window queries (SEQUOIA). 98

9.1 Threshold-based detection of risk patients for heart diseases. . 104

9.2 Threshold-based classification of time series. 105
9.3 The nine basic distances between two intervals A and B. . . . 106

10.1 Threshold-Crossing Time Intervals. 110

11.1 Mapping of Time Intervals to the Time Interval Plane. 117
11.2 Time Intervals in Parameter Space for Arbitrary Threshold. . 118
11.3 Determination of threshold-crossing time intervals from parameter space. 120
11.4 Time Series Decomposition. 122
11.5 Time Series Decomposition Example. 123
11.6 Linear time series decomposition. 124

12.1 Properties of threshold queries w.r.t. object pruning. 131
12.2 Threshold-based ε-range query algorithm. 133
12.3 Example of the threshold-based nearest-neighbor query. 140
12.4 Step-wise lower-bounding distance computation of the threshold-based nearest-neighbor query example. 141
12.5 Threshold-based nearest-neighbor query algorithm. 143

13.1 Scalability of the *threshold-query* algorithm against database size. 148
13.2 Scalability of the *threshold-query* algorithm against time series length. 149
13.3 Pruning Power of the threshold-based nearest-neighbor algorithm. 150
13.4 Comparison to Traditional Distance Measures. 151
13.5 Comparison of Different Interval Similarity Distances. 152
13.6 Comparison of Different Set Similarity Distances. 153

14.1 Example for a RkNN query with $k = 1$. 160

15.1 Using conservative and progressive approximation for RkNN search. . . . 163
15.2 Illustration of the relationships between k and the kNN distance for different synthetic data distributions. . . . 165
15.3 Visualizations of the conservative and progressive approximations of the k-nearest neighbor distances of a sample object for different values of k. . . . 166
15.4 Constraints on the optimal line: Line intersects at least a) one point of the approximated set (Lemma 15.1) b) one point of the UCH (Lemma 15.2) . . . 170
15.5 Illustration of an anchor point. . . . 172
15.6 Illustration of the proof of Lemma 15.3: Monotonicity of the error of the conservative approximation of kNN distances . . . 173
15.7 Finding the optimal approximation. . . . 177
15.8 Aggregated approximation lines. . . . 179

16.1 The RkNN search algorithm. . . . 182

17.1 Runtime w.r.t. database size on Oldenburg dataset. . . . 184
17.2 Runtime w.r.t. parameter k on Oldenburg dataset (2.500 data objects). . . . 185
17.3 Pruning capability w.r.t. parameter k on Oldenburg data set (5.000 data objects). . . . 186
17.4 Comparison of runtime w.r.t. database size. . . . 187
17.5 Comparison of runtime w.r.t. parameter k. . . . 187
17.6 Pruning capability w.r.t. parameter k on color texture dataset. 189

19.1 Basic notations. . . . 196
19.2 Illustration of three different weighting functions. . . . 199
19.3 Different Gaussian distributions of distances from original objects to summarizations. . . . 202

19.4 Precision vs recall for different summarization techniques on best single representation and two best weighting functions. . 206
19.5 Precision vs recall for different summarization techniques on standard combination strategies and proposed weighted combination strategies. 206
19.6 Precision vs recall for different weighting strategies when performing similarity search for videos of the same artist. 207

20.1 Architecture of the proposed framework. 210
20.2 An example genre hierarchy. 211
20.3 Instance reduction with help of support objects. 212
20.4 Border distance based derivation of weights for a multi-represented object. 213
20.5 Accuracy for classification on single- and multi-representations. 215
20.6 Accuracy for classification on single- and multi-representations. 215
20.7 Classification time per object. 216
20.8 MUSCLE User Interface. 217

22.1 A cylinder as approximation of the segments in the parameter-space. 228

List of Tables

1.1 List of author's publications on which this thesis is based. . . 6

4.1 Operators on interval groups. 64
4.2 Operators applied on the example given in Figure 4.2. 65
4.3 Storage of interval groups applied on the example given in Figure 4.2. 66

8.1 Summary of the Spatial Test Datasets. 90
8.2 $GroupInt(DC)$ evaluated for boolean* and ranking** intersection queries for the RI-tree (PLANE). 96
8.3 $GroupInt(ZLIB)$ evaluated for boolean intersection queries for the RI-tree with different resolutions (SEQUOIA). 97

12.1 Notations and operations on time interval sets. 129

13.1 Summary of the Temporal Test Datasets. 146

17.1 Summary of the RkNN Test Datasets. 184

19.1 Summary of the Video Test Dataset. 205

20.1 Summary of the Audio Test Dataset. 214

Bibliography

[ABK+06] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *Proc. SIGMOD*, 2006.

[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. Similarity search on time series based on threshold queries. In *Proc. EDBT*, 2006.

[AKK+06b] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Threshold similarity queries in large time series databases. In *Proc. ICDE*, 2006.

[AM99] R. J. Alcock and Y. Manolopoulos. Time-series similarity queries employing a feature-based approach. In *Proc. 7th Hellenic Conference on Informatics, Ioannina, Greece*, 1999.

[And79] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, 9:216–219, 1979.

[AY99] Y. A. Aslandogan and C. T. Yu. Techniques and systems for image and video retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):56–63, 1999.

[BC94] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAI-94 Workshop on Knowledge Discovery in Databases (KDD-94)*, 1994.

[BHKS93] T. Brinkhoff, H. Horn, H.-P. Kriegel, and R. Schneider. A storage and access architecture for efficient query processing in spatial database systems. In *SSD*, 1993.

[BKK96] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-Tree: An index structure for high-dimensional data. In *Proc. VLDB*, 1996.

[BKK+04] S. Brecheisen, H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Der virtuelle prototyp: Datenbankunterstützung für cad-anwendungen. *Datenbank-Spektrum*, 4(10):22–29, 2004.

[BKK+06] S. Brecheisen, H.-P. Kriegel, P. Kunath, A. Pryakhin, and F. Vorberger. Muscle: Music classification engine with user feedback. In *Proc. EDBT*, 2006.

[BKKP06] S. Brecheisen, H.-P. Kriegel, P. Kunath, and A. Pryakhin. Hierarchical genre classification for large music collections. In *ICME*, 2006.

[BKS93] T. Brinkhoff, H.-P. Kriegel, and R. Schneider. Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In *Proc. ICDE*, 1993.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proc. SIGMOD*, pages 322–331, 1990.

[BSSJ99] R. Bliujute, S. Saltenis, G. Slivinskas, and C. S. Jensen. Developing a datablade for a new index. In *Proc. ICDE*, 1999.

[BW94] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Systems Research Center, Palo Alto, California, 1994.

[CCF+99] W. Chen, J.-H. Chow, Y.-C. Fuh, J. Grandbois J., M. Jou, N. Mattos, B. Tran, and Y. Wang. High level indexing of user-defined types. In *Proc. VLDB*, 1999.

[CF91] K. B. Clark and T. Fujimoto. *Product Development Performance - Strategy, Organization, and Management in the World Auto Industry*. Harvard Business Scholl Press, Boston, MA, 1991.

[CF99] K. Chan and W. Fu. Efficient time series matching by wavelets. In *Proc. ICDE*, 1999.

[CN04] Y. Cai and R. Ng. Index spatio-temporal trajectories with chebyshev polynomials. In *Proc. SIGMOD*, 2004.

[Cor99a] IBM Corp. Ibm db2 universal database application development guide, version 6, 1999. Armonk, NY.

[Cor99b] Oracle Corp. Oracle8i data cartridge developer's guide, release 2 (8.1.6), 1999. Redwood Shores, CA.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-Tree: an efficient access method for similarity search in metric spaces. In *Proc. VLDB*, 1997.

[CSL99] H. S. Chang, S. Sull, and S. U. Lee. Efficient video indexing scheme for content-based retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, 9, 1999.

[CV95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3), 1995.

[CVK04] C. H. L. Costa, J. D. Jr. Valle, and A. L. Koerich. Automatic classification of audio data. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6), 2004.

[CW00] A. Cannane and H. Williams. A compression scheme for large databases. In *Australasian Database Conference*, 2000.

[CZ02] S.S. Cheung and A. Zakhor. Efficient video similarity measurement with video signature. In *Proc. ICIP*, 2002.

[DeM97] M. DeMers. *Fundamentals of Geographic Information Systems*. John Wiley & Sons, New York, 1997.

[Deu96] P. Deutsch. Rfc1951, deflate compressed data format specification, May 1996. http://rfc.net/rfc1951.html.

[DP03] C. Ding and H. Peng. Minimum redundancy feature selection from microarray gene expression data. In *Proc. CSB*, 2003.

[EM97] T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Informatica*, 34(2):103–133, 1997.

[ES93] M. Egenhofer and J. Sharma. Topological relations between regions in r^2 and z^2. In *SSD*, 1993.

[FFS00] J. C. Freytag, M. Flasza, and M. Stillger. Implementing geospatial operations in an object-relational database system. In *SSDBM*, 2000.

[FI92] U. M. Fayyad and K. B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8, 1992.

[FJM97] C. Faloutsos, H. V. Jagadish, and Y. Manolopoulos. Analysis of the n-dimensional quadtree decomposition for arbitrary hyperrectangles. *TKDE*, 9(3):373–383, 1997.

[FL95] C. Faloutsos and K.-I. Lin. Data-mining and visualization of traditional and multimedia datasets. In *Proc. SIGMOD*, 1995.

[FM84] A. Fournier and D. Y. Moniwno. Triangulating simple polygons and equivalent problems. In *ACM Trans. Graph., 3, 2*, pages 153–174, 1984.

[FR89] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *PODS'89*, 1989.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Maolopoulos. Fast subsequence matching in time-series databases. In *Proc. SIGMOD*, 1994.

[FvDFH00] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Longman, 2000.

[Gae95] V. Gaede. Optimal redundancy in spatial database systems. In *SSD*, 1995.

[Geu01] P. Geurts. Pattern extraction for time series classification. In *PKDD*, 2001.

[GFKS02] T. Gärtner, P. A. Flach, A. Kowalczyk, and A. Smola. Multi-instance kernels. In *Proc. ICML*, 2002.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[GGM02] H. Greenspan, J. Goldberger, and A. Mayer. A probabilistic framework for spatio-temporal video representation & indexing. In *Proc. ECCV*, 2002.

[Gut84] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD*, 1984.

[HB01] A. Hampapur and R. M. Bolle. Comparison of distance measures for video copy detection. In *IEEE International Conference on Multimedia and Expo (ICME'01)*, page 188, 2001.

[HS95] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proc. SSD*, 1995.

[HSD73] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE TSAP*, 3(6):6103–623, 1973.

[Huf52] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proc. Inst. Radio Engineers*, 40(9):1098–1101, 1952.

[IL00] G. Iyengar and A. B. Lippman. Distributional clustering for content-based retrieval of images and videos. In *Proc. Int. Conf. Image Processing*, pages 81–84, 2000.

[IS98] Inc. Informix Software. Datablade developers kit user's guide, version 3.4, 1998. Menlo Park, CA.

[Jag90] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *Proc. SIGMOD*, 1990.

[Joh06] T. K. Johnson. *A reformulation of Coombs' Theory of Unidimensional Unfolding by representing attitudes as intervals*. PhD thesis, University of Sydney, Psychology, Sydney, Australia, 2006.

[Kau87] A. Kaufman. An algorithm for 3d scan-conversion of polygons. In *Proc. Eurographics*, 1987.

[KBS93] H.-P. Kriegel, T. Brinkhoff, and R. Schneider. Efficient spatial query processing in geographic database systems. *IEEE Data Engineering Bulletin*, 16(3):10–15, 1993.

[KCMP01] E. Keogh, K. Chakrabati, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. SIGMOD*, 2001.

[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 *Proc. SIGMOD*, 1997.

[KKKP06] H.-P. Kriegel, P. Kröger, P. Kunath, and A. Pryakhin. Effective similarity search in multimedia databases using multiple representations. In *Proc. MMM*, 2006.

[KKPR04a] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Effective decompositioning of complex spatial objects into intervals. In *DBA*, 2004.

[KKPR04b] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Object-relational management of complex geographical objects. In *ACM-GIS*, 2004.

[KKPR05] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. Distributed intersection join of complex interval sequences. In *Proc. DASFAA*, 2005.

[KKPS04] H.-P. Kriegel, P. Kröger, A. Pryakhin, and M. Schubert. Using support vector machines for classifying large sets of multi-represented objects. In *Proc. SDM*, 2004.

[KM00] F. Korn and S. Muthukrishnan. Influenced sets based on reverse nearest neighbor queries. In *Proc. SIGMOD*, 2000.

[KP01] E. Keogh and M. Pazzani. Derivative dynamic time warping. In *Proc. SDM*, 2001.

[KPP+03] H.-P. Kriegel, M. Pfeifle, M. Pötke, M. Renz, and T. Seidl. Spatial data management for virtual product development. *Computer Science in Perspective: Essays Dedicated to Thomas Ottmann*, 2598:216–230, 2003.

[KPPS02] H.-P. Kriegel, M. Pfeifle, M. Pötke, and T. Seidl. A cost model for interval intersection queries on ri-trees. In *SSDBM*, 2002.

[KPPS03a] H.-P. Kriegel, M. Pfeifle, M. Pötke, and T. Seidl. The paradigm of relational indexing: a survey. In *BTW*, 2003.

[KPPS03b] H.-P. Kriegel, M. Pfeifle, M. Pötke, and T. Seidl. Spatial query processing for high resolutions. In *Proc. DASFAA*, 2003.

[KPS00] H.-P. Kriegel, M. Pötke, and T. Seidl. Managing intervals efficiently in object-relational databases. In *Proc. VLDB*, 2000.

[KPS01] H.-P. Kriegel, M. Pötke, and T. Seidl. Interval sequences: An object-relational approach to manage spatial data. In *SSTD*, 2001.

[KR04] E. Keogh and C. Ratanamahatana. Making time-series classification more accurate using learned constraints. In *SDM*, 2004.

[KRSB99] K. V. R. Kanth, S. Ravada, J. Sharma, and J. Banerjee. Indexing medium-dimensionality data in oracle. In *Proc. SIGMOD*, 1999.

[KS04] H.-P. Kriegel and M. Schubert. Classification of websites as sets of feature vectors. In *Proc. DBA*, 2004.

[Lev44] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly J. Appl. Math.*, 2:164–168, 1944.

[LZ77] A. Lempel and J. Ziv. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[MH99] T. Möller and E. Haines. *Real-Time Rendering*. A.K. Peters Ltd., Natick, MA, 1999.

[MJFS96] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of hilbert space-filling curve. Technical Report 3611, University of Maryland, College Park, MD, 1996.

[MP94] C. B. Medeiros and F. Pires. Databases for gis. *SIGMOD Record*, 23(1):107–115, 1994.

[MPT99] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *SIGGRAPH*, 1999.

[MTT00] Y. Manolopoulos, Y. Theodoridis, and V. J. Tsotras. *Advanced Database Indexing*. Kluwer, Boston, 2000.

[NWH01] M. Naphade, R. Wang, and T. Huang. Multimodal pattern matching for audio-visual query and retrieval. In *Proc. SPIE*, 2001.

[Ore89] J. Orenstein. Redundancy in spatial databases. In *Proc. SIGMOD*, 1989.

[PCST99] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin dags for multiclass classification. In *Proc. NIPS*, 1999.

[RH93] M. A. Roth and S. J. Van Horn. Database compression. *SIGMOD Record*, 22(3), 1993.

[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. 9th Pacific-Asian Int. Conf. on Knowledge Discovery and Data Mining (PAKDD'05), Hanoi, Vietnam*, 2005.

[Rov02] D. Roverso. Plant diagnostics by transient classification: The aladdin approach. *IJIS, Special Issue on Intelligent Systems for Plant Surveillance and Diagnostics*, 17:767–790, 2002.

[SAA00] Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Reverse nearest neighbor queries for dynamic databases. In *Proc. DMKD*, 2000.

[Sai94] N. Saito. *Local feature extraction and its application using a library of bases*. PhD thesis, Yale University, New Haven, Connecticut, 1994.

[SBP98] H.-P. Kriegel S. Berchtold and M. Pötke. Database support for concurrent digital mock-up. In *Proc. IFIP Int. Conf. PROLAMAT*, 1998.

[Sch91] M. Schroeder. *Fractals, Chaos, Power Laws: Minutes from an infinite paradise*. W.H. Freeman and company, New York, 1991.

[SFGM93] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The sequoia 2000 storage benchmark. In *Proc. SIGMOD*, 1993.

[SFT03] Amit Singh, Hakan Ferhatosmanoglu, and Ali Saman Tosun. High dimensional reverse nearest neighbor queries. In *Proc. CIKM*, 2003.

[SHP98] T. Seidl and Kriegel H.-P. Optimal multi-step k-nearest neighbor search. In *Proc. SIGMOD*, 1998.

[SJL+03] J. R. Smith, A. Jaimes, C-Y. Lin, M. Naphade, A. P. Natsev, and B. Tseng. Interactive search fusion methods for video database retrieval. In *Proc. ICIP*, 2003.

[SK93] M. Schiwietz and H.-P. Kriegel. Query processing of spatial objects: Complexity versus redundancy. In *SSD*, 1993.

[SMS+00] J. Srinivasan, R. Murthy, S. Sundara, N. Agarwal, and S. DeFazio. Extensible indexing: A framework for integrating domain-specific indexing schemes into oracle8i. In *Proc. ICDE*, 2000.

[SN02] R. Steinmetz and K. Nahrstedt. *Multimedia Fundamentals, Volume 1: Media Coding and Content Processing, Second Edition*. Prentice Hall, 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.

[TC02] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE TSAP*, 10(5):293–302, 2002.

[TK00] T. Tolonen and M. Karjalainen. A computationally efficient multipitch analysis model. *IEEE TSAP*, 8(6):708–716, 2000.

[TKR99] Y.-P. Tan, S.R. Kulkarni, and P.J. Ramadge. A framework for measuring video similarity and its application to video query by example. In *IEEE International Conference on Image Processing (ICIP'99)*, volume 2, pages 106–110, 1999.

[TPL04] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse kNN search in arbitrary dimensionality. In *Proc. VLDB*, 2004.

[VLB05] J. R. R. Viqueira, N. A. Lorentzos, and N. R. Brisaboa. *Survey on Spatial Data Modelling Approaches*. Idea Group, 2005.

[WLCS04] Y. Wu, C.-Y. Lin, E. Chang, and J. R. Smith. Multimodal information fusion for video concept detection. In *Proc. ICIP*, 2004.

[YF00] B. K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proc. VLDB*, 2000.

[YL01] Congjun Yang and King-Ip Lin. An index structure for efficient reverse nearest neighbor queries. In *Proc. ICDE*, 2001.

[Zha03] T. Zhang. Semi-automatic approach for music classification. In *Proc. SPIE Conf. on Internet Multimedia Management Systems*, 2003.

[ZRHM98] Y. Zhuang, Y. Rui, T. S. Huang, and S. Mehrotra. Adaptive key frame extraction using unsupervised clustering. In *Proc. ICIP*, 1998.