Matthias Renz

Enhanced Query Processing on Complex Spatial and Temporal Data

Dissertation an der Fakultät
für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München,
eingereicht am 16. Oktober 2006.

Enhanced Query Processing on Complex Spatial and Temporal Data

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

von

Matthias Renz

Tag der Einreichung: 16.10.2006
Tag der mündlichen Prüfung: 05.12.2006

Berichterstatter:
Prof. Dr. Hans-Peter Kriegel, Ludwig-Maximilians-Universität München
Prof. Dr. Thomas Seidl, Rheinisch-Westfälische Technische Hochschule Aachen

Acknowledgement

I would like to express my deepest thanks to all people which supported and encouraged me in the past years while i was working on this work, even if I cannot mention them all by name.

I would especially like to thank my supervisor and first referee on this thesis, Prof. Dr. Hans-Peter Kriegel. He initiated and supported this work with his long standing experiences and the organizational background and gave me the opportunity to work on this challenging domain. Without the inspiring, productive and supportive working environment, he created in the database research group, this work could never have been done. I am also very grateful to Prof. Dr. Thomas Seidl for his interest in my work and his immediate willingness to act as the second referee.

This work was inspired by many fruitful discussions and cooperations with my colleagues in the database research group. In particular, I want to thank Peter Kunath, Dr. Peer Kröger, Alexey Pryakhin, Johannes Assfalg and Dr. Martin Pfeifle for constructive and productive team work and Elke Achtert, Prof. Dr. Christian Böhm, Karsten Borgwardt, Stefan Brecheisen, Dr. Matthias Schubert and Arthur Zimek for many helpful discussions.

I also appreciate the substantial help of the students whose study thesis or diploma thesis I supervised. They helped me to manage the large amount of necessary tasks including implementation, data processing, and testing.

I am extremely grateful for the background support of Susanne Grienberger, who managed much of the administrative work and carefully read this thesis to polish my English. Furthermore, I want to express special

thanks to Franz Krojer, for taking care of our technical environment and unhesitatingly providing me with all the technical tools that helped to accelerate the progress of this work.

Last but not least, I want to thank my family and friends for their support and encouragement during the time that I was engaged in this study. In particular I thank my parents who always supported my career and encouraged me to find my way. But especially I want to thank my wife Helene for all the love and care she gave me, for her great encouragement and for many sacrifices she shouldered in the last month. I thank her for the motivations she gave me when I was down and for giving me the power I needed to complete this work.

Abstract

Innovative technologies in the area of multimedia and mechanical engineering as well as novel methods for data acquisition in different scientific subareas, including geo-science, environmental science, medicine, biology and astronomy, enable a more exact representation of the data, and thus, a more precise data analysis. The resulting quantitative and qualitative growth of specifically spatial and temporal data leads to new challenges for the management and processing of complex structured objects and requires the employment of efficient and effective methods for data analysis.

Spatial data denote the description of objects in space by a well-defined extension, a specific location and by their relationships to the other objects. Classical representatives of complex structured spatial objects are three-dimensional CAD data from the sector "mechanical engineering" and two-dimensional bounded regions from the area "geography". For industrial applications, efficient collision and intersection queries are of great importance.

Temporal data denote data describing time dependent processes, as for instance the duration of specific events or the description of time varying attributes of objects. Time series belong to one of the most popular and complex type of temporal data and are the most important form of description for time varying processes. An elementary type of query in time series databases is the similarity query which serves as basic query for data mining applications.

The main target of this thesis is to develop an effective and efficient

algorithm supporting collision queries on spatial data as well as similarity queries on temporal data, in particular, time series. The presented concepts are based on the efficient management of interval sequences which are suitable for spatial and temporal data. The effective analysis of the underlying objects will be efficiently supported by adequate access methods.

First, this thesis deals with collision queries on complex spatial objects which can be reduced to intersection queries on interval sequences. We introduce statistical methods for the grouping of subsequences. Involving the concept of multi-step query processing, these methods enable the user to accelerate the query process drastically. Furthermore, in this thesis we will develop a cost model for the multi-step query process of interval sequences in distributed systems. The proposed approach successfully supports a cost based query strategy.

Second, we introduce a novel similarity measure for time series. It allows the user to focus specific time series amplitudes for the similarity measurement. The new similarity model defines two time series to be similar iff they show similar temporal behavior w.r.t. being below or above a specific threshold. This type of query is primarily required in natural science applications. The main goal of this new query method is the detection of anomalies and the adaptation to new claims in the area of data mining in time series databases. In addition, a semi-supervised cluster analysis method will be presented which is based on the introduced similarity model for time series.

The efficiency and effectiveness of the proposed techniques will be extensively discussed and the advantages against existing methods experimentally proofed by means of datasets derived from real-world applications.

Zusammenfassung

Innovative Technologien in den Bereichen Multimedia und Maschinenbau sowie neueste Methoden zur Datengewinnung in den verschiedensten Teilbereichen der Wissenschaft, wie z.B. Geo-Wissenschaften, Umwelt- und Klimaforschung, Medizin, Biologie und Astronomie, ermöglichen eine exaktere Darstellung und somit eine präzisere Analyse der Daten. Die daraus folgende quantitative und qualitative Zunahme, insbesondere von räumlichen und zeitlichen Daten, führt zu neuen Herausforderungen bei der Verwaltung und Verarbeitung von komplex strukturierten Objekten und erfordert den Einsatz von effizienten und effektiven Methoden zur Datenanalyse.

Unter räumlichen Daten versteht man die Beschreibung von Objekten im Raum durch wohldefinierte Ausdehnung, Lage im Raum und ihre Beziehungen zu anderen Objekten. Klassische Vertreter von komplex strukturierten räumlichen Objekten aus dem Bereich Maschinenbau sind CAD-Daten (3D-Objekte) und aus der Geographie begrenzte Flächenstücke (2D-Objekte). Für industrielle Anwendungen sind effiziente Kollisions- und Schnittanfragen von großer Bedeutung.

Zeitliche Daten sind Daten, die zeitabhängige Vorgänge beschreiben, wie beispielsweise die Dauer von bestimmten Ereignissen oder die Beschreibung von sich zeitlich verändernden Attributen von Objekten. Zu den am häufigsten verwendeten zeitlichen Daten gehören Zeitreihen, welche eine der komplexesten aber auch wichtigsten Beschreibungsformen für sich zeitlich ändernde Vorgänge sind. Eine elementare Anfrageform in Zeitreihen-Datenbanken ist die Ähnlichkeitsanfrage, die als Basisanfrage für "Data Mining" Anwendungen dient.

Ziel dieser Arbeit ist die Entwicklung von effektiven und effizienten Algorithmen, die sowohl für Kollisionsanfragen auf räumliche Daten als auch für Ähnlichkeitsanfragen auf zeitlichen Daten, insbesondere auf Zeitreihen, verwendet werden können. Die vorgestellten Konzepte basieren auf der effizienten Verwaltung von Intervall-Sequenzen. Diese eignen sich als Basisdatentyp für räumliche und zeitliche Daten. Die effektive Analyse der zugrunde liegenden Objekte wird durch den Einsatz von geeigneten Zugriffsstrukturen effizient unterstützt.

Zum einen beschäftigt sich diese Arbeit mit Kollisionsanfragen von komplexen räumlichen Objekten, die sich auf Schnitterkennung zwischen Intervallsequenzen reduzieren lassen. Es werden statistische Methoden zur Gruppierung von Teilsequenzen eingeführt, die es ermöglichen, mittels mehrstufiger Anfragebearbeitung den Anfrageprozess erheblich zu beschleunigen. Ferner wird ein Kostenmodell für die mehrstufige Anfragebearbeitung von Intervallsequenzen in verteilten Systemen entwickelt, welches eine kostenbasierte Anfragestrategie erfolgreich unterstützt.

Zum anderen wird ein neuer Ähnlichkeitsbegriff für Zeitreihen eingeführt, der es erlaubt, Ähnlichkeitsmessungen auf bestimmte Amplituden der Zeitreihen zu fokusieren. Dieses neue Ähnlichkeitsmaß definiert zwei Zeitreihen als ähnlich, wenn sie ein vergleichbares Verhalten bezüglich Überschreitungen eines bestimmten Grenzwertes aufweisen. Hauptsächlich finden Anfragen basierend auf diesem Ähnlichkeitstyp in naturwissenschaftlichen Gebieten Anwendung. Ziel dieser neuen Anfragemethode ist die Erkennung von Anomalien und die Adaption an neue Anforderungen im Bereich "Data Mining" in Zeitreihen-Datenbanken. Des Weiteren wird auf der Grundlage des eingeführten Ähnlichkeitsmodells eine Methode zur semi-überwachten Cluster-Analyse entwickelt.

Die Effizienz und Effektivität der vorgestellten Techniken wird ausgiebig diskutiert und die Vorteile gegenüber herkömmlicher Verfahren mittels Datensätzen aus realen Anwendungen experimentell nachgewiesen.

Survey of Chapters

Contents

Part I

Complex Spatial and Temporal Data

Chapter 1

Introduction

The substantial progress in sensor technology in recent years and the increasing growth of storage technologies have facilitated new prospects in the recording of data derived from complex real-world objects or observed processes. The ability to accurately map these objects in our digital world and the request of handling larger and larger collections of data requires huge memory space. This evolution is very challenging for modern information systems which have to process large amounts of data in order to retrieve the relevant information or help to find yet unknown knowledge from the recorded data in an efficient way. In particular, due to the naturally complex structure of objects or processes of our real life, modern information systems ask for more and more efficient and effective data retrieval methods.

In this thesis, we first propose methods with the special focus on efficient management and effective processing of highly resolved spatial objects. The ability to handle complex structured spatial objects is very important as real-world objects naturally have a spatial extension in one or more dimensions and most of them are characterized by extremely complex shapes. Special methods which can cope with large collections of highly resolved spatial objects are necessary. Secondly, we propose effective and efficient similarity search methods for time series, one of the most complex structured type of temporal data. Time series are an important type of data as they describe time varying object states, and are the most common way to describe dy-

namic processes or observations of our real life. The fact that time series may cover an arbitrarily long time range and, moreover, that the dynamics of the observed entities may be very high makes an efficient and effective handling of this complex data difficult.

There are a lot of applications which require an accurate recording of object data and the potentials of state-of-the-art digital analysis gain importance, in particular for life science, medical science, environmental science and mechanical engineering. The efficient support of recording, processing and retrieval of objects, derived from our physical world or created from engineering design, is absolutely necessary for modern information systems and pose a challenge for the development of efficient and effective query algorithms.

1.1 Complex Spatial and Temporal Objects in Information Systems

We start with a general definition of *Information Systems*.

Information systems are systems consisting of components for the recording, managing, processing/analysis and presentation of information [LM86]. They do not only serve as simple management tools for displaying existing data, but are mainly used to derive novel data and knowledge by means of (complex) processing and analysis operations.

Modern information systems are able to handle complex data like spatial objects and/or time varying data. In order to cope with the complex nature of objects of a modern information system, we have to extract and decompose them into a set of manageable entities which allows us to store, manage, analyze, and present huge amounts of such data efficiently. The entities used to represent the objects of, and support operations on our small *universe* should have a very simple structure compared to the described objects, even though often only an approximative description, for example a minimal bounding box, can be achieved. The choice of an adequate type of

approximation used to describe the objects of course depends heavily on the type of objects but also depends on the particular application.

1.1.1 Spatial Information Systems

Spatial information systems efficiently manage data related to a space. More precisely, spatial information systems are the technology of acquiring, managing, analyzing, and displaying information in a spatial context. Computer Aided Design (CAD) and Geographic Information Systems (GIS) are the most prominent examples of this expanding technology in the spatial context. Examples for complex spatial objects are *real estates* for land registry or three-dimensional *parts* for mechanical engineering. The simplest and most popular form of representation of the objects often used in this context are minimal bounding rectangles (MBRs), conservatively covering the spatial objects. This representation can be easily handled and we can apply efficient access methods, as for instance the R^*-tree [BKSS90]. Since for complex structured spatial objects one-value approximations are often to coarse, they are commonly represented by a set of spatial primitives like simple tiles or intervals which yield a more accurate conservative covering of the spatial objects. As spatial information systems are often used in industrial environments, e.g. for mechanical engineering tasks, it is important to provide the integration of spatial data management into existing Engineering Data Management systems (EDM) [Pöt01]. In particular, the interval based representation of spatial objects is very suitable for such an integration, using the concept of relational indexing [KPPS03b]. For this reason, in this thesis we use the interval based form of object approximation for spatial objects.

1.1.2 Temporal Information Systems

Temporal Information systems provide handling data which change with time. Examples are descriptions of any activity during a time frame, like access activities for certain web sites, records of stock prizes or daily measure sequences of meteorological attributes. Generally, we can distinguish

two research directions, distinct with respect to their focus, that can be easily identified in the literature:

- *Temporal reasoning* supports various inference tasks, involving time-oriented data such as planning and execution, and traditionally has been linked with the machine-learning and artificial-intelligence community.

- *Temporal data maintenance* deals with storage and retrieval of data that have heterogeneous temporal dimensions, and typically is associated with the (temporal) database community.

In this thesis, we follow the direction of temporal data maintenance focusing on the storage and retrieval of time series. Though time series, sometimes also denoted as time sequences, are often disregarded in the context of temporal database systems, they are the most important but also most complex type of temporal data. In generally, time series are used to describe activities of time varying entities. In most applications a time series represents the activity of only one attribute. Time series data play a key role, for instance, in biological- and medical information systems. Examples are blood pressures of patients recorded over a specific range of time for medical observations of patients. They are important for medical decision making, e.g. in clinical diagnosis and therapy planning.

The most important type of query for time series databases are similarity queries. Thereby, time series are commonly represented as point objects in a high-dimensional vector space and efficiently organized in point access methods. For efficiency reasons, dimensionality reduction methods are applied on the original time series objects before applying any access method. In this thesis, we propose another time series reduction method based on a novel similarity model.

1.2 Applications of Spatial and Temporal Information Systems

In the following, we exemplarily emphasize the importance of spatial and temporal database systems by illustrating different applications of modern information systems from biology and medicine, multimedia, geography [DeM97] and mechanical engineering [BKP98].

1.2.1 Biological- and Medical Information Systems

Medicine and molecular biology are nowadays data driven sciences, where electronic access to terminology, information resources and tools is crucial for success and leads to cross-fertilization in both domains. Medical information systems are used to collect, process and publish health care information. They include systems to collect, store and retrieve data about specific patients and systems to assemble, store relations of cause and effect using available medical knowledge. Thus, there is a wide range of medical information resources to evaluate.

Providing effective retrieval solutions for efficient handling of time-oriented data, in particular data which change with time are very important for *Biological- and Medical Information Systems*. Hence, the design of effective and efficient analysis methods for time series is a major theoretical and practical research area in medicine and biology.

1.2.2 Multimedia Information Systems

Obviously, one of hte main purposes of a multimedia database is to organize multimedia data. Unlike numeric or character data, multimedia data requires huge amounts of storage capability and increased processing speed. Multimedia comprises many kinds of media, including image, video, audio, graphics, animation, hypertext, and hypermedia.

A still *image* is just a static picture. But even a single static image can require large amounts of storage. It is possible that a color picture needs as much storage space as an entire book - over one million bytes. Unlike still pictures, *video* contains temporal as well as spatial relationships across frames. Temporal relationships such as "before", "after", and "during" have an additional meaning to spatial ones. Consequently, $MPEG$ compression utilizes spatial (intra-frame) as well as temporal (inter-frame) redundancy elimination: Frames are compressed "spatially" (based on portions of a given frame containing identical data) and "temporally" (based on preceding frames containing identical data). *Audio* such as music, human voice, and sound comprises frequency and amplitude. Audio, like video, can be compressed, and it is usually interlocked (compressed and/or stored together) with its related video. The most interesting and exciting thing about multimedia databases is how quickly they evolve. This growth will ignite an explosion of multimedia applications. This explosion, in turn, will fuel an intense need for powerful multimedia databases.

1.2.3 Geographic Information Systems

In the past couple of decades, computer based systems have been developed and designed specifically to handle geographical information. These systems are generally known as Geographical Information Systems or more commonly abbreviated as GIS. They are designed to allow the user the capture of spatial information, its storage, analysis, manipulation and the production of maps as outputs. The development of computer systems to handle spatial information has changed our perceptions of information and there is now an enormous range and quantity of information held within these systems.

1.2.4 Computer Aided Design

Computer aided design (CAD) is the modeling of physical systems on computers, allowing both interactive and automatic analysis of design variants, and the expression of designs in a form suitable for manufacturing. CAD sys-

tems are designed to create, manipulate and analyze detailed two- or three-dimensional models of physical objects, such as mechanical parts, buildings, and molecules. In contrast to *GIS*, *CAD* systems have to cope with object models of high precision which makes it challenging for efficient retrieval methods.

1.3 Outline of this Thesis

Specialized techniques are required to manage effectively and efficiently complex spatial and temporal objects in modern information system. The contributions of this thesis are mainly focused on techniques for speeding up query processing on highly resolved spatial objects, in particular *CAD* data, and the enhancement of similarity search in time series databases.

The starting point of this thesis is a broad overview of modeling techniques and methods for managing complex structured spatial and temporal data, specifically three-dimensional CAD objects and time series objects. The mutual base of both types of data is the ability to reduce the representation to simple sequences of one-dimensional intervals. In the case of spatial objects, the intervals describe locally the occupancies of spatial primitives associated with an object. Time series, in contrast, are represented by means of time intervals describing the duration of certain events. The advantage of this kind of data representations is that intervals or interval sequences are more easy to handle than the original object representations. In this thesis, we propose techniques which help us to cope with large collections of the two types of complex structured objects on the basis of interval sequences representations.

The major contributions of this thesis include:

- Using statistical models to accelerate spatial queries on interval-based representations of spatial objects organized within object-relational DBMS.
- An adaption of the statistic driven query processing method to provide

database support for haptic exploration tasks.

- Data adaptive approximation of spatial objects based on a cost model and compression techniques allowing to minimize the I/O cost.
- Statistic driven methods to accelerate the processing of spatial joins.
- A cost based spatial join algorithm for distributed data.
- A novel similarity measure for time series enabling data mining tasks focused on certain amplitude values (threshold values).
- An efficient threshold invariant data decomposition method and an efficient access method for threshold based similarity search.
- An efficient pruning strategy for threshold-based distance-range queries and threshold-based nearest-neighbor queries.
- A semi-supervised time series analysis approach based on the threshold-based similarity measure.

The remaining chapters of Part I (Chapter 2-4) provide a brief and rather general overview over existing models of data representation for spatially extended objects and time series objects, and give a short sketch on managing intervals and interval sequences. The concepts described in this chapter form the basis of the proposed techniques. In addition, the chapter introduces some basic notations used throughout this thesis.

Part II introduces some approaches involving statistical information in order to accelerate the processing of spatial intersection queries for linearized spatial objects.

Chapter 5 gives a brief introduction and motivates to manage complex spatial objects in a linearized form that supports an easy integration into commercial database systems.

Chapter 6 shows how statistics can be used for accelerating relational access methods by reducing the number of generated join partners which results in fewer logical reads and consequently improves the overall runtime. We cut

down the number of join partners by suitably grouping several join partners together according to statistics about data distribution and a statistic driven cost model.

Chapter 7 presents an approach that achieves efficient query processing along with industrial-strength database support for real time haptic rendering systems which compute force feedback (haptic display). This approach externalizes and accelerates the approved main memory $Voxmap-PointShell^{TM}$ (VPS) approach. Analog to Chapter 6 we group numerous independent database queries together according to a cost model which takes statistical information, reflecting the actual data distribution, into account.

Chapter 8 introduces interval containers as a new and general concept to approximate interval sequences. In contrast to the common interval decomposition that suffers from the high redundancy of complex shaped objects resolved with high resolution, interval containers combine groups of intervals into approximations - one approximation per group. Additionally, we store the exact information of these interval containers in a compressed way. 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 9 investigates two spatial join methods for two sets of very complex spatial objects, a nested-loop based join algorithm and a sort-merge based join algorithm. We present an approach that is based on a fast filter step, performing the spatial join on simple primitives which conservatively approximate the objects. Our main attention is focused on the problem how to generate approximations adequate for speeding up the join of high-resolution objects.

Chapter 10 presents a new distributed intersection join for interval sequences of high cardinality which tries to minimize these transmission cost. This approach is based on a suitable probability model for interval intersections. The probability model is used to speed up the processing of the data at server site as well as on the various clients. On the client sites, we group intervals based on this probability model together. These 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 or not, we fetch the exact information of the most promising approximations first. This strategy helps to cut down the transmission cost considerably.

Part III introduces the new concept of threshold-based similarity search for time series databases. Like the form of representation used for the spatial objects as proposed in Part II, in this concept, time series are also represented by means of interval sequences. 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 to develop 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 representations. Furthermore, we show how the time series can be indexed that the new similarity queries can be processed in an very efficient way.

Chapter 11 motivates the novel similarity model for time series and preliminarily give some definitions required in the remaining chapters. Exemplarily it is shown that this new analysis concept is important for several applications in medicine, biology and for analysis of environmental air pollution.

Chapter 12 gives an overview of several distance measures for one-dimensional intervals and we discuss whether these measures suitably reflect the similarity between the intervals.

Chapter 13 formally introduces a novel similarity measure called *threshold distance* and defines two query methods based on this similarity measure, the *threshold-based ε-range query* and the *threshold-based k-nearest-neighbor query*.

Chapter 14 shows how time series can be decomposed into manageable entities. In particular, we propose an efficient decomposition algorithm. Then the resulting entities can be suitably transformed into (low-dimensional) spatial entities which, in turn, can be efficiently organized by means of conventional spatial access methods like the R^*-tree. The advantage of this indexing

concept is that at query time only those parts of the times series which are relevant for the query have to be accessed. The retrieved information suffices to answer the query correctly. Consequently, we do not need an additional refinement step which usually requires to load the entire time series information from disk.

Chapter 15 presents efficient algorithms for the two similarity queries proposed in Chapter 13. Both algorithms follow the filter-refinement paradigm. A suitable pruning strategy used to filter out true drops as early as possible is introduced for both query algorithms. Furthermore, we develop a filter distance that lower-bounds the threshold distance and which is used for the pruning strategy of the threshold-based nearest-neighbor query. Thereby, the filter step is performed in several iterations. In each iteration, the filter distance can be efficiently computed for all objects in the database and converge to the exact distance with increasing iterations. This strategy allows us to prune several true drops as early as possible.

Chapter 16 proposes a novel approach for cluster analysis of time series based on adaptable threshold similarity. The most important issue in threshold similarity is the choice of the threshold τ. Thus, the threshold τ is automatically adapted to the characteristics of a small training dataset using the silhouette width value which is a measure of cluster validity. Thus, the optimal value of the parameter τ is learned from a small training set in order to yield an accurate clustering of the entire time series database.

Chapter 17 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) on several established benchmark datasets. Furthermore, we evaluate different similarity measures for the proposed time series representation. Additionally, we demonstrate the usefulness of the semi-supervised time series analysis. It is shown that our cluster analysis using adaptable threshold similarity can be successfully applied to many scientific real-world data mining applications. For instance,

we show that semi-supervised threshold queries applied to gene expression data are very worthwhile.

Part IV concludes this thesis.

Chapter 18 summarizes and discusses the major contributions of the thesis. It concludes with indicating some potentials for possible future research directions.

Chapter 2

Complex Spatial Data

Spatial data denotes data which is related to a specific space. Typical instances of the spatial data represent objects which descend from our physical world or artificial objects derived from engineering design. Each database object has an identity and a well-defined location and extension in space. Spatial data types represent the spatial objects while spatial predicates describe the relationships between them. In this chapter, we give a brief introduction into the well-studied area of spatial data modeling and spatial queries. For in-depth descriptions, surveys, and evaluations, we refer the reader to the referenced literature.

2.1 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 our 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. In this section, we first give a general overview of spatial data modeling, followed by the presentation of specific modeling

techniques taking special emphasize on three-dimensional *CAD*-objects.

Data modeling requires specifying, at least the following three main components: spatial data type, data structures and operations on spatial data, in particular, spatial predicates [VLB05]. As spatial data is mainly associated with Geographic Information Systems (GIS), it is often defined as information that describes the distribution of things upon the surface of the earth. Actually, in the geographical context, spatial data contains information concerning the location, shape of and relationships among geographic features [DeM97]. Consequently, traditional spatial data types include points, lines and polygons allowing to represent any geographical entity.

The spatiality of an application is reflected by the existence of spatial entities, but also by the existence of spatial relationships between these entities. Queries on spatial databases typically evaluate the relationships by spatial predicates, allowing users to query the database to return a *true* or *false* answer. Spatial predicates can be mainly categorized into three different types: topological, directional and metrical. Topological properties are the most fundamental primitives for spatial predicates. The principle topological relationships between two spatial objects have been captured by the 9-intersection model proposed by Egenhofer and Sharma [ES93]. It includes the following predicates: *disjoint*, *meets*, *equal*, *inside*, *contains*, *covers*, *covered-by*, and *overlaps*. Directional predicates evaluate the relative position of spatial objects, like *above*, *below*, *left* or *right*. Metric predicates are associated with quantitative information like distance between two objects or the size of the overlapping area of two objects. The most important predicate used in many mechanical engineering and architecture applications is the *intersection* predicate. It is a combination of the two basic predicates "*not disjoint*" and "*not meets*". As already mentioned above, in this thesis we take special emphasize on the efficient evaluation of the intersection predicate applied to three-dimensional objects, in particular *CAD*-objects.

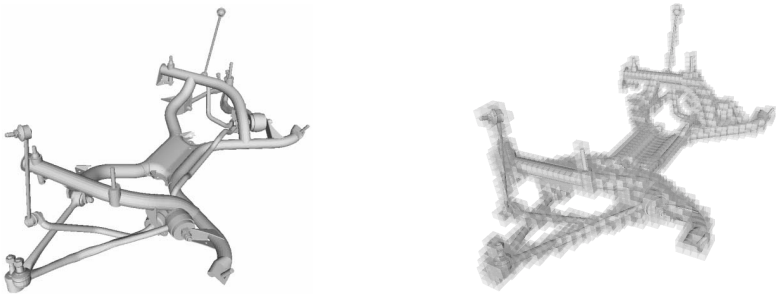
2.1.1 Modeling 3-dimensional Objects

As they are the natural instances of our real world, the modeling of three-dimensional objects are 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 of the most emerging technologies in the field of spatial data management and are getting more and more indispensable in mechanical engineering facilities. In order to cope with the demands of accurate geometric modeling, we defined a set of 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 selection or collision queries. The supported geometric data models include triangle meshes for visualization and interference detection, voxel sets as conservative approximations for spatial keys, and point shells to enable haptic queries with interactive response time.

2.1.2 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]. As a solution, approximative polygon meshes, e.g. triangle meshes, can be derived from the accurate surface representation. These triangle meshes allow an efficient and interactive display of complex objects, for instance by means of VRML encoded files, and serve as an ideal input for the computation of spatial interference. For the digital mock-up (DMU), for instance, spatial interference detection or collision queries are a very important database primitive.

In the following, we assume a multi-step query processor which retrieves a candidate part S, possibly colliding with a query part Q. In order to refine such collision queries, a fine-grained spatial interference detection be-



(a) Triangulated surface **(b)** Voxelized surface

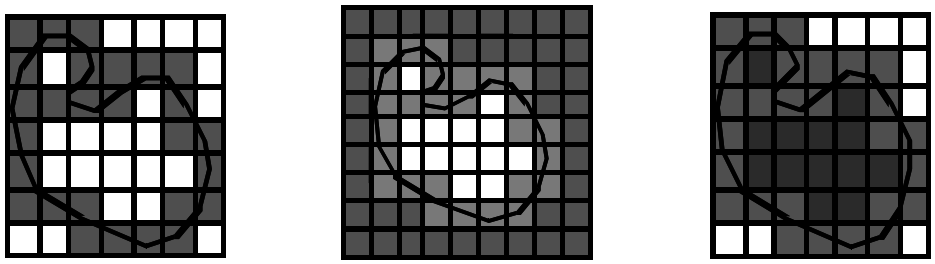
Figure 2.1: Scan conversion on a triangulated surface.

tween Q and S can be implemented on their triangle meshes. We distinguish three actions for interference detection [MH99]: collision detection, collision determination, and collision response:

Collision detection: This basic interference check simply detects if the query part Q and a stored part S collide. Thus, collision detection can be regarded as a geometric intersection join of the triangle sets for S and Q which already terminates 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.

Collision response: It determines the actions to be taken in consequence of a positive collision detection or determination. In our case of a spatial database for virtual engineering, a textual, visual or haptic feedback on the interfering parts seems to be appropriate. Note, that the haptic feedback requires not only to compute and report the power of the resulting force, but we have to report also the force direction and the corresponding anchor point of the force vector.



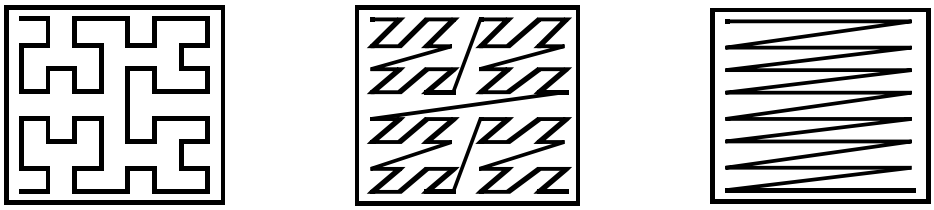
(a) Voxelized surface **(b)** Filled exterior **(c)** Inverted result

Figure 2.2: Filling a closed voxelized surface.

2.1.3 Voxel-Sets and Voxel-Sequences

In order to employ efficient access methods for interval sequences like the Relational Interval-tree (RI-tree)(cf. chapter 4.4.1) 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 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 2.1(a)), a conservative approximation of the part surface is produced (cf. Figure 2.1(b)). In the following, we assume a uniform three-dimensional voxel grid covering the global 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 to voxelize the triangle mesh of a solid object first (cf. Figure 2.2(a)) which yields a consistent representation of the object surface. Next, we apply a 3D flood-fill algorithm [FvDFH00] to compute the



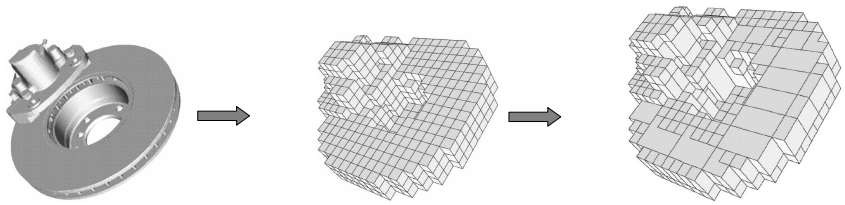
(a) Hilbert order **(b)** Z-order **(c)** Lexicographic order

Figure 2.3: Examples of space-filling curves in a two-dimensional space.

exterior voxels of the object (cf. Figure 2.2(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 as interior which are neither boundary nor exterior voxels (cf. Figure 2.2(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 consistent 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 in close spatial neighborhood. Examples for space filling curves include Hilbert-, Z-, and the Lexicographic-order, depicted in Figure 2.3. 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 which can be described by



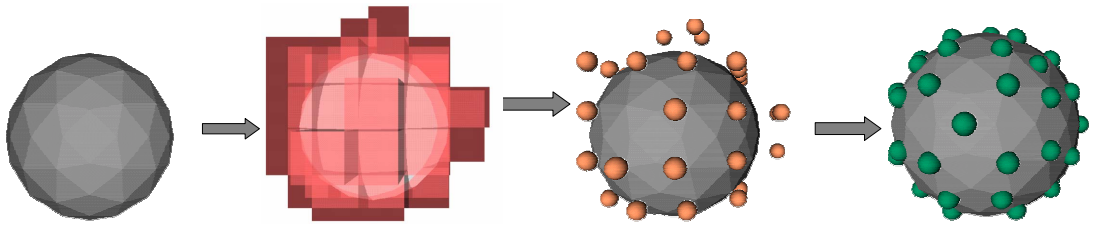
(a) Triangle mesh **(b)** Voxel set **(c)** Interval sequence

Figure 2.4: Conversion pipeline from triangulated surfaces to interval sequences.

a sequence of intervals. Figure 2.4 summarizes the complete transformation process from triangle meshes over voxel sets to interval sequences.

2.1.4 Point Shells

In order to achieve real-time interference detection for moving objects, two properties of the geometric representation are of major importance: (1) efficient geometric transformations, e.g. translation and rotation, and (2) efficient intersection tests. Triangle meshes naturally qualify for (1), as their topology is invariant to geometric transformations of triangle vertices. If consecutive triangle intersection joins rely on hierarchical indexes like OBB-Trees or k-DOPTrees [GLM96][KHM+98], criterion (2) is principally fulfilled as well. In the case of haptic rendering, a constant refresh rate of at least 1,000 Hz has to be guaranteed to create a realistic force feedback [MPT99]. The performance of triangle-based intersections is still too slow and unstable for this purpose [Pöt01]. For voxel sets, on the other hand, intersections can be computed very efficiently by using bitmap operations, thus fulfilling (2) even for the high requirements of haptic rendering. But a voxelized representation of a moving object has to be recomputed for each single position and orientation, and therefore fails for (1). As a solution, McNeely, Puterbaugh and Troy [MPT99] have proposed the Voxmap-PointShell technique (VPS), combining the high performance of voxel intersections with an efficient representation for dynamic objects. Thus, both criteria (1) and (2) are fulfilled



(a) Triangulation **(b)** Voxelization **(c)** Center points **(d)** Surface points

Figure 2.5: Computation of point shells.

for real-time haptic rendering of objects moving in a static environment.

As the basic idea of VPS, point shells representing the moving objects are checked for interference with a voxelized static environment. We compute the point shell for a moving object in four steps: first, a triangle mesh for the object surface is derived (cf. Figure 2.5(a)). Next, we voxelize the resulting mesh (Figure 2.5(b)) and get a first approximation of the point shell by the center points of all boundary voxels (Figure 2.5(c)). As an extension to the original algorithm, we further increase the accuracy of the point shell in a final step to generate a smoother surface representation as proposed in [RPP+01]: within each boundary voxel we interpolate the closest surface point to the voxel center (cf. Figure 2.5(d)). Finally, we obtain a set of accurate surface points which are uniformly distributed over the surface of the moving object. In addition, for each surface point normal vectors pointing to the interior of the object are computed. These normal vectors are required for the computation of the direction of repulsion. The set of all resulting surface points along with the normal vectors comprises the point shell. Its accuracy is determined by the resolution of the underlying voxel grid and the triangle mesh.

2.2 Fundamental Spatial Queries

In the previous section, we have shown how complex spatial objects, in particular three-dimensional CAD objects, can be transformed into manageable

entities. Now, in this section we will give a coarse overview of some operations on spatial data and show how spatial query methods can be supported.

2.2.1 Spatial Selection Queries

Spatial selection queries are of great importance as they build the fundament or at least are used to serve as preprocessing step for the evaluation of several spatial predicates. Even distance based operations like the distance range query or nearest neighbor query can profit from efficient spatial selection queries. Therefore, an efficient implementation of spatial selections is, in general, an important requirement for a good performance of spatial queries.

The spatial selection query retrieves a subset of the stored objects which in combination with the query object fulfil the spatial intersection predicate, i.e. which have a common intersection with the query object. We distinguish two types of spatial selection queries, the point query and the region query[1] (cf. Figure 2.6 and 2.7):

- **Point query:** Given a query point P and a set of objects D. The point query yields all the objects of D geometrically containing P.
- **Region query:** Given a region R and a set of objects D. The region query yields all the objects of D sharing points with R.

Region queries in one-dimensional spaces are also called interval queries (cf. Figure 2.6(b)). Obviously, the shape of the query region adapts to the shape of the query object. Since the shape of the query object can have an arbitrary complex structure, it has a great influence on the performance of the query. As a consequence, region queries in (two or higher)-dimensional spaces are preprocessed in a filter step using simple conservative approximations of the query object like rectangles or windows in two-dimensional space (cf. Figure 2.7(c)) or boxes in three-dimensional space.

[1] The point query can also be seen as a specialized region query having an extension equal to zero.

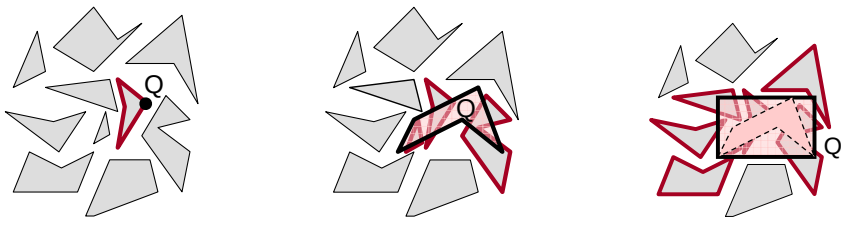
(a) One-dimensional point query

(b) One-dimensional region query

Figure 2.6: Examples of one-dimensional spatial selection queries.

2.2.2 Spatial Join Queries

One of the most common query types in Spatial Database Management Systems is the spatial join. In this thesis, we concentrate on the *intersection join*, as the intersection is the most important join predicate for complex spatial objects [GG98]. It retrieves all object pairs from two (or more) given data sets (sources) that satisfy the spatial-intersection predicate, i.e. all pairs of overlapping objects are reported as shown in Figure 2.8. A usual spatial join example of 2D geographical data is "find all cities which are crossed by a river". In the automobile industry, spatial join processing of complex 3D high-resolution objects is also required, e.g. to support efficient processing of queries like "find all engine parts which intersect the car body".

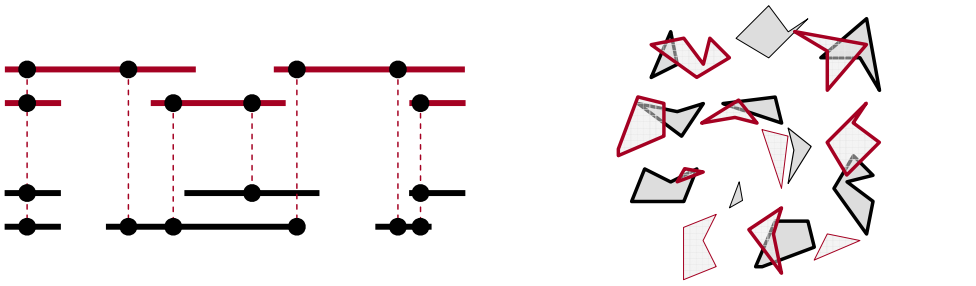


(a) Point query

(b) Region query

(c) Window query

Figure 2.7: Examples of two-dimensional spatial selection queries.



(a) One-dimensional intersection join **(b)** Two-dimensional intersection join

Figure 2.8: Spatial intersection join.

2.2.3 Spatial Indexing

An increasing number of applications require a fast access to those data objects in the database that occupy a given location in space, e.g. for spatial selection queries. In order to improve the performance of spatial queries, a spatial database system has to provide an effective and efficient query processor. Spatial indexing and multi-step query processing strategies (cf. Section 2.2.4) are the basic concepts of this fundamental system component [BKSS94][Güt94]. A spatial index structure or spatial access method, partitions the multidimensional search space for spatial queries. Particularly for spatial selection queries, they allow the query processor to quickly exclude large sets of irrelevant objects. Therefore, only a (generally small) subset of the database has to be considered to detect the actual query results. As the accurate representation of spatial objects can have arbitrary complexity, spatial index structures typically use conservative approximations to maintain their knowledge about the spatial shape and location of each object [Pöt01]. There are a lot of spatial index structures that facilitate accessing spatial data efficiently. A broad overview of spatial access methods is given in [GG97] and [AMW97]. In [GG97] spatial access methods are categorized into mapping methods, object bounding methods (non-replicating methods), clipping methods (replicating methods) and multiple layer methods.

Mapping methods simply map a d-dimensional extended object into a point in a two-dimensional space and use existing point access methods for

indexing these point objects. One alternative approach used by such methods is the decomposition of geometric objects into simple ones (e.g. rectangles) and sorting them by using space filling curves.

The most popular access method is the *object bounding method*. It decomposes the space in a hierarchical manner and stores the objects at the leaf level of the hierarchical structure. As nodes of the same level may overlap, a point query search may result in traversing multiple paths of the hierarchy. Most prominent instances of this class is the R-tree [Gut84] and the R^*-tree [BKSS90].

Clipping methods as well are based on a hierarchical organization of data space decompositions. In contrast to the bounding method which use only one entry per object (non-replicating method), this method uses clipping of objects and stores the resulting sub-objects in several nodes (replicating method). This method prevents overlapping of intermediate nodes at the same level in order to ensure that only one path of the hierarchical structure will be traversed for point queries.

Multiple layer methods, like the multi-layer grid file [SW88], partition the space more than one time and each partition is referred to as a layer which is organized in an hierarchical manner.

Although non-replicating index structures provide the minimal storage complexity, one-value approximations of the corresponding spatially extended objects often are far too coarse [Pöt01]. 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 such applications, the accuracy can be improved by decomposing spatial objects independently from the index partitions or, alternatively, by using a replicating index structure which is inherently tuned for redundancy. Due to the drawbacks of non-replicating index structures with complex spatial objects, the query processing methods presented in Part II of this thesis follow the replicating approach.

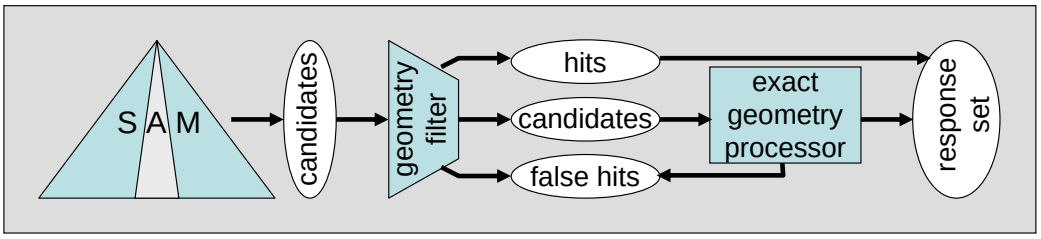


Figure 2.9: Multi-step query processing.

2.2.4 Multi-Step Query Processing

An efficient processing of spatial queries is provided by the multi-step procedure shown in Figure 2.9 [KBS93][BHKS93]. The main goal of the depicted spatial query processor is to reduce the expensive query evaluation step (exact geometry processor) by preprocessing operations (geometry filter) in preceding steps which reduce the number of objects investigated in the expensive refinement step. In the filter step, a superset of the objects qualifying for the spatial predicate is computed. This set is called candidates. A cascade of subsequent filter steps may further reduce the number of candidates, e.g. by the usage of more accurate conservative or progressive representations [BKS93a]. The geometry filter for selection queries as shown in Figure 2.7(b) could be, for example, an intersection evaluation on the minimal bounding rectangles (MBR) of the objects. Intersecting rectangles can be evaluated more efficiently than the exact object geometry and they are suitable object approximations used in common spatial access methods (SAM) like the R^*-tree [BKSS90]. For highly selective queries, the first filter step should be processed on a spatial access method (SAM). The multi-step query process is finished by the refinement step which checks the exact geometry of the candidates.

2.3 Industrial Applications of CAD Databases

In this section, we will illustrate the practical impact of the concepts which will be proposed in this thesis by means of realistic applications. Therefore

we require efficient query processing methods for complex spatial objects. Modern spatial database applications, as for instance virtual engineering, require a more fine-grained representation of spatial objects and special access methods for accurate spatial selection than it is required in standard GIS applications. In mechanical engineering, three-dimensional Computer Aided Design (CAD) is employed throughout the entire development process. From the early design phases to the final production of cars, airplanes or ships, thousands to millions of CAD files and many more associated documents including technical illustrations and business documents are generated. Most of this data comprises spatial product components or spatially related content. Recently, new CAD applications have emerged to support virtual engineering on this data, i.e. the evaluation of product characteristics without building even a single physical prototype. Typical applications include the digital mock-up (DMU) [BKP98] or haptic rendering of product configurations [MPT99].

If we look at CAD databases from a spatial point of view, each instance of a part occupies a specific region in the three-dimensional product space (cf. Figure 2.10). Together, all parts of a given product version and variant thereby represent a virtual prototype of the constructed geometry. Virtual engineering requires access to this 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.

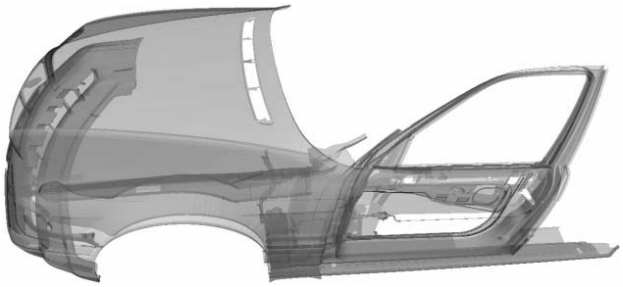
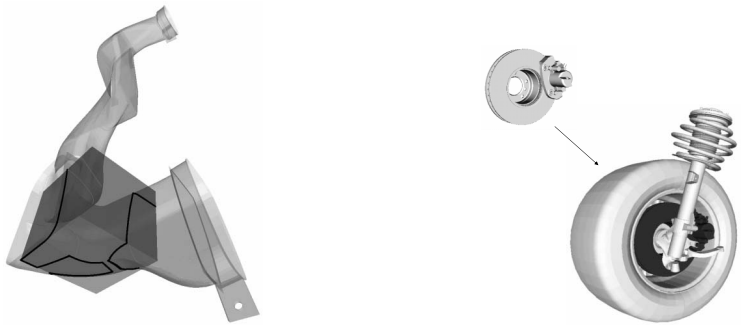


Figure 2.10: Virtual prototype of a car.



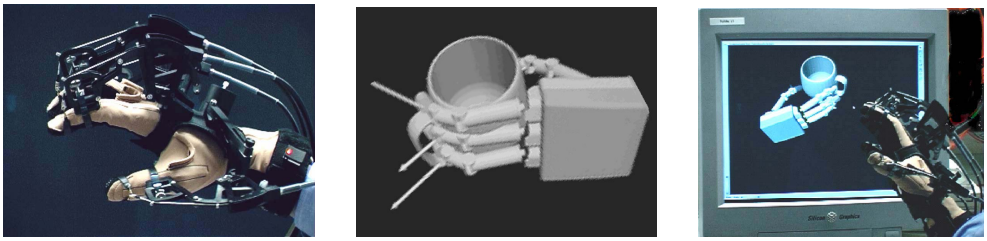
(a) Box volume query. **(b)** Collision query.

Figure 2.11: Spatial query on CAD data.

In the following, we present three industrial applications of virtual engineering which immediately benefit from efficient access methods on spatial CAD data. We have analyzed and evaluated them in cooperation with partners in the automotive and aerospace industry, including the Volkswagen AG, Wolfsburg, Germany, the German Aerospace Center DLR e.V., Oberpfaffenhofen, Germany and the Boeing Company, Seattle, USA.

2.3.1 Digital Mock-up of Prototypes

In the car industry, late engineering changes caused by problems with fit, appearance or shape of parts already 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, purely 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 are managed by a variety of users [BKP98]. 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



(a) Haptic device. **(b)** Virtual environment. **(c)** Haptic interaction.

Figure 2.12: Sample scenario for haptic rendering.

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 2.11 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 2.11(a)), can be delegated to an appropriate main memory-based CAD tool.

2.3.2 Haptic Rendering

The modern transition from the physical to the digital mock-up has exacerbated the well-known problem of simulating real-world engineering and maintenance tasks. Therefore, many approaches have been developed to

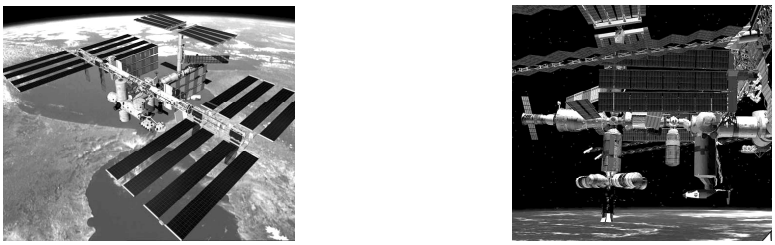


Figure 2.13: Virtual environment of the International Space Station.

emulate the physical constraints of natural surfaces, including the computation of force feedback, to capture the contact with virtual objects and to prevent parts and tools from interpenetrating [GLM96][LSW99][MPT99]. Figure 2.12(a) [Ren00] depicts a common haptic device to transfer the computed force feedback onto a data glove. The simulated environment along with the force vectors is visualized in Figure 2.12(b). By using this combination of haptic algorithms and hardware, a realistic force loop between the acting individual and the virtual scene can be achieved. Naturally, a real-time computation of haptic rendering requires the affected spatial objects to reside in main memory. In order to perform haptic simulations on a large scale environment comprising millions of parts, a careful selection and efficient prefetching of the spatially surrounding parts is indispensable. Figure 2.13 illustrates the complexity of usual virtual environments by the example of the International Space Station (ISS). In order to simulate and evaluate maintenance tasks, e.g. performed by autonomous robots, an index-based prefetching of persistent spatial objects can be coupled with real-time haptic rendering [Ren02].

2.3.3 Spatial Document Management

During the development, documentation, and maintenance of complex engineering products, many other files besides the geometric surfaces and solids of product components are generated and updated. Most of this data can also be referenced by spatial keys in the three-dimensional product space (cf. Figure 2.14), including kinematic envelopes which represent moving parts in

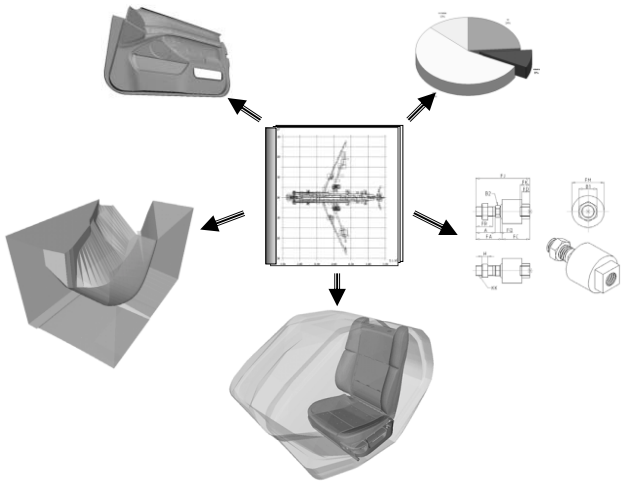


Figure 2.14: Spatial referencing of engineering documents.

any possible situation or spatial clearance constraints to reserve unoccupied regions, e.g. the minimal volume of passenger cabins or free space for air circulation around hot parts. Furthermore, technical illustrations, evaluation reports or even plain business data like cost accounting or sales reports for specific product components can be spatially referenced. Structurally referencing such documents can become very laborious. For example, the meeting minutes concerning the design of a specific detail of a product may affect many different components. Spatial referencing provides a solution by simply attaching the meeting minutes to a spatial key created for the region of interest. A very intuitive query could be: "*retrieve all meeting minutes of the previous month concerning the spatial region between parts A and B*". Such queries can be efficiently supported by spatial indexes.

Chapter 3

Complex Temporal Data

The continuously decreasing cost of storage capacity, the associated growth in the volumes of data being stored and the mounting recognition in the value of data varying with time has resulted in the prospect of organizing a large amount of complex structured data that search tasks can be performed in a very efficient way. Search problems in databases are rarely based on exact matches but rather on some application specific notion of similarity. In similarity search methods and mining techniques the time component provides us the ability to suggest cause and effect. This information is missed when the temporal component is ignored. In particular, temporal data mining is an important extension to mining static data as it has the capability of mining activity rather than just states and, thus, inferring relationships of contextual and temporal proximity, some of which may also indicate a cause-effect association. The consideration of the behavioral aspects in similarity search and mining tasks seems to be very promising for understanding "why" rather than merely "what".

3.1 Modeling Temporal Data

Temporal data denotes data which vary with time. Time varying data is essential to explain aspects of behavior associated with the implicit time-

varying nature of the universe.

In temporal databases, the objects to be managed are assigned to a (sequence of) time point(s) or time interval(s) expressing at which time a certain event on the object starts and ends. Instances of such kind of temporal data are sequences of time intervals. For example, records representing phone-call activities of an employee in a calling center.

In the following, we will give a short sketch about the data model generally used in temporal databases. For a deeper insight into the field of temporal databases we refer the interesting reader to the overview given in [JS99] and [EJS98].

3.1.1 Dimensions of Time

In the context of temporal databases two dimensions of time are of general interest, the *valid time* and the *transaction time*.

Valid time concerns the time at which a fact was *true* in reality. The valid time of an event is the time at which the event occurred in the real world, independent on the recording of that event in the database.

Transaction time concerns the range of time at which a fact was present in the database as stored data. The transaction time (usually a time interval) of a fact identifies the transaction that inserted the fact into the database and the transaction that removes the fact from the database.

3.1.2 Modeling Complex Temporal Data

Many applications associated with time varying data would suffer from simple temporal object information describing merely the event driven behavior of an object, just like when a certain event concerning an object starts and ends or the time frame at which an object appears and disappears in the dataset. In particular, record-keeping applications such as personnel- and medical-record management, and scientific applications such as weather monitoring

require a more detailed description of what happened in our *mini-world* during a specific range of time. Simple descriptions of the current state of the objects like "*active*" or "*inactive*" do not provide such information. Instead of qualitative information describing whether an object or event is active or not, assigning quantitative object-state information to the time attribute considered as an additional dimension of the object description widen the range of opportunities in temporal analysis. Time series (or time sequences) are the well-known type of temporal data providing the "rich" behavioral description of time varying objects.

3.1.3 What this Thesis is Not About

In this thesis, we do not consider classical temporal databases, for instance, databases with time stamps. As mentioned above, for a broad overview of classical temporal databases we refer the interesting reader to [JS99] and [EJS98].

Furthermore, in this thesis we will not consider classical methods for analyzing stationarity, trends, seasonally and autocorrelation in sequences. This topic is well discussed in [Cha03].

Rather, this thesis is about efficient and effective methods for measuring similarity between time-series objects stored in large time-series databases.

3.2 Time Series

Time series also known as *time sequences* are sequences, discrete or continuous, of quantitative data assigned to specific moments in time. They occur in virtually every medical, scientific and business domain and can be simple or complex, depending on the complexity of the data assigned to each time slot.

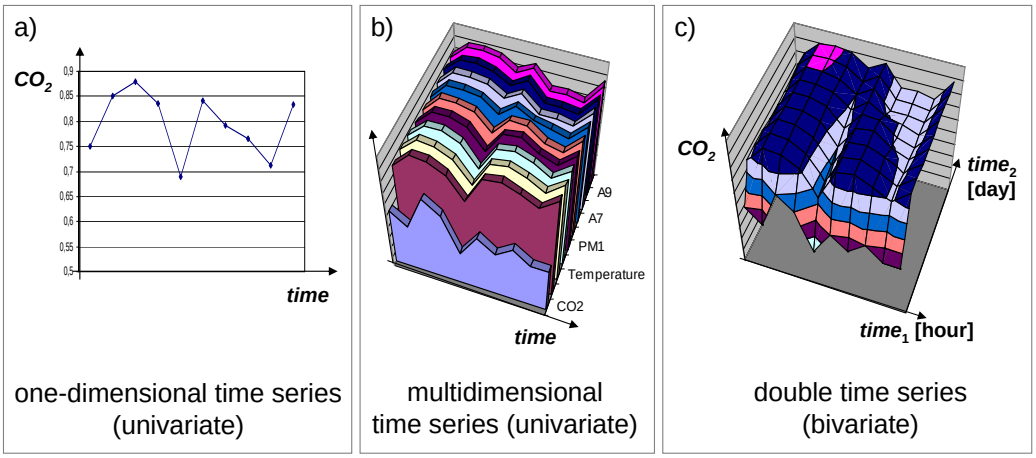
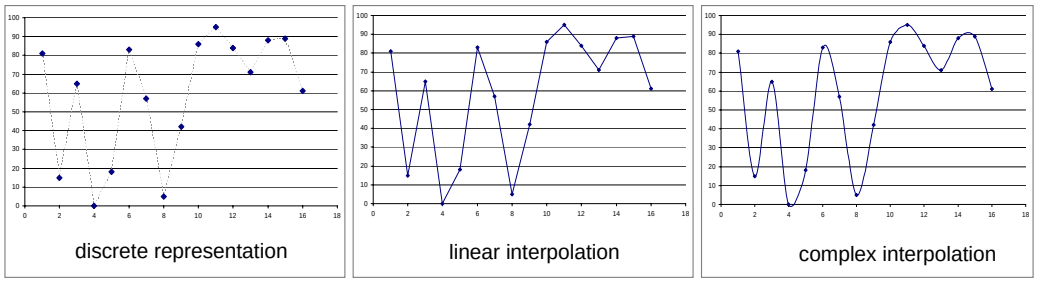


Figure 3.1: Different types of time series

3.2.1 Types of Time Series

(Univariate) *One-dimensional time series* assign a single value to each time slot. The daily averaged CO_2 concentration in Munich, Germany, is an example of a one-dimensional time series (cf. Figure 3.1(a)). If the data assigned to each time slot comprise several attributes (maybe of different domains), we call it multidimensional time series, one dimension for each attribute. An example of this type of time series is the time sequence of several meteorological and environmental attributes like temperature, CO_2 concentration and particulate matter PM_1 as depicted in Figure 3.1(b).

Furthermore, we can distinguish time series by the dimension of the used variable. In general, time sequences represent the variation over one single time attribute which we call *univariate* time series as the time series depicted in Figure 3.1(a) and 3.1(b). However, if we need to express the variation of one (or more) attribute(s) depending on multiple time variables, e.g. time t_1 = course of the day in steps of one hour and time t_2 = course of the year in steps of one day, we need *multi-variate* time series. In case of two time dimensions they are called *bivariate* time series. An example is depicted in Figure 3.1(c). Let us note that not all variables in multivariate time series necessarily need to be time dimensions. They can also be variables on other domains, e.g. spatial domains like the distance to a certain location.



time series X = < 81 ; 15 ; 65 ; 0 ; 18 ; 83 ; 57 ; 5 ; 42 ; 86 ; 95 ; 84 ; 71 ; 88 ; 89 ; 61 >

Figure 3.2: Interpolation of discrete time series

3.2.2 Interpolation of Discrete Time Series

Usually time series are discrete, i.e. they describe time varying attribute values by measurements made at discrete time slots. An example of a discrete representation of a time series is depicted in Figure 3.2 on the left hand side. The quality of the representation of the temporal variation of any phenomenon is often limited by the incompleteness of our observations. Representative observations of highly dynamical data are very difficult in practice. In particular, if the sampling rate of the observation (sensor) is much smaller than the variation frequency of the observed signal, i.e. the amplitude of the observed attributes can change significantly between two consecutive observations, the observation does not comply with the real signal. However, in general we can assume that the sampling rate of the observations is adjusted to the dynamic of the observed attribute in such a way that the frequency, phase, and amplitude of a signal cannot change significantly between the time slots of two subsequent measurements. In this case, the missing data can be interpolated with high fidelity. Due to its simplicity, the most popular method for time series interpolation is the *linear interpolation*. An example is depicted in Figure 3.2 (center). Depending on the already available knowledge of the dynamic behavior of the observed attribute, we can interpolate the missing data by more complex functions like polynomial functions of higher degree, as shown in Figure 3.2 (right).

3.3 Similarity Measures for Time Series

The similarity measure defined on time series objects is the basis of time series mining applications/tasks. Since we are dealing with subjective notions of similarity, the choice of an appropriate similarity measure depends on the user, the domain and the task at hand. We need the ability to handle this subjectivity. The similarity between time series can be categorized into three types of objectives [RKBL05]:

- similarity in time
- similarity in shape
- similarity in change

In the following, we will briefly give an understanding of these three types.

3.3.1 Similarity in Time

Similarity in time denotes the grade of correlation, i.e. time series which are correlated are defined to be similar in time. A common metric of similarity in time is the Euclidean distance $d_{eucl}(X, Y)$ between two time series $X = (x_1, .., x_N)$ and $Y = (y_1, .., y_N)$. The Euclidean distance is the most popular similarity measure for time series and formally, applied to time series, it 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 3.3. 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 w.r.t. the Euclidean distance measure. Comparing the two pairs of time series in our example, the Euclidean distance conforms with our intuitive perception of similarity, at least in this example.

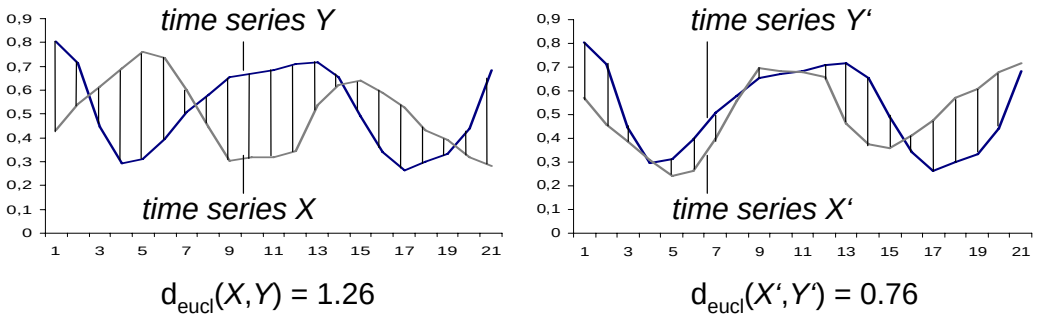
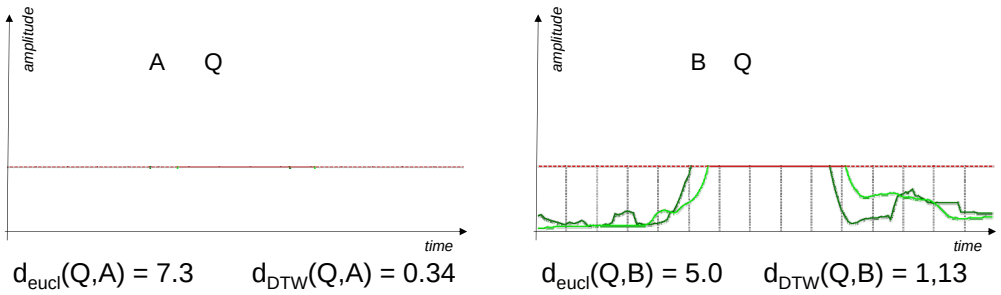


Figure 3.3: Euclidean distance between time series

Some times, instead of the Euclidean distance (L_2-norm) the Manhattan distance (L_1-norm) is applied to measure the (dis)similarity of time series. The difference between the Euclidean distance and the Manhattan distance in the context of measuring the similarity between time series is that the Euclidean distance more penalizes large amplitude differences at certain time slots.

3.3.2 Similarity in Shape

Similarity in shape denotes the grade of similar patterns, i.e. time series are defined to be similar in shape if they have similar patterns of change irrespective of time. The temporal synchronism of the amplitude response of two time series is less relevant than for similarity in time. Rather, this similarity measure focuses the shape characteristics of the time series independent of the time and duration of occurrence of significant patterns. Similarity in shape is invariant for temporal distortions or different amplitude scales which makes this similarity measure very important for many applications. For example, some patients have delayed responses on medical treatments, though the response characteristics are quite similar. The most common distance function which measures the similarity in shape is the dynamic time warping (DTW). A detailed description will be given in Section 3.3.4. In the example shown in Figure 3.4 we compare the similarity in shape with the similarity in time. We have given one reference time series Q, and two other time series A and B. Now we will consider the Euclidean distance d_{eucl} representing the



(a) Shape similar time series **(b)** Time similar time series

Figure 3.4: Comparison between similarity in shape and similarity in time.

similarity in time and the DTW-distance d_{DTW} representing the similarity in shape between Q and the other time series A and B respectively. Intuitively, the similarity in shape between A and Q (cf. Figure 3.4(a)) is higher than between B and Q (cf. Figure 3.4(b)). Contrary, the similarity in time between A and Q is smaller than between B and Q. In other words, Q is more shape similar to A and more time similar to B. Obviously, this is also reflected by the two distance functions d_{eucl} and d_{DTW}. Note that d_{eucl} and d_{DTW} are not directly comparable because they are different distance (similarity) functions. For this reason we took the relative similarity between the time series objects into account.

3.3.3 Similarity in Change

Similarity in change denotes the grade of similar autocorrelation. This similarity measure is important for trend analysis in time series databases. For example, in financial analysis applications a query could be to retrieve all stock prices in the year 2005 which show a positive trend from the 15^{th} July to the beginning of December and afterwards fall back again. Figure 3.5 depicts an example of three time series A, B and C showing the stock prizes over the period of one year. Time series A and B have similar autocorrelations which differ significantly from that of C.

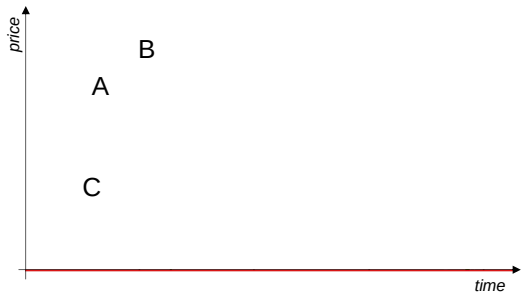


Figure 3.5: Example of stock prize time series

3.3.4 Time Warped Measures

For the determination of the similarity in shape the Euclidean distance function is too sensitive to minor distortions in the time axis. As mentioned above, Dynamic Time Warping (DTW) can fix this problem [BC94]. 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 in consideration of some constraints. The DTW distance can be computed by means of dynamic programming. Suppose there are given two time series $X =< x_1, x_2, ..., x_n >$ and $Y =< y_1, y_2, ..., y_m >$ of different length n and m, respectively. Let $d_{(i,j)}$ denote the distance between one value x_i of X and another value y_j of Y. The distance between two time series elements is defined as follows: $d_{(i,j)} = (x_i - y_j)^2$. In order to apply the DTW method, we define a $n \times m$-matrix where the (i^{th}, j^{th}) element of this matrix contains the distance $d_{(i,j)}$. A warping path W is a contiguous set of matrix elements that defines a mapping between X and Y. The k^{th} element of $W = \langle w_1, w_2, ..., w_k, ..., w_K \rangle$ is defined as $w_k = (i, j)_k$ and

$$max(m, n) \leq K < m + n - 1.$$

The warping path W is subject to the following three constraints: *boundary condition*, *continuity* and *monotonicity*.

- Boundary condition: $w_1 = (1, 1)$ and $w_K = (m, n)$, simply stated, that it requires the warping path to start and finish in diagonally opposite corner cells of the matrix.

- Continuity: given $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a - a' \leq 1$ and $b - b' \leq 1$. This restricts the allowable steps in the warping path to adjacent cells (including diagonally adjacent cells).
- Monotonicity: Given $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a - a' \geq 0$ and $b - b' \geq 0$. This forces the points in W to be monotonically spaced in time.

Out of the many warping paths that satisfy the above conditions, the dynamic time warping method detects the path which minimizes the warping cost:

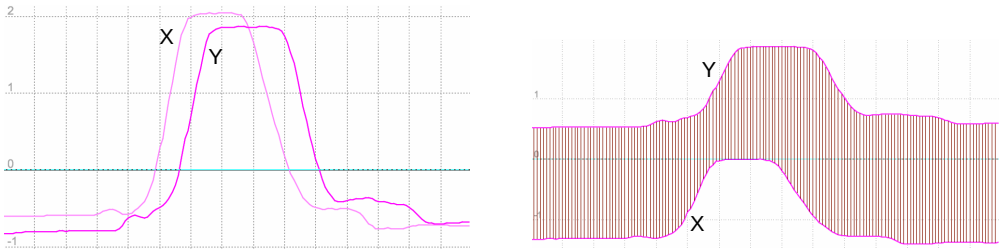
$$DTW(X, Y) = \min\{\frac{\sqrt{\sum_{w \in W} d_w}}{|W|}\},$$

where W is a correct warping path of X and Y, i.e. it is any warping path which fulfills the three warping conditions above.

The length of the path W, denoted as $|W|$, in the denominator is used to compensate for the fact that warping paths may have different lengths. This path can be found very efficiently by using dynamic programming to evaluate the following recurrence which defines the cumulative distance $\gamma(i, j)$ as the distance $d_{(i,j)}$ found in the current cell and the minimum of the cumulative distances of the adjacent elements:

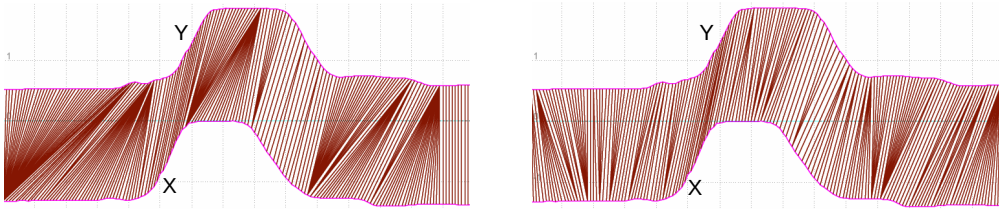
$$\gamma(i, j) = d_{(i,j)} + min\{\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1)\}.$$

In the example shown in Figure 3.6(c), we can see the alignment of the DTW distance between the two time series X and Y which are depicted in Figure 3.6(a). In order to achieve a better display of the alignment, we add an additional offset to the time series. The alignment shows which values of the time series are matched. For comparison, Figure 3.6(b) shows the alignment of the Euclidean distance between the two time series. We can see that the DTW distance is less sensitive to minor distortions in time, and thus, matches both time series more suitable than the Euclidean distance. However, the DTW distance has problems when the two sequences partially differ in their amplitudes, for instance, a valley in one time series may be deeper than the corresponding valley in the other time series. Then, DTW



(a) Original time series X and Y

(b) Alignment of Euclidean distance



(c) Alignment of DTW

(d) Alignment of DDTW

Figure 3.6: Alignment between two time series for different distance measures (Euclidean distance, DTW and DDTW).

attempts to explain the difference in terms of the time-axis and produces singularities, i.e. long sequences of asynchronous time warping, as shown in Figure 3.6(c).

Keogh et al. presented in [KP01] an enhanced distance function based on the DTW function called *Derivative Dynamic Time Warping* (DDTW). The DDTW tries to avoid singularities which commonly occurred with DTW. The DDTW alignment of our example is shown in Figure 3.6(d). However, singularities still occur due to other phenomenons like misleading discrepancies in the derivations of the time series. Nevertheless, dynamic time warping and variant approaches achieve often better results than the Minkowski metric based approaches, but the main problem of dynamic time warping techniques, their expensive computation, is still apparent. Only by approximating the time series with some compressed or down-sampled representation, the performance of DTW applied on the new representation can be improved drastically [KP99b].

3.3.5 Weighted Distance Measures

The intuition of weighted distance measures is that for some queries different parts of the sequence may have different importance for the analysis task. The weighted distance approach can be easily applied to the Minkowski metric based similarity measure. Assume we have two time series $X = (x_1, .., x_N)$ and $Y = (y_1, .., y_N)$ and the corresponding weights $w_i \in \mathrm{R}$ for each time slot i. Then, the weighted L_p-distance between X and Y is defined as follows:

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

In the example shown in Figure 3.7 the similarity between the two time series X and Y in the time sector 0 - 10 has more importance than the similarity in sector 20 - 30.

The problem is that the weights which lead to a good mining quality are often unknown in advance. One possibility to solve this problem is performing queries with relevance feedback. Relevance Feedback is the reformulation of a search query in response to feedback provided by the user for the results of previous versions of the query [KP99a]. The basic idea is to perform initially a query by choosing the same value for all weights and display the results to the user. Based on these results, the user can now change those weights which seem to be not promising for the mining task. Afterwards, the new query cycle with the new weights can be executed. This procedure can be repeated with the hope that the iterative weight updates will converge to the optimal query. An example approach using relevance feedback according to weighted distance measurement based analysis of time series is proposed in [KP99a].

3.4 Similarity Search Applications

In the medical, biological and environmental sector an immense amount of time series data are acquired for analysis tasks. As a consequence, efficient and effective data mining methods on time series data are very important.

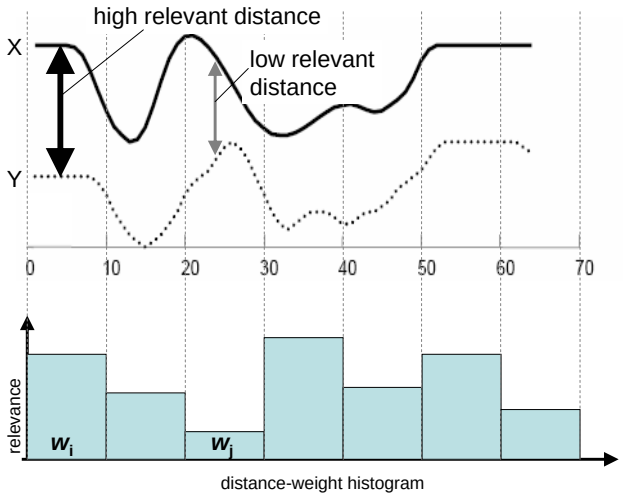


Figure 3.7: Weighted Similarity Measure

Similarity search in time series databases is a vital operation for many data mining applications, including *clustering*, *classification* and *mining of association rules*. As they are the most prominent mining methods, they will be briefly discussed in the following.

3.4.1 Clustering

Clustering determines which elements in a dataset are similar. It works to group records together according to an algorithm or mathematical formula that attempts to find centroids or centers around which similar records gravitate. It is the process of dividing a dataset into mutually exclusive subgroups without relying on predefined classes. The dataset is divided in such a way that similar records belong to the same subgroup and records of different subgroups are dissimilar. The clustering algorithms can be broadly classified into the following types [JMF99]:

Partition based clustering directly decomposes the data set into a set of disjoint clusters. More specifically, it determines an integer number of partitions that optimize a certain criterion function. The criterion function may emphasize the local or global structure of the data and its optimization

is an iterative procedure. Most prominent instances of partitional cluster methods are k-means [McQ67], PAM [KR90], and CLARANS [NH94].

Hierarchical clustering proceeds successively by either merging smaller clusters into larger ones or by splitting larger clusters. The result of the algorithm is a tree of clusters, called dendrogram, which shows how the clusters are related. By cutting the dendrogram at a desired level, a clustering of the data items into disjoint groups is obtained. Example approaches are Single Link (SL) [Sib73] and BIRCH [ZRL96].

Density-based clustering. The key idea of this type of clustering is to group neighboring objects of a dataset into clusters based on density conditions. A flat density-based cluster algorithm is DBSCAN [EKSX96] and a hierarchical density-based variant is OPTICS [ABKS99].

Distribution- or model-based clustering uses a distribution-based quality function. Each object is assumed to be drawn from one of k underlying Gaussian distributions [JD88]. Usually, objects are assigned to one of the k clusters using a maximum likelihood decision. Sample algorithms include the EM-algorithm [DLR77] and DBCLASD [XEKS98].

3.4.2 Classification

Classification is the systematic grouping of records into predefined categories which we call classes based upon shared characteristics. Classifications that are created non-empirically are called a priori classifications. Classifications that are created empirically by looking at the data are called a posterior classification.

3.4.3 Association Rule Mining

Association rule mining finds interesting associations and/or correlation relationships among large set of data items. Association rules show attribute value conditions that occur frequently together in a given dataset. They provide information of this type in the form of "if-then" statements. These

rules are computed from the data and, unlike the if-then rules from the logic, association rules are probabilistic in nature. In addition to the antecedent (the "if" part) and the consequent (the "then" part), an association rule has two numbers that express the degree of uncertainty about the rule. In association analysis the antecedent and consequent are sets of items (called itemsets) that are disjoint (do not have any items in common). Association rules are very important for many time series mining applications, in particular for environmental and medical data analysis because the association rules describe dependencies between the courses of different records. Especially association rules help to identify causations of anomalies in time series which could be an indication of a specific disease.

3.5 Indexing Time Series

Typically time series data occupy a large amount of memory space. Thus large databases require giga/tera-bytes of storage. As a consequence, we need a representation of the data we can efficiently manipulate and suitable access methods allowing efficient access to the data. In particular, similarity search methods require appropriate access methods. They should allow to efficiently determine the time series which are similar to a given query time series according to the specified similarity model.

3.5.1 Rules of Indexing Time Series

Faloutsos et al. [FRM94] give the following desirable properties for an indexing method for time series:

- Queries using the index structure should be faster than a sequential scan. The processing of each single time series will be to slow for large databases. The more time series objects can be pruned based on fast similarity approximations, the higher the performance gain of the index structure. Obviously, insertions and deletions of time series objects must not result in a complete rebuilding of the index structure.

- In order to achieve a correct result, no false dismissals must occur. This implies that similarity distance measures associated with the index structure have to fulfill the lower bounding property, or in other words, the estimated similarity distance between two time series has always to be less than or equal to the exact similarity distance between them.
- Furthermore, the index should incur little space overhead and should allow queries of various length in order to avoid poor applicability.

Keogh et al. [KCPM01] added the following criteria to the list above:

- The index structure should be possible to build the index in reasonable time and it should preferably be able to handle more than one distance measure.

3.5.2 Vector Space Transformation

The common approach for indexing time series is to project time series of length n into n-dimensional space R^n. In this representation the dissimilarity between two time series objects are represented by the distance[1] between them, i.e. a small distance between two points in the feature space indicates a high similarity of the corresponding time series objects. The advantage of doing this is that we have abstracted away the details of "time series" that all query processing can be imagined as finding points in space. Now, common spatial access methods like the R-tree can be used to organize the transformed time series objects efficiently.

3.5.3 Curse of Dimensionality

Naturally, the dimensionality of the vector space which is used to bring the time series objects in a spatial context according to the above transformation is very high. A general property of high-dimensional feature spaces is

[1] Here, we assume any Minkowski metric as similarity measure.

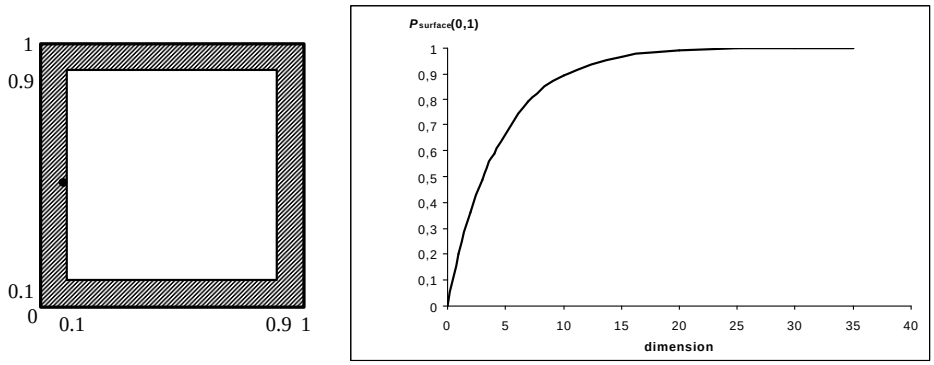


Figure 3.8: Probability of a point near by the data space boundary.

that they have an impact on the performance of similarity search algorithms. These phenomena are usually summed up by the term "*curse of dimensionality*".

The curse of dimensionality refers to the exponential growth of hypervolume as a function of dimensionality [Bel61]. It can be explained in the following way: let us assume a uniform distribution of the data points inside a hypercube with side length 1, i.e. $\mathcal{D} \subseteq [0,1]^d$ (cf. Figure 3.8 left). The volume of such a data space is $1^d = 1$. The probability $P_{\text{surface}}(r)$ that a point randomly taken from a uniform and independent distribution in a d-dimensional space has at most a distance of r to the space boundary can be computed as follows:

$$P_{\text{surface}}(r) = 1 - (1 - 2 \cdot r)^d.$$

As it is shown in Figure 3.8, the probability that a point is inside a 10% border of the data boundary rapidly increases with growing dimensionality. For $d = 3$ dimensions, $P_{\text{surface}}(0.1)$ is already 0.488% and reaches 0.965% for $d = 15$ dimensions.

This observation has great influence on the performance of the similarity search algorithms and, in particular, cause problems for index supported similarity queries. The performance of R-Trees degrades exponentially with the number of dimensions. Somewhere above 6 - 20 dimensions, the R-Tree degrades to linear scanning. However, usually the time series have a length

of hundreds or perhaps thousands. In order to cope large time series with the feature vector based indexing method, we have to reduce the dimensionality of the time series objects artificially.

3.6 GEMINI: A Generic Indexing Approach for Large Time Series

In general, a time series of length n corresponds to a n-dimensional feature vector. The (dis)similarity between two time series is usually measured by an appropriate distance function in this feature space, e.g. the Euclidean distance, Dynamic Time Warping (DTW) [KP99b], Pearson's correlation coefficient [Coh88], or angular separation also known as cosine distance which is measured in terms of the cosine value of the angle between two vectors. Recent approaches either focus on an entire matching of the query time series with the database objects or on subsequence matching.

Entire matching approaches compute the similarity between the query time series and the database time series by considering the entire time course using any of the above mentioned distance measures. Since the length n of the time series objects is usually very large, the analysis of time series data based on the entire time series information is usually very limited. Due to the *curse of dimensionality*, the efficiency of data analysis methods decrease rapidly with increasing time series length. Thus, it is mandatory to find more suitable representations of time series data, e.g. by reducing the dimensionality. The key idea is that such a suitable representation satisfies the lower bounding property, i.e. the distance between two time series based on the compact representation is always lower or equal to the true distance of the two time series. In [FRM94] the GEMINI method is introduced. It can exploit any dimensionality reduction method to allow efficient indexing of time series based on the lower bounding property. The general idea of this 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 $\mathrm{R}^{n'}$ ($n' << n$) which can be efficiently handled by any spatial access method

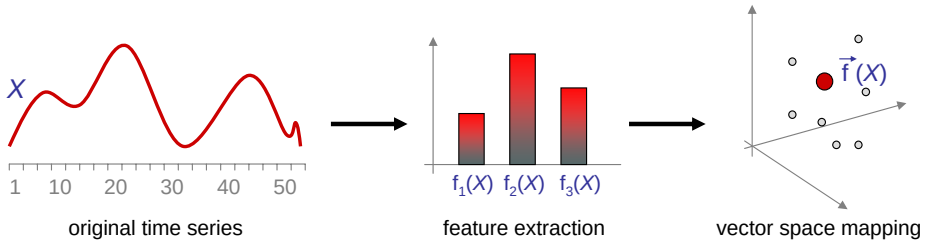


Figure 3.9: Feature based dimensionality reduction (GEMINI approach).

of our choice. The transformation chain of GEMINI is depicted in Figure 3.9.

Given the lower bounding property, one can safely drop a subset of database objects based on the evaluation of the distance applied to the reduced representation. This condition is absolutely necessary in order to avoid false dismissals, i.e. no true result are lost by performing the query on the dimensionality reduced feature space. This pre-selection based on the compressed representation is also called filter step. The remaining objects need to be refined, i.e. the distance on the exact representation needs to be computed. In [KSF+96], the GEMINI framework is adapted for k-nearest neighbor search. Later, Seidl and Kriegel proposed in [SK98] an improved k-nearest neighbor search algorithm which minimizes the number of candidates to be considered. In fact, the authors propose to use several filter steps in order to reduce the number of candidates that need to be refined. Several dimensionality reduction techniques that lower bound the Euclidean distance have been successfully applied to similarity search in time series databases, e.g. [AFS93, WFS04, CF99, YF00, KJF97, ABB03, KCMP01, CN04] (cf. Section 3.7).

Subsequence matching approaches usually try to match a query subsequence to subsequences of the database objects. A time series object o is similar to the query subsequence q as long as o has a subsequence similar to q. Usually, a subsequence matching problem is transferred into an entire matching problem by moving a sliding window (window size is set to the size of the corresponding subsequence) over each time series object in the database and materializing the corresponding subsequence. Thereby, it is assumed, that

the length of the query subsequence is fixed. If this length changes, a corresponding sliding window has to be moved over each database time series again. Obviously, subsequence matching is orthogonal to interval-focused similarity. In interval-focused similarity, the time slot relevant for matching is fixed. Two time series are not considered similar even if they have a similar subsequence but at different time intervals. In addition, the concept of interval-focused similarity allows to specify multiple relevant time intervals of different length. Both, the number of relevant intervals and the length of these intervals may change from query to query.

3.7 Time Series Representations

In the literature, there are a lot of different approaches concerning efficient similarity search algorithms for time series. Most of them refer to dimensionality reduction. The proposed approaches mainly differ in the representation of the time series. Figure 3.10 gives an overview of the most important techniques, including DFT [AFS93] and extensions [WFS04], DWT [CF99], PAA [YF00], SVD [KJF97, ABB03], APCA [KCMP01], Chebyshev Polynomials [CN04]. Keogh et al. [RKBL05] has grouped them into the categories: *Model Based*, *Data Adaptive*, *Non Data Adaptive* and *Data Dictated*.

These categories can be further split into several subcategories. A survey is given in [KCMP01]. We have added one representation (*Threshold-Crossing time Intervals*) into the category *Data Dictated* as a novel approach proposed in this thesis (cf. Chapter 13). This new representation is silhouetted against the other techniques by the used similarity model. While the traditional techniques (partially lower bounding) approximate Minkowski metrics and DTW by aggregating over the time axis, we propose to aggregate time series over the amplitude axis. In contrast to the other approaches our time series representation allows us to focus the similarity search to certain maybe more relevant amplitude values.

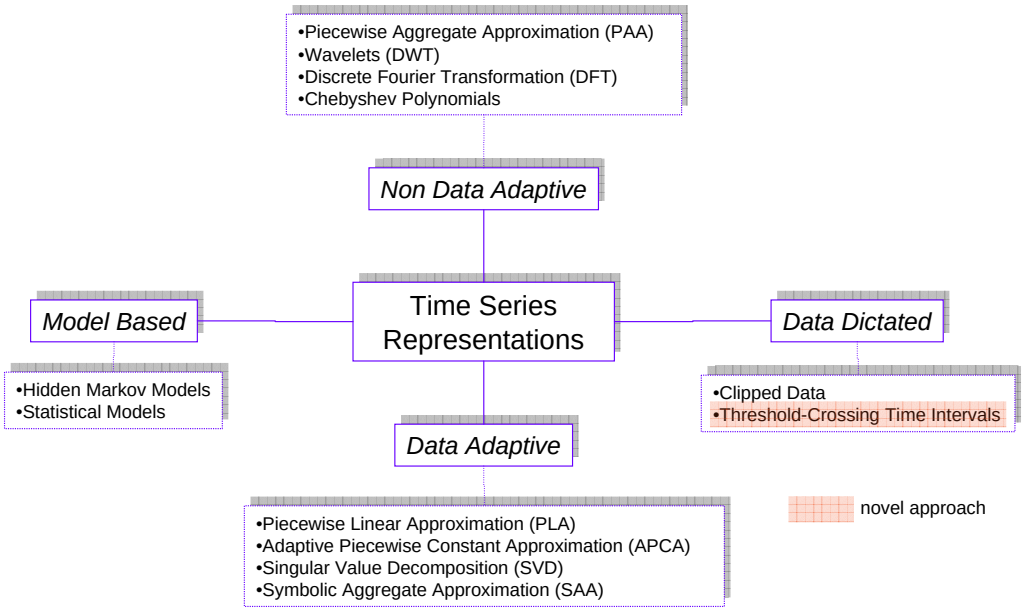


Figure 3.10: Classification of all (relevant) time series representations proposed for data mining.

Chapter 4

Intervals and Interval Sequences

One-dimensional intervals or interval sequences are a very suitable data structure for representing temporal and spatial data [Pöt01]. Originally, interval sequences are used to handle finite domain constraints [Ram97] or represent periods on transactions or valid time dimensions [TCG+93]. Recently, they are used to represent complex-structured objects, in particular those objects having a temporal or even spatial component (cf. Figure 4.1). For instance, rasterized three-dimensional objects can be transformed into interval sequences by means of space-filling curves (cf. Section 2.1.3). Furthermore, any time series may be aggregated to an interval sequence, such as periods of "high" stock prices for technical chart analysis. As mentioned in Section 3.7, this type of representation is required for our novel similarity search approach based on threshold-crossing time intervals which is proposed in Part III of this thesis. Another typical application of one-dimensional interval sequences includes the temporal tracing of user activity for service providers. The advantage of intervals or interval sequences is, that they are easy to manage and can be organized in an efficient way by means of adequate access methods.

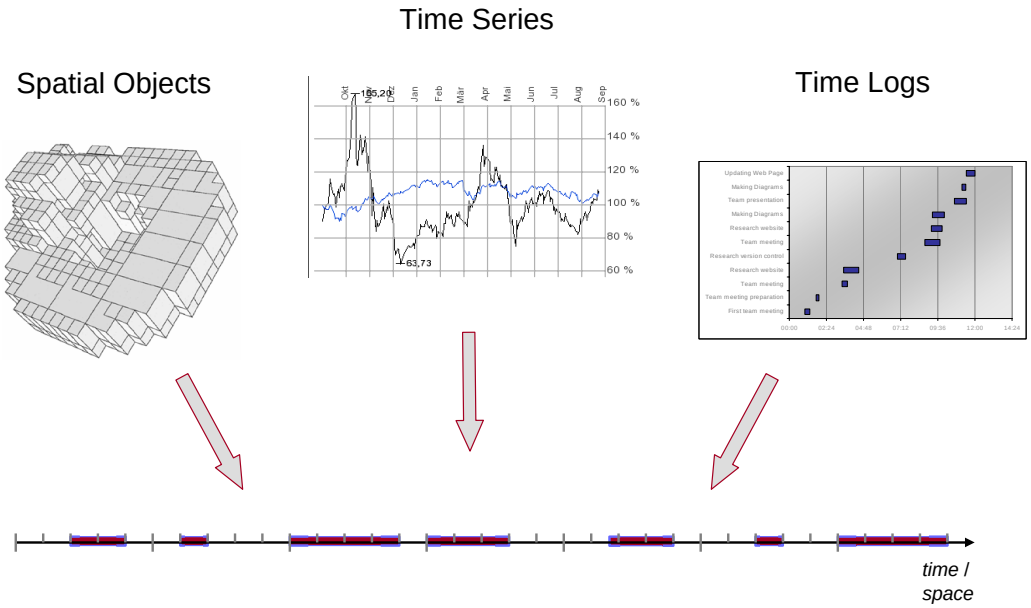


Figure 4.1: Applications of one-dimensional interval sequences

4.1 Applications on Interval Data

Intervals occur as transaction time and valid time ranges in temporal databases [SOL94] [Ram97] [BÖ98] or as line segments on a space-filling curve in spatial applications [FR89] [BKK99] (cf. Section 2.1.3).

4.1.1 Interference Checks for Spatial Data

Spatially extended objects, mainly objects of geographical information systems or objects of CAD-systems, usually have very complex and intricate shapes. Consequently, even simple spatial operations like interference checks performed on the original data could be very costly. Approximating the objects by a sequence of intervals reduces the interference check problem to simple and cheap interval intersection operations. Based on this approach, in this thesis we present solutions for redundancy problems that occur especially for high resolution spatial data (cf. Part II).

4.1.2 Data Mining in Time Series Databases

Another important application of interval sequences is abstracting time series for supporting a higher level of understanding dynamic processes. They are sometimes used for the retrieval of similar time series [KP99b] [KCPM01], and more often applied in the medical domain [KF00] [GU99] [BLMB02]. Especially in the medical domain expert knowledge is quite often necessary in early stages to fix thresholds and make design decisions. Temporal description of events in time series, e.g. interval time series, are of vital importance for data mining tasks, for instance in mining rules [VHTM99]. Mining such relationships allows the user to gain insight on the temporal relationships among various items. In this thesis we propose a novel similarity measure for time series data based on aggregating time series by interval sequences of threshold events and show how these representations can be organized to speed-up query processing and data mining tasks (cf. Part III).

4.2 Definition

Generally, in mathematics different types of intervals are distinguished. In the real number space R intervals are closed, open or half open. As simple closed intervals suffices for our approaches, in this thesis, we define intervals as follows:

Definition 4.1 (Interval) *Let* $\mathrm{B} \subseteq \mathrm{R}$ *be a domain of boundary points. A tuple* $t = (l, u) \in B^2$ *is called interval, iff* $l \leq u$*. It represents all elements* $x \in B$*, where* $l \leq x \leq u$*. The components* l *and* u *are the lower and upper bound of* t*, respectively. The interval* t *is called degenerate or point, iff* $l = u$*.*

Now, we can define an interval sequence as a sequence of disjunctive intervals.

Definition 4.2 (Interval Sequence) *Let* B^2 *be a domain of intervals. A sequence* $S =< b_1, b_2, .., b_n >$ *of intervals* $b_1 = (l_1, u_1)$*,* $b_2 = (l_2, u_2)$*, ... ,* $b_n = (l_n, u_n) \in B^2$ *is called* interval sequence*, iff* $\forall i \in \{1, .., n-1\} : u_i < l_{i+1}$*.*

4.3 Basic Operations on Intervals

As we have seen in the previous section, interval sequences are disposed to represent spatially or temporally extended objects[1]. Consequently, there is a practical need for a clear and proper treatment of various useful operations on interval data in the context of spatial- and temporal data.

4.3.1 Predicates on Intervals

The most popular predicate used in retrieval applications dealing with interval data is the *intersection* predicate.

Definition 4.3 (Interval Intersection) *Let* $D = \{(l, u) \in \mathrm{B}^2 | l \leq u\}$ *be a domain of intervals with boundaries of* $\mathrm{B} \subseteq \mathrm{R}$. *In the following, we say that two intervals* $s_1 = (l_1, u_1)$, $s_2 = (l_2, u_2)$ intersect *(i.e.* $intersect(s_1, s_2) = true$ *or, alternatively,* s_1 intersects s_2*), iff* $(l_1 \leq u_2) \wedge (l_2 \leq u_1)$.

A more general predicate definition is given in Allen's temporal interval logic [All83]. It describes relations between temporal intervals and is often applied for spatial and temporal reasoning and data-mining tasks [Höp01]. For any pair of intervals we have 13 possible relationships as illustrated in Figure 4.2. For instance, we say "A *meets* B" if interval A terminates at the same point in time at which B starts. The inverse relationship is "B *is-met-by* A". The *intersect* predicate can now be formulated by $\neg(A \text{ after } B) \wedge \neg(B \text{ after } A)$.

Allen's algebra gives qualitative information about the relationship between two intervals. However, qualitative information like "A *before* B" may not necessarily suffice for some applications. For example, let the intervals A and B describe the occurrence of two events which may be dependent of each other. Then, if we know that the event denoted by B follows the

[1] We consider time as an additional dimension of the object representation. Since time series cover several consecutive time slots, they have an extension in the time dimension and we call them temporal extended objects.

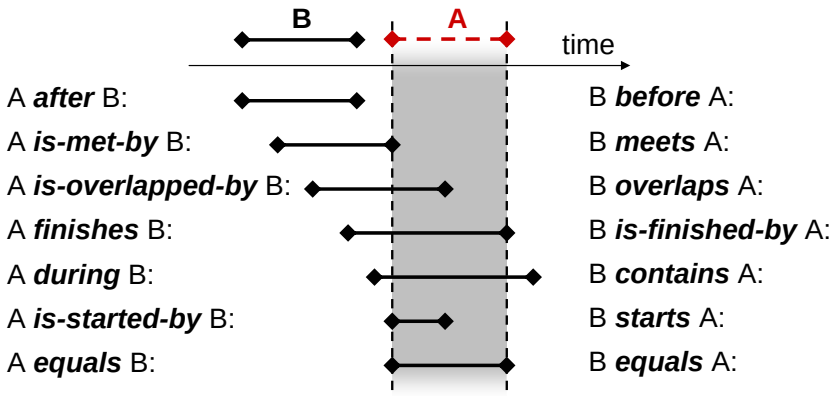


Figure 4.2: Allen's interval relationsships.

event denoted by A, i.e. "A *before* B", the temporal distance between the occurrences of both events could be very valuable for the analysis of their temporal dependency.

4.3.2 Functions on Intervals

Due to the lack of quantitative information associated with predicates on intervals, we will now define some useful functions on intervals providing more information about the relationship between two intervals. At first, we define the function *intersection_length* which computes the length of the intersection between two intervals.

Definition 4.4 (Intersection Length) *Let* $D = \{(l, u) \in \mathrm{B}^2 | l \leq u\}$ *be a domain of intervals with boundaries of* $\mathrm{B} \subseteq \mathrm{R}$. *The length of the intersection between two intervals* $s_1 = (l_1, u_1)$, $s_2 = (l_2, u_2)$ *is computed by the function* $intersection\_length : D \times D \rightarrow \mathrm{R}_0^+$ *as follows:*

$$intersection\_length(s_1, s_2) = \begin{cases} min(u_1, u_2) - max(l_1, l_2), \text{ if } intersect(s_1, s_2); \\ 0, \text{ otherwise}; \end{cases}$$

The *intersection_length* routine can be very vital for applications which require to return intersection-query results in descending order of the intersection length, i.e. for intersection ranking queries.

Beside the intersection based functions on intervals, distance functions reflecting the "similarity" between intervals are also important. The most important similarity function for intervals is the *Euclidean distance based on endpoints*, taking for two given time intervals the difference between the lower values and that between the upper values into account.

Definition 4.5 (Endpoint Based Euclidean Distance) *Let* $D = \{(l, u) \in \mathrm{B}^2 | l \leq u\}$ *be a domain of intervals with boundaries of* $\mathrm{B} \subseteq \mathrm{R}$. *The Endpoint Based Euclidean distance* d_{eucl} *between two intervals* $s_1 = (l_1, u_1)$, $s_2 = (l_2, u_2)$ *is defined as follows:*

$$d_{eucl}(s_1, s_2) = \sqrt{(l_1 - l_2)^2 + (u_1 - u_2)^2}.$$

A detailed discussion on different similarity distance measures for intervals is given in chapter 12.

4.4 Efficient Management of Intervals and Interval Sequences

A variety of methods has been published concerning interval management in databases, most of them addressing temporal applications. In the following we give a short summary of the survey on interval handling given in [KPS00a]. For in-depth descriptions of some of the specialized techniques, including append-only structures for transaction time intervals, we refer the reader to the survey of Tansel et. al. [TCG+93].

In the context of computational geometry, several main memory based data structures that support 1D interval data have been developed [PS93] [Sam90]. Among them, the Segment Tree of Bentley, the Priority Search Tree of McCreight and the Interval-tree of Edelsbrunner are the most popular. More recent developments include the Interval Skip List and the IBS-Tree of Hanson et al. [HJ96]. As major limitation, the main memory resident data structures do not meet the characteristics of secondary storage. In a

disk-oriented context, access is block-oriented and only small portions of a structure may reside in main memory at a given point of time. The concept of Segment Indexes [KS91] is a way to overcome the problem by combining optimal interval structures with efficient disk-oriented indexing techniques.

A variety of block-oriented access methods for intervals has been presented in the literature [TCG+93] [MTT00].

The *Time Index* of Elmasri, Wuu and Kim [EWK90] is an index structure for valid time intervals. A set of linearly ordered indexing points is maintained by using a B^+-tree, and for each point, a bucket of pointers refers to the associated set of intervals. The *Interval B-tree* (IB-tree) of Ang and Tan [AT95] can be regarded as an implementation of Edelsbrunner's Interval-tree [Ede80] using an augmented B^+-tree rather than a binary tree. The original main memory model is thus transformed to an efficient secondary storage structure while preserving the optimal space and time complexity. Nevertheless, the complex three-fold structure of the Interval-tree is retained, and a dedicated structure of its own is used for each level. The *Interval B^+-tree* (IB^+-tree) of Bozkaya and Özsoyoglu [BÖ98] is a secondary storage model of the Interval-tree of [CLR90] that differs from Edelsbrunner's Interval-tree by the fact that it utilizes the lower bounds of the intervals as primary keys. As a result, predicates referring to the upper bounds of intervals such as "*meets*" or "*after*" are not supported well. The *TP-Index* of Shen, Ooi and Lu [SOL94] is based on a transformation of intervals into a triangular 2D space. Duplicates are avoided and the index is well suited for appending intervals since the data space may grow dynamically at the upper bound. A similar mapping organized by a grid file is presented by Lee and Tseng [LT98]. The *External Segment Tree* of Blankenagel and Güting [BG94] also provides a block-based storage of intervals. As in the main memory segment tree, intervals are decomposed into segments and referenced by a skeleton structure. Similarly, the External Memory Interval-tree of Arge and Vitter [AV96] is a stand-alone externalization of Edelsbrunner's Interval-tree. The fan-out of the original backbone tree is increased from 2 to $\sqrt{b}$ for disk blocks of size b.

Beside originally one-dimensional interval index structures even multi-dimensional index structures can be employed for the task of managing 1D intervals. In general, however, spatial access methods such as Guttman's *R-tree* [Gut84] and its variants including R^+*-tree* [SRF87] and R^**-tree* [BKSS90] may not behave well for one-dimensional intervals. Particularly the long durations and high overlaps of intervals in many temporal applications induce severe performance problems [EWK90] [GLOT96].

There are methods which use B^+-tree index structures to organize intervals rather than to augment indexes or to introduce new structures. Following the paradigm of relational indexing [Pöt01] they can be easily integrated into an existing Relational Database Management System (RDBMS). The positional Interval-Spatial Transformation (IST) of Goh et al. [GLOT96] is based on encoding intervals by space-filling curves called D-, V- and H-ordering that map the boundary points into a linear space. Recently, the Relational Interval-tree (RI-tree) [KPS00b] has been proposed which is a relational implementation of Edelsbrunner's Interval-tree. As we use this concept to organize the intervals for managing spatial data as proposed in [KPS00a], we shortly review the general concept of the RI-tree in the following.

4.4.1 Relational Interval-tree

The Relational Interval-tree (RI-tree) is an access method for intervals which is very suitable for interval-collision queries. Since we used RI-tree in our approaches, we want to give a short overview on this access method.

The conceptual structure of the RI-tree is based on a virtual binary tree of height h which acts as a backbone over the range $[0...2h-1]$ of potential interval bounds. Traversals are performed purely arithmetically by starting at the root value $2h$ and proceeding in positive or negative steps of decreasing length $2h-i$, thus reaching any desired value of the data space in $O(h)$ time. This backbone structure is not materialized, and only the root value $2h$ is stored persistently in a meta-file. For the storage of intervals, the nodes of

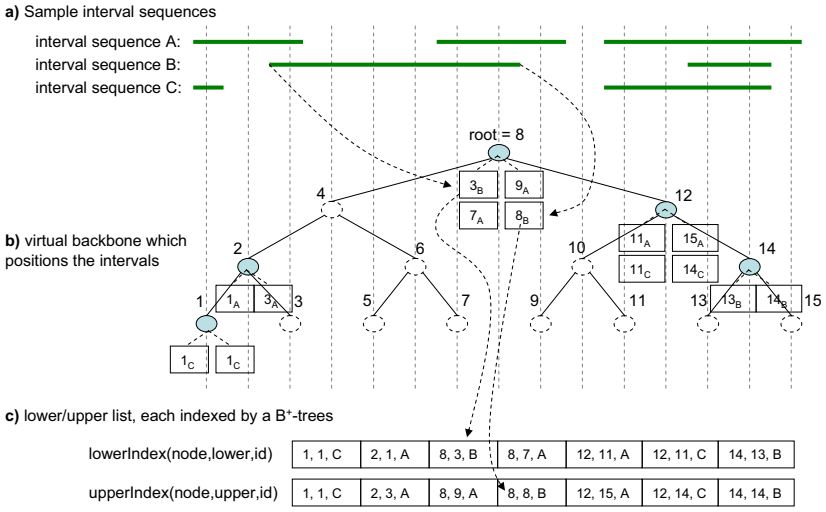


Figure 4.3: Block-based Relational Interval Tree

the tree are used as artificial key values: each interval is assigned to a fork node which is the first intersected node when descending the tree from the root node down to the interval location.

An instance of the RI-tree consists of two sorted page lists lowerIndex(*node*, *lower*, *id*) and upperIndex(*node*, *upper*, *id*), each organized in a B^+-tree. The lists store the artificial fork node value *node*, the bounds *lower* and *upper* and the *id* of each interval or interval sequence. Thereby, the entries are primary sorted in ascending order of the attribute *node*. They are secondary sorted by the attributes *lower* and *upper*, whereas *lower* is in ascending order and *upper* is in descending order. Any interval is represented by exactly one entry in each of the two B^+-trees and, thus, $O(n/b)$ disk blocks of size b suffice to store n intervals. For inserting or deleting intervals, the *node* values are determined arithmetically, and updating the indexes requires $O(log_b n)$ I/O operations per interval. We store an interval sequence by simply labeling each associated interval with the sequence identifier. Figure 4.3 illustrates the Relational Interval-tree by an example.

Now, we will consider simple interval intersection queries (*lower*, *upper*) which are processed in two steps. In the preparation step, range queries are collected in two lists, $leftNodes$ and $rightNodes$ which are obtained in the

following way: by a purely arithmetic traversal of the virtual backbone from the root node down to lower and to upper, respectively, at most $2 \cdot h$ different nodes are visited. Nodes left of *lower* are collected in $leftNodes$ since they may contain intervals who overlap *lower*. Analogously, nodes right of *upper* are collected in $rightNodes$ since their intervals may contain the value of *upper*. As a third class of affected nodes, the intervals registered at nodes between *lower* and *upper* are guaranteed to overlap the query and, therefore, are reported without any further comparison by a so-called *inner query*. The query preprocessing procedure is purely main memory-based and, thus, requires no I/O operations. In the second step, all three node lists are joined with the B^+-tree indexes $upperIndex$ and $lowerIndex$. The upper bound of each interval registered at nodes in $leftNodes$ is checked against *lower* (*left queries*), and the lower bounds of the intervals from $rightNodes$ are checked against *upper* (*right queries*). The *inner query* corresponds to a simple range scan over the nodes within (*lower*, *upper*) and, thereby, can be applied to $lowerIndex$ or $upperIndex$. The query requires $O(h \cdot log_b n + r/b)$ I/Os to report r results from an RI-tree of height h. The height h of the backbone tree depends on the expansion and resolution of the data space, but is independent of the number n of intervals. An example query for one query interval is depicted in Figure 4.4. A right directed arrow for a node denotes that the upper bounds of the intervals of this node has to be considered, whereas a left directed arrow denotes that the lower bounds of the intervals has to be taken into account. The example query is split into the left queries combined with the inner query 2,3,4-5 which are applied to the $upperIndex$ and the right queries 6,7,8 which are applied to the $lowerIndex$. The naive approach disregards the important fact that the intervals of an interval sequence represent the same object. As a major disadvantage, many overlapping queries are generated. This redundancy causes an unnecessary high main memory footprint for the transient query tables, an overhead of query time, and lots of duplicates in the result set that have to be eliminated. In our approach we follow the basic idea in [KPS01] which avoids the generation of redundant queries rather than to discard the respective queries after their generation.

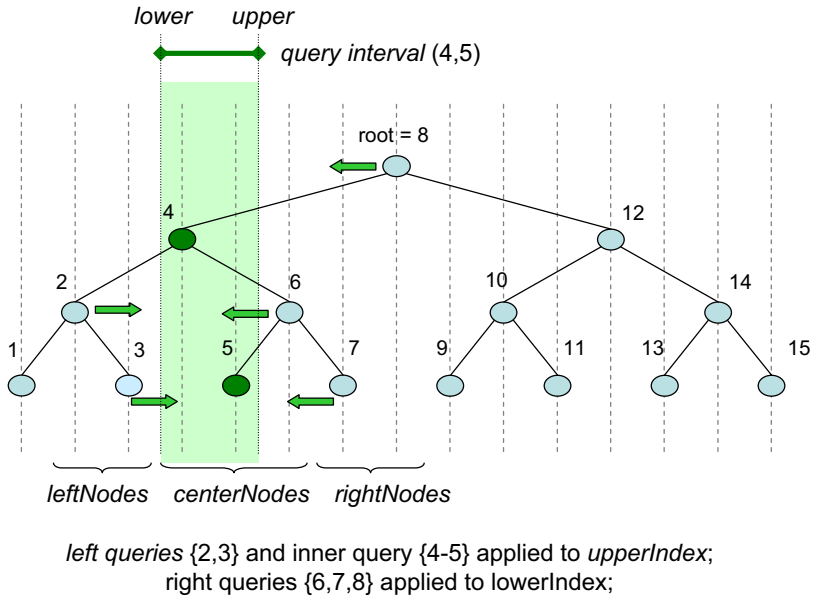


Figure 4.4: Interval Query onto the Relational Interval-tree

4.5 Statistics on Intervals

Statistics about the current distribution of the indexed data is a very important component to estimate the selectivity of queries, and thus, is a required input for query optimization. The selectivity of a predicate is used by the query optimizer to determine an efficient execution plan. In order to achieve a good estimation for the selectivity of a specific predicate without retrieving the actual results, the predicate has to be evaluated on an accurate approximation of the data distribution. Beside *parametric techniques* which approximate the given data by using a standard mathematical distribution and *sampling* which adapts to the actual data distribution by processing a small fraction of the stored tuples, *statistics* are a very popular approach in database systems, as they typically can be efficiently computed and occupy only a small amount of secondary storage [Pöt01]. Furthermore, statistics do not require a priori assumptions about the data distribution.

In this section, we will present two approaches for statistics, the *interval histogram* and the *quantile vector*, both reflecting the current distribution of

interval data. Both statistics are specially designed for linearly ordered data. We use them in our approaches in order to generate appropriate interval approximations (cf. Part II).

4.5.1 Interval Histogram

Histograms are a very popular method to capture the distribution of spatially or temporally extended data, in particular intervals, at any desired resolution.

Definition 4.6 (Interval Histograms) *Let $D = [1, 2^h - 1]$ be a domain of interval bounds $h \geq 1$. Let the natural number $\nu \in \mathrm{N}$ be the* resolution*, and $\beta_\nu = (2^h - 1)/\nu$ the corresponding* bucket size*. Let $b_{i,\nu} = [1+(i-1)\cdot\beta_\nu, 1+i\cdot\beta_\nu]$ denote the span of bucket i, $i \in \{1, .., \nu\}$. Let further $I = \{(l, u), l \leq u\} \subset D^2$ be a database of intervals. Then, $IH(I, \nu) = (n_1, .., n_\nu) \in \mathrm{N}^\nu$ is called the interval histogram on I with resolution ν, iff for all $i \in \{1, .., \nu\}$:*

$$n_i = |\{\Psi \in I | \Psi \ \ intersects \ \ b_{i,\nu}\}|.$$

The main drawback of interval histograms is that possible intervals may span multiple histogram buckets if they have large extensions or if the resolution of the histogram is very accurate. As a consequence, the interval replications among multiple histogram buckets can be very high, causing the accuracy of interval-based selectivity estimation to deteriorate.

4.5.2 Quantile Vector

Another approach to build statistics reflecting the current distribution of intervals is the quantile-based approach. Whereas histograms can be naturally applied to one-dimensional interval data, a quantile-based approach has to operate on a linear representation of the original intervals, for example the RI-tree key values of the fork nodes of the intervals.

Definition 4.7 (Quantile Vector) *Let $(M, \leq)$ be a totally ordered multiset. Without loss of generality, let $M = \{m_1, m_2, ..., m_N\}$ with $m_j \leq m_{j+1}$,*

$1 \leq j < N$. *Then,* $Q(M, \nu) = (q_0, ..., q_\nu) \in M^\nu$ *is called a quantile vector for* M *and a resolution* $\nu \in \mathrm{N}$*, iff the following conditions hold:*

- $q_0 = m_1$.
- $\forall i \in \{1, .., \nu\} : \exists j \in \{1, .., N\} : q_i = m_j \wedge \frac{j-1}{N} < \frac{i}{\nu} \leq \frac{j}{N}$.

The multi-set M of our quantile vector $(q0, ..., q_\nu)$ is formed by the values of the first attribute A_1 of the domain values of our index I. By using the node quantiles for the RI-tree index, we get an aggregated view on the locations of the stored intervals.

Part II

Spatial Query Processing for Complex Structured Objects

Chapter 5

Introduction

Modern database applications including computer-aided design impose new requirements on efficient spatial query processing. Particular problems arise from the need of high resolutions for large spatial objects, including cars, space stations, planes and industrial plants. Applications which need an integration into industrial-strength systems need that the spatial objects are efficiently managed within commercial database management systems. As a consequence, a seamless and capable integration of spatial indexing into industrial-strength databases is essential. Unfortunately, most commercially relevant database systems provide no built-in access method for temporal and spatial data types, nor do they offer a generic framework to facilitate the integration of user-defined search trees based on disk blocks. In [KPPS03b] the paradigm of relational indexing is proposed. The idea is to use relational access methods to integrate index support for temporal and spatial data types. As access methods of this class are designed on top of the pure SQL layer, they can be easily virtually implemented on any available relational database server. A comprehensive overview about relational access methods for spatial data and how to integrate them into modern Object Relational Database Management Systems (ORDBMS) is given in [Pöt01].

In the previous sections we have shown how complex spatial objects can be suitably modeled by sequences of points or intervals of space primitives (Voxels/Points), and how these interval sequences can be efficiently orga-

nized by the Relational Interval-tree. However, the representation of highly resolved spatial objects having an intricate structure which are consistent with spatial access methods either cause to much approximation error which drastically lowers the query selectivity or implicate very high redundancy. A good trade-off between the opposing attributes redundancy and accuracy is indispensable for efficient spatial query processing on high resolution objects.

The approaches presented in the following chapters involve statistical information in order to accelerate the processing of spatial intersection queries for linearized spatial objects.

First, we will show in Chapter 6 how statistics can be used for accelerating relational access methods by reducing the number of generated join partners which results in fewer logical reads and consequently improves the overall runtime. We take special emphasis on replicating (space partitioning) access methods like the Relational Quad-tree and the Relational Interval-tree (RI-tree). Both access methods are based on the build-in B^+-tree already integrated in standard database management systems. The advantage of replicating storage of complex three-dimensional objects (e.g. CAD parts of an airplane) is that it does not cause region queries to produce too many false hits that have to be eliminated by subsequent filter steps compared to one-value approximations organized in a non-replicating access method like the R-tree [Pöt01].

Furthermore, in Chapter 7 we present an approach which achieves efficient query processing along with industrial-strength database support for real time haptic rendering systems, providing force feedback (haptic display).

In Chapter 8 we introduce interval containers as a new and general concept. In contrast to the common interval decomposition which suffer from the high redundancy of highly resolved complex shaped objects, interval containers combine groups of intervals into approximations - one approximation per group.

Based on the interval container concept, we present in Chapter 9 intersection join methods adequate for interval sequences representing complex spatial objects. Our main focus are methods for generating the object ap-

proximations adequate for speeding up the join processing of high-resolution objects.

Finally, in Chapter 10 we present a new distributed intersection join algorithm for interval sequences of high-cardinality, distributed over several clients. Our approach aims at minimizing the transmission cost to the server which performs the join processing. This approach is based on a suitable probability model for interval intersections.

Chapter 6

Statistic Driven Acceleration of Spatial Queries

6.1 Introduction

Relational access methods perfectly fit to the common relational data model and are highly compatible in many cases with the extensible indexing frameworks of existing object-relational database systems. In order to integrate these index structures into modern ORDBMSs, we need suitable cost models [KPPS02], which exploit the built-in statistics facilities of the database server. Based on these statistics, it is possible to estimate the selectivity of a given query and to predict the cost of processing it. In this chapter, we will show how these statistics can be used to minimize the overall navigational cost of space partitioning relational index structures which are organized by the built-in B^+-tree of the DBMS. In particular, we will consider the Relational Interval-tree and the Relational Quad-tree, the most prominent space partitioning relational access methods which are adequate for organizing complex spatial objects.

Our approach accelerates the relational access methods by trying to reduce the total number of logical reads for a given interval sequence query. Former approaches which try to generate efficient read schedules for a given

set of disk pages [SLM93] must know the actual position of the pages on the storage media. However, in an ORDBMS, the user has no access to the exact information where the blocks are located on the disk. As this information is not available in an ORDBMS, we pursue another idea which exploits statistics related to the corresponding relational access method in order to accelerate spatial query processing. The relational access method can be any custom index structure mapped to a fine granular relational schema organized by built-in access methods, as for instance the B^+-tree. We introduce our approach in general as well as exemplarily for spatial intersection queries performed on the Relational Quad-tree (RQ-tree) and the Relational Interval-tree (RI-tree). This work has been published in [KKPR04c].

We first look at very comprised statistic values which can already be very useful for accelerating the relational access methods. Then we will show how we can benefit from simple statistics related to the underlying B^+-tree. Afterwards, we introduce a new statistic-based query decomposition approach in addition to the existing error- and size-bound decomposition approaches [Ore89].

6.2 Statistics Related to the Relational Access Methods

In [KPPS03b] relational access methods are defined as follows:

Definition 6.1 (Relational Access Method) *An access method is called a relational access method, iff any index-related data is exclusively stored in and retrieved from relational tables. An instance of a relational access method is called a relational index. The following tables comprise the persistent data of a relational index:*

- ***User table:*** *a single table, storing the original user data being indexed.*
- ***Index tables:*** *n tables, $n \geq 0$, storing index data derived from the user table.*

- ***Meta table:*** *a single table for each database and each relational access method, storing O(1) rows for each instance of an index.*

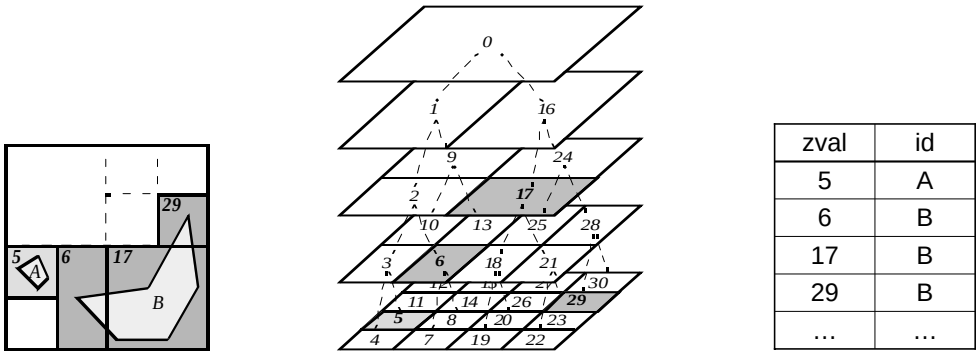
The stored data is called user data, index data, and meta data.

As already indicated in the definition, the metadata table is a single table for each database and each relational access method, storing O(1) rows for each instance of an index. All schema objects belonging to the relational index, in particular the name of the index table, and other index parameters are stored in this global meta table.

6.2.1 Examples of Space-Partitioning Relational Access Methods

The most prominent examples for space-partitioning relational access methods are the Relational Interval-tree (RI-tree) and the Relational Quad-tree which is a relational mapping of the Linear Quadtree [Sam 90]. In Section 4.4.1 we have already introduced the basic concept of the RI-tree. In this section, we present the basic idea of the Linear Quadtree according to the in-depth discussion of Freytag, Flasza and Stillger [FFS 00].

The Relational Quad-tree organizes the multidimensional data space by a regular grid. Any spatial object is approximated by a set of tiles. Among the many possible one-dimensional embeddings of a grid approximation, the Z-order is one of the most popular [Güt94]. The corresponding index representation of a spatial object comprises a set of Z-tiles which is computed by recursively bi-partitioning the multidimensional grid. By numbering the Z-tiles of the data space according to a depth-first recursion into this partitioning, any set of Z-tiles can be represented by a set of linear values. Note that thereby redundancy is introduced to approximate spatially extended data. Figure 6.1 depicts some Z-tiles on a two-dimensional grid along with their linear values. The linear values of the Z-tiles of each spatial object can be stored in an index table obeying the schema (zval, id), where both columns comprise the primary key. Each row in the index table exclusively



zval	id
5	A
6	B
17	B
29	B
...	...

(a) Regular grid **(b)** Recursive partitioning **(c)** Index table

Figure 6.1: Relational Quad-tree.

belongs to a single data object. The linear ordering positions each Z-tile of an object on its own row in the index table.

In order to process spatial selection on the Relational Quad-tree, the query region is also required to be decomposed to a set of Z-tiles. For each resulting linear value zval, the intersecting tiles have to be extracted from the index table. Due to the Z-order, all intersecting tiles having the same or a smaller size than the tile represented by zval occupy the range ZLowerHull(zval) = [zval, ZHi(zval)] which can be easily computed [FFS00]. In the example of Figure 6.1, we obtain ZLowerHull(17) = [17, 23]. In a similar way, we also compute ZUpperHull(zval), the set of all larger intersecting tiles. As in the case of ZUpperHull(17) = 0, 16 the corresponding linear values usually form no consecutive range. To find all intersecting tiles for a given zval, a range scan on the index table is performed with ZLowerHull(zval) and multiple exact match queries are executed for ZUpperHull(zval). These queries are optimally supported by a built-in B^+-tree on the zval column.

6.2.2 Index Specific Statistics

Especially in the case of space partitioning index structures, often a few values describing the actual data distribution help to reduce the I/O cost dramatically. If we assume for instance that one half of the data space is

denotation	explanation
$MaxNodeLevel$/ $MinNodeLevel$ (RI-tree)	These two parameters reflect the highest and lowest level of the fork nodes of the intervals in the database. If we arithmetically traverse the primary structure for a given query interval $q = (l, u)$, we only have to collect those nodes n as join partners, for which $MinNodeLevel \leq Level(n) \leq MaxNodeLevel$ holds.
$MaxLeftDist$/ $MaxRightDist$ (RI-tree)	These two parameters reflect the maximum distance of the boundary values of any database interval to its corresponding fork-node. If we arithmetically traverse the primary structure for a given query interval $q = (l, u)$, we only have to collect those nodes n as join partners for which $n - MaxLeftDist \leq u$ and $n + MaxRightDist \geq l$ holds.
$MaxTileLevel$/ $MinTileLevel$ (RQ-tree)	These two parameters reflect the highest and lowest level of stored tiles within the database. If we compute the upper hull of a given query tile q, we only have to consider those tiles t as join partners for which $MinTileLevel \leq Level(t) \leq MaxTileLevel$ holds.

Table 6.1: Simple statistics for the RI-tree and RQ-tree.

completely empty, and we carry out a range query in this area, we can omit a lot of unnecessary I/O accesses if we take the actual data distribution into consideration. Consequently, it is beneficial to store the variable data extension along with the fixed data space extension.

In Table 6.1, we summarized some optimizations which are suitable for the RI-tree and the RQ-tree. These simple statistics are especially useful for indexing extended spatial objects. Very often only the lower levels of the virtual primary structure are engaged, as spatial objects tend to decompose into numerous small tiles or intervals [KPPS03a].

6.3 Statistics Related to the built-in access method (B^+-tree)

In this section, we show how to apply suitable statistics in order to estimate the selectivity of a query or sub-queries. Selectivity estimation for queries is a very important component to guide the strategy of the query processor. In particular, it helps to predict the potential execution times of a query. The selectivity information can then be used to estimate the cost of different query strategies which helps to select or build the cost optimal solution of the query processing. Generally, this information is used to select the most promising access method for a query. But it can also be used at a more fine granular level - it enables to reduce the processing cost of access methods by helping to find the "cost optimal" navigation through the index structure. In particular, access methods like the B^+-tree allowing different execution strategies, i.e. hierarchical navigation or linear scan on the leaf level, to obtain the result set can potentially benefit from such a guided search algorithm. This idea is primarily pursued by the approach introduced in this section.

We have already introduced two methods to build statistics on interval data, the interval histogram and a quantile-based method. In [KPPS02] it was shown that applying quantiles ('equi-count histograms') is more suitable for estimating the selectivity and the corresponding I/O cost than using histograms ('equi-width histograms'). We will now discuss how we can utilize the quantile based information to accelerate the query process for interval sequences itself. An interval intersection query leads to several index range scans on the corresponding built-in index structure, the B^+-tree. The general idea of our approach is to minimize the overall navigational cost of the B^+-tree by applying extended index range scans. Thereby, we read false hits from the index which have to be filtered out by a subsequent refinement step. As shown in Figure 6.2, our approach closes the gaps between the index scan ranges if and only if the number of additional read data is comparably small, more precisely, the cost related to these false hits is smaller than the navigational cost related to an additional range scan. The decision whether to close a gap or not is based on the quantile statistics. In the following we

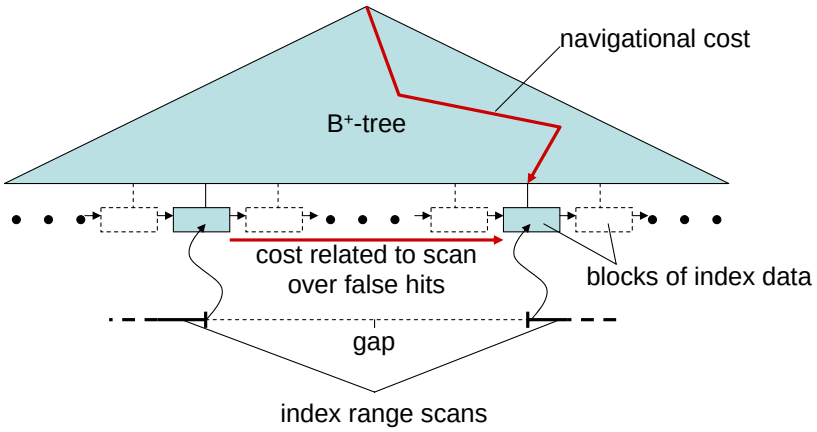


Figure 6.2: Minimizing navigational cost of the B^+-tree.

will formally introduce this idea.

6.3.1 Index Range Scan Sequences

For spatial intersection queries, the query object Q leads to many disjoint range queries $s_i = (l_i, u_i)$ on the index I, e.g. the B^+-tree. We consider them as a sequence $Seq_{Q,I} = (\langle s1, ..., sn \rangle)$ of index range scans (cf. Figure 6.3a) for which the following assumptions hold:

- The elements r_i stored in the index are of the same type as l_i, u_i. Furthermore, we assume that the elements r_i can be regarded as a linear ordered list $L(I) = \langle r_1, ..., r_N \rangle$ for which $r_1 \leq ... \leq r_N$ holds.
- We assume that the data pages p_i of the index obey a linear ordering $\leq$ and fulfill the following property:

  $$r' \leq r'' \Leftrightarrow p(r') \leq p(r''),$$

  where $p(r)$ denotes the disk page of the index I which contains the entry r.

I/O-cost. The I/O cost $C^{I/O}(s)$ associated with one index range scan $s = (l, u)$ of $Seq_{Q,I} = (\langle s_1, ..., s_n \rangle)$ are composed from two parts:

- $C_n^{I/O}(s)$ the navigational I/O cost for finding the first page of the result set, and
- $C_s^{I/O}(s)$ the cost for scanning the remaining pages containing the complete result set.

Formally,

$$C^{I/O}(s) = C_n^{I/O}(s) + C_s^{I/O}(s),$$

with the following two properties:

1. $C_n^{I/O}(s) = C_n^{I/O}(p(r'))$ (navigational cost)
2. $C_s^{I/O}(s) = C_s^{I/O}(\langle p(r'), ..., p(r'')\rangle)$ (scan cost),

where $r', r'' \in L(I)$ and $\forall r \in L(I) : (r' \leq r \leq r'') \Leftrightarrow (l \leq r \leq u)$ holds.

The I/O cost $C^{I/O}(Seq_{Q,I})$ associated with $Seq_{Q,I} = (\langle s_1, ..., s_n\rangle)$ are determined by

$$C^{I/O}(Seq_{Q,I}) = \sum_{i=1..n} C^{I/O}(s_i).$$

6.3.2 Extended Index Range Scan Sequences

The main purpose of our approach is to minimize the overall cost for the navigational part of the index. Therefore, we try to reduce the number of generated range queries on the index I, while only allowing a small increase in the output cost. This can be achieved by merging two suitable adjacent range scans $s' = (l', u')$ and $s'' = (l'', u'')$ together to one extended range scan $xs = (l', u'')$.

Intuitively, an extended range scan $xs = \langle s_r, ..., s_s\rangle$ is an ordered list of index range scans. When carrying it out, we traverse the index directory only once and perform a range scan (l_r, u_s), as for example (l_3, u_4) in Figure 6.3b. Performing the extended range scan, we read false hits from the index I which have to be filtered out in a subsequent refinement step. The overall cost $C(xs)$ of an extended range scan xs is composed from the sum of the I/O

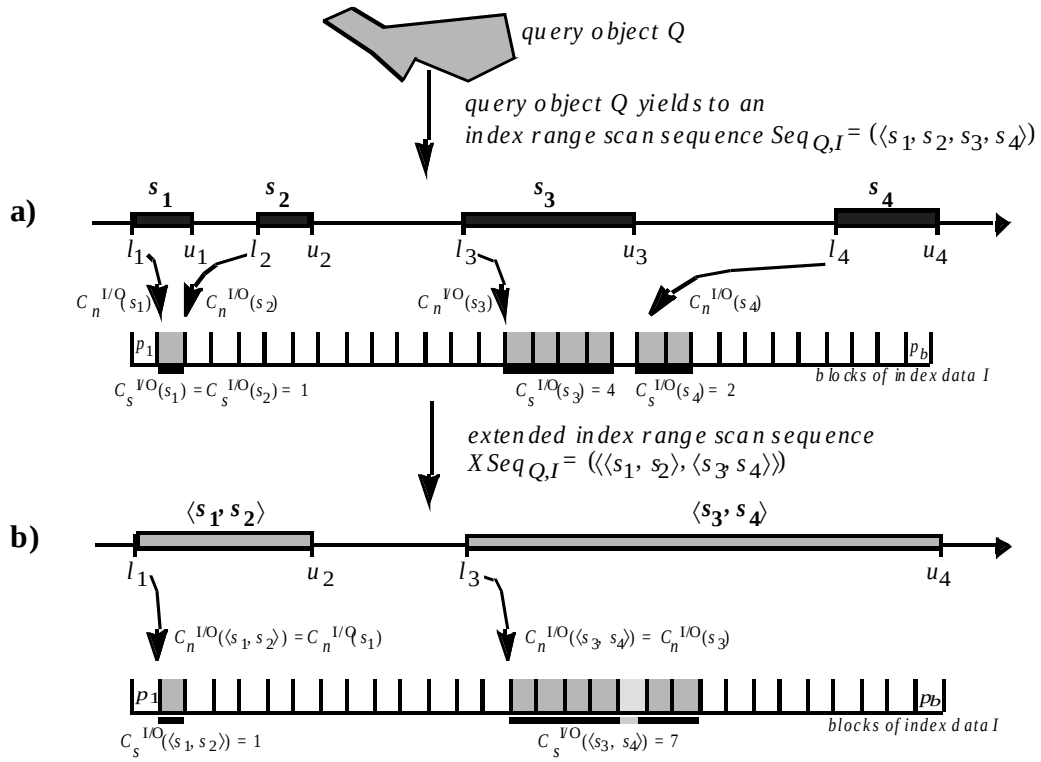


Figure 6.3: Accelerated query processing.

cost of the extended range scan and the CPU cost related to the refinement step:

$$C(xs) = C^{I/O}(xs) + C^{CPU}(xs).$$

I/O-cost. $C^{I/O}(xs)$ associated with one extended range scan $xs = \langle s_r, ..., s_s \rangle$ are composed from two parts

$$C^{I/O}(xs) = C_n^{I/O}(xs) + C_s^{I/O}(xs),$$

with the following properties:

1. $C_n^{I/O}(xs) = C_n^{I/O}(sr)$ (navigational cost)
2. $C_s^{I/O}(xs) = C_s^{I/O}(l_r, u_s)$ (scan cost).

CPU-cost. $C^{CPU}(xs)$ associated with one extended range scan $xs = \langle s_r, ..., s_s \rangle$ denotes the cost which is required to perform the filter operation for all tuples

resulting from the extended range scan:

$$C^{CPU}(xs) = C^{CPU}(\langle r', .., r'' \rangle),$$

where

$$\forall r \in L(I) : (r' \leq r \leq r'') \Leftrightarrow (l_r \leq r \leq u_s).$$

The total cost $C(XSeq_{Q,I})$ associated with an extended index range scan sequence $XSeq_{Q,I} = (\langle xs_1, .., xs_m \rangle)$ can be computed as follows:

$$C(XSeq_{Q,I}) = \sum_{j=1..m} C(xs_j).$$

Obviously, there might exist extended index range scan sequences $XSeq_{Q,I}$ for which $C(XSeq_{Q,I}) << C(Seq_{Q,I})$ holds. For each gap g between two adjacent range queries s' and s'' we decide if the cost of scanning over the gap g are lower than the navigational I/O cost related to s''. The decision whether to merge range scan s' and s'' to one extended range scan and apply an additional refinement step afterwards in order to filter out false hits is based on statistics which are necessary for the cost models anyway.

Selectivity Estimation. The multi-set M of our quantile vector $(q_0, ..., q_v)$ (cf. Definition 4.7) is formed by the values of the first attribute of the domain values of our index I, i.e. the fork-node ids in case of the RI-tree and the tile ids in case of the RQ-tree. By means of these statistics, we can estimate the I/O cost $C_s^{I/O}(s)$ associated with one range scan $s = (l, u)$. In the following formula, b denotes the number of disk blocks at the leaf level of I, v denotes the resolution of the quantile vector, N denotes the overall number of entries stored in the index I and *overlap*() returns the intersection length of two intersecting intervals:

$$C_s^{I/O}((l,u)) \approx C_s^{est}((l,u)) = \frac{\sum_{i=1..v} \frac{overlap((l,u),(q_{i-1},q_i))}{q_i - q_{i-1}} \cdot \frac{N}{v}}{N/b}.$$

We can also apply the above formula to estimate the total cost $C_s(g) = C_s^{I/O}(g) + C^{CPU}(g)$ related to scanning over a gap $g =]u', l''[$ between two adjacent range queries s' and s''. The CPU cost can be estimated by

$$C^{CPU}(g) = k \cdot C_s^{I/O}(g)$$

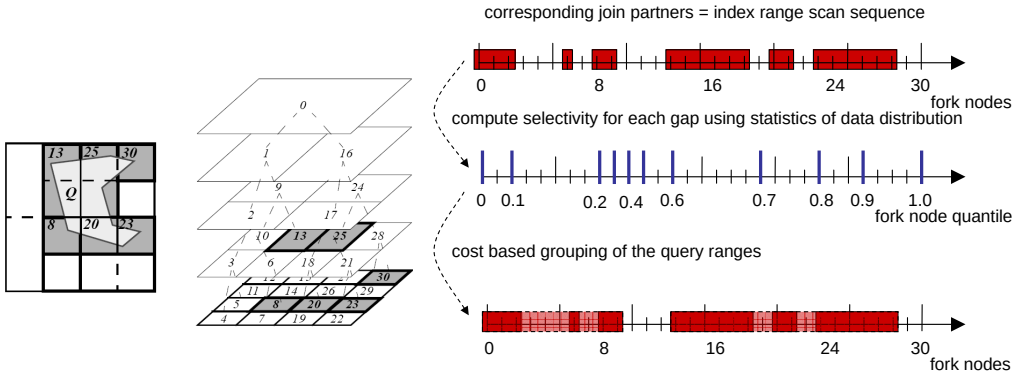


Figure 6.4: Cost-Based Tile Grouping.

with a parameter $k > 0$, since both the I/O cost and the CPU cost are directly proportional to the size of the result set of the range scan. If $C_s(g)$ are lower than $C_n(s'')$, we close the gap g.

Building Extended Index Range Scan Sequences. We can find the extended range scan sequence $XSeq_{Q,I}$ trying to minimize $C(XSeq_{Q,I})$, by deciding for each of the $n-1$ gaps between the index range scans $s_1, ..., s_n$ of the index range scan sequence $Seq_{Q,I} = (\langle s_1, ..., s_n \rangle)$ whether we close this gap or skip it. Thus, we obtain an extended index range scan sequence

$$XSeq_{Q,I} = (\langle \langle s_{i_0+1}, ..., s_{i_1} \rangle, ..., \langle s_{i_{m-1}+1}, ..., s_{i_m} \rangle \rangle)$$

which satisfies the following property:

$$\forall i \in 1...n-1 : i \in i_1..i_{m-1} \Leftrightarrow C_n^{est}(s_{i+1}) < C_s^{est}((u_i, l_{i+1})).$$

Usually, the actual navigational cost $C_n^{I/O}$ is independent of the actual range scan and can easily be estimated by C_n^{est}, e.g. by the height of the B^+-directory.

In the following, we will show how our approach can be applied to the intersect predicate for the two B^+-tree based index structures, the Relational Interval-tree and the Relational Quad-tree.

6.3.3 Adoption to the Linear Quad-tree

Assume object Q in Figure 6.4 is used as query object. Then there are multiple exact match and range scan queries building a query interval sequence which have to be performed in order to detect all intersecting database objects. We can reduce the cost by closing small gaps on the leaf-level of the underlying B^+-tree. By using the information stored in the statistics, i.e. using the tile quantiles, the number of join partners which correspond directly to the navigational cost $C_n^{I/O}$ can be reduced drastically. The quantile vector is built over the values stored in the leaf-level of the corresponding B^+-tree.

We investigate all gaps included in the sequence of our generated join partners and decide whether it is beneficial to close this gap. Assume, the height of our B^+-directory is n. If we close the gap, we reduce the navigational cost as follows:

$$C_n^{I/O} = C_n^{I/O} - n.$$

On the other hand, we estimate the cost $C_s(g)$ required to read the leaf blocks on our index ($zval$) which are covered by the database tiles of the actual investigated gap g. If these estimated cost are lower than n, we close this gap. Thus, we reduce the join cost $C_n^{I/O}$ by n, while not increasing the output cost C_s by more than n. This procedure is depicted in Figure 6.4.

6.3.4 Adoption to the Relational Interval-tree

Similarly to the Linear Quad-tree, we can prepare the interval sequence query for the Relational Interval-tree (cf. Section 4.4.1) using our cost-based grouping algorithm. This algorithm is independent of the index structure at the primary level, i.e. the structure of the Linear Interval-tree. It is only based on a B^+-tree and on a quantile vector. In the case of the Relational Interval-tree, the quantile vector is formed by the fork nodes of the intervals stored in the database.

6.4 Statistics Related to the Object Decomposition

Both, the $error-$ and $size-bound$ decomposition approach for spatially extended objects lead to a sequence of simple query objects, e.g. a sequence of tiles or intervals. The resulting sequence of entities in turn leads to a sequence of range queries on the embedding B^+-tree. In this section, we introduce an additional decomposition approach which decomposes the query object based on the expected I/O cost. The expected I/O cost can be estimated in the same way as a query optimizer estimates the cost for a given query. Like the approach of the last section, the decomposition of the query object is controlled by our statistics.

```
QV: quantile vector
Q: query object

Decompose(Q, QV){
    query_sequence_list := split_at_maximum_gap(Q);
    cost_0 := statistic_look_up(Q, QV);
    cost_dec := 0;
    for each q in query_sequence_list do
        cost_dec := cost_dec + statistic_look_up(q, QV);
    if cost_0 > cost_dec then
        for each q in query_sequence_list do
            Decompose(q, QV);
    else report(Q);
}
```

Figure 6.5: Grouping Algorithm *Decompose*.

Figure 6.5 depicts this top down grouping algorithm which is beneficial for the both discussed index structures. The algorithm starts with a query object comprising the complete object. In each step, we determine the maximum included *gap* and split along this *gap*, resulting in a sequence of query objects. Then we estimate the I/O cost related to the original query object and the

cost related to the sequence. If the cost of the original query object is smaller than the cost of the sequence, we terminate the algorithm. The query object now consists of a sequence of query objects. In an additional refinement step, we eliminate the false hits which result from the fact that we have not decomposed the spatial object with the maximum possible accuracy.

Box Volume Queries. The introduced approach is especially useful for highly selective box volume queries on the RI-tree or the RQ-tree. 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. We decompose a query box dependent on the stored data. If there are not many data stored in the query area, the box is decomposed into comparable few simple query objects, i.e. tiles or intervals. On the other hand, if the query returns a lot of results, we decompose the query into comparable many simple query objects.

A box can be described by a few parameters, e.g. by two points. The few parameters which are necessary to describe the box are attached to each incompletely decomposed interval or tile. In the refinement step, we further decompose the query intervals or tiles on demand from the compact geometric information.

6.5 Experimental Evaluations

In this section, we will experimentally investigate the performance of the proposed *statistic-based* decomposition method. First, we will specify the experimental environment.

6.5.1 Test Datasets

The tests are based on two test datasets CAR and $PLANE$. Both were provided by our industrial partners, a German car manufacturer and an American plane producer, in form of high-resolution voxelized three-dimensional CAD parts. The CAR dataset consists of approximately 14 million voxels and 200 parts, whereas the $PLANE$ dataset consists of about 18 million voxels and 10,000 parts. The CAR data space is of size 2^33 and the $PLANE$ data space has a size of 2^42. In both cases the Z-curve was used as a space filling curve to enumerate the voxels. Table 6.2 gives a summarization of the dataset specification.

datasets	CAR	$PLANE$
# voxels	$14 \cdot 10^6$	$18 \cdot 10^6$
# parts	200	10000
data space (# voxels)	2^{33}	2^{42}

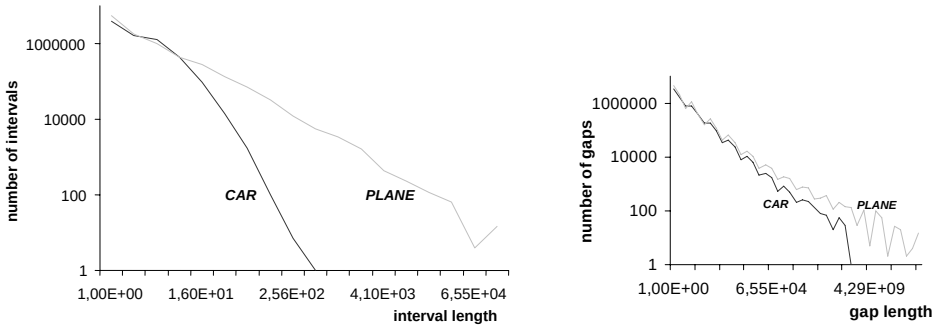
Table 6.2: Dataset specification.

6.5.2 System Specification

We have implemented our approach for the RI-tree and the RQ-tree on top of an object-relational DBMS, the Oracle9i Server using PL/SQL for the computational main memory based programming. All experiments were performed on a $PentiumIII/700$ 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.

6.5.3 Histograms of the Test Datasets

Figure 6.6 depicts the interval and gap histograms for our two test datasets. Both consist of many short intervals and short gaps and only a few longer ones. Consequently, mainly the lowest levels of the RQ-tree and RI-tree



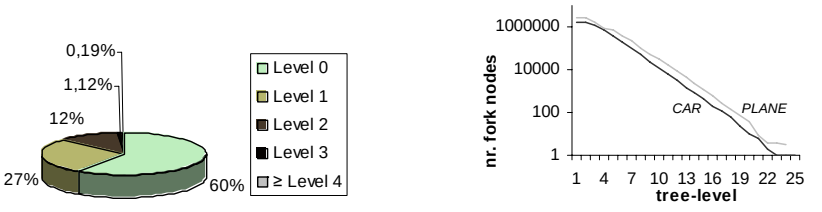
(a) Intervals **(b)** Gaps

Figure 6.6: RI-tree histogram.

contain index entries. Figure 6.7a shows that in the case of the RQ-tree on the CAR dataset only the five lowest of 33 levels are occupied. Similar observations are made for the RI-tree (cf. Figure 6.7b) where most intervals are registered at very low fork-node levels. The observation that spatial objects are decomposed into many small intervals and tiles, are not confined to our two test datasets but holds for spatial objects in general [Gae95] [KPPS03a]. Therefore, the statistics presented in Table 6.1 are very beneficial for efficient query processing on spatially extended objects.

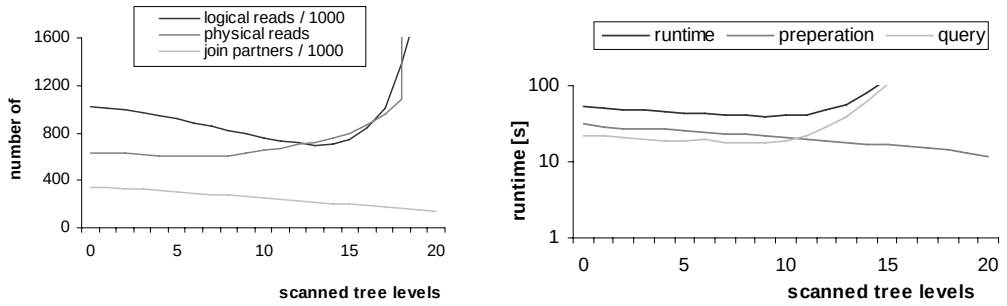
6.5.4 Query Processing

In this section, we examine the benefits of using extended index range scans. For the RI-tree and the RQ-tree, we used 10% of the database objects as



(a) Tile Levels (CAR) **(b)** Fork Node Levels (CAR, $PLANE$)

Figure 6.7: Used index levels.



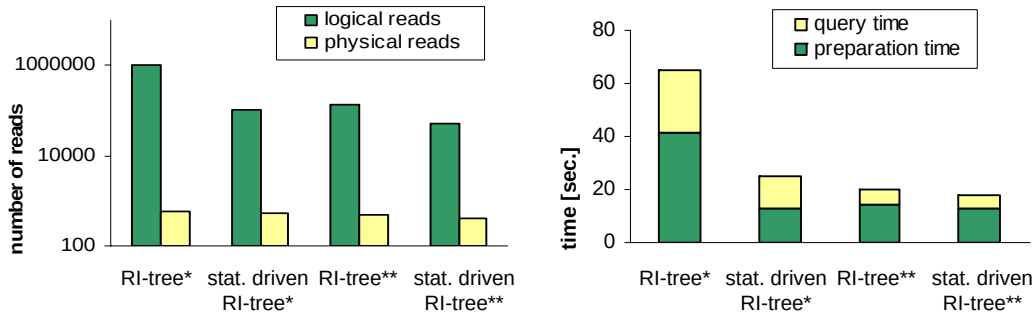
(a) Number of logical/physical reads **(b)** Runtime performance

Figure 6.8: RI-tree optimizations without using statistics.

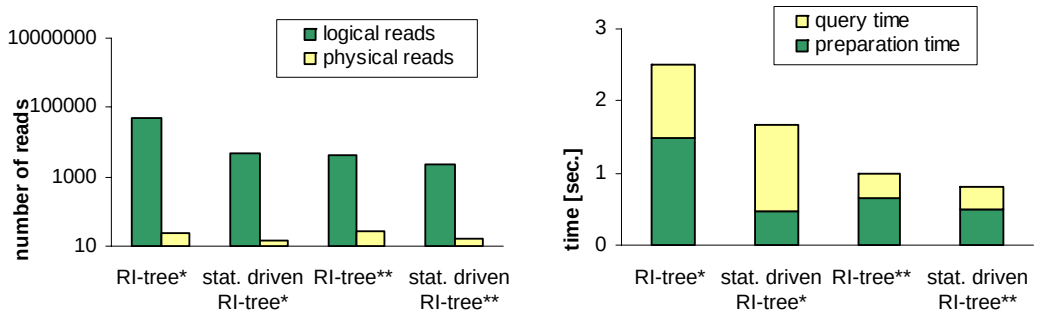
query objects and report the average results from these queries.

Extended range scans without statistics. In a first experiment, which does not use any statistical information, we point out the benefits of using our extended index range scans (cf. Section 6.3.2). For a given query object, we did not collect all possible join partners, but omitted the last levels and used an extended index range scan instead. Figure 6.8a shows that the number of join partners decreases with an increasing number of scanned tree levels. At the beginning, the number of logical reads also decreases, but if we neglect too many tree levels of the RI-tree, the number of logical reads increases again along with the increasing number of physical reads. The number of physical reads stays almost constant if we scan over only a small number of levels. On the other hand, the number of physical reads dramatically increases if the number of scanned tree levels exceeds 18 because of the increasing number of false hits which are filtered out in a consecutive filter step. In Figure 6.8b it is shown that the preparation time decreases with an increasing number of scanned tree levels. Due to the reduced number of join partners and the decreasing preparation time, the overall runtime reaches a minimum if we neglect the last 10 levels of the RI-tree and apply an extended range scan instead. By using a fixed scan level, we can already improve the query response time by 30%. In the following sections, we will see that if we use statistics to form our extended range scans, we can further improve the overall query response behavior.

Extended range scans with statistics. In Figure 6.9 it is shown in detail



(a) *CAR* dataset

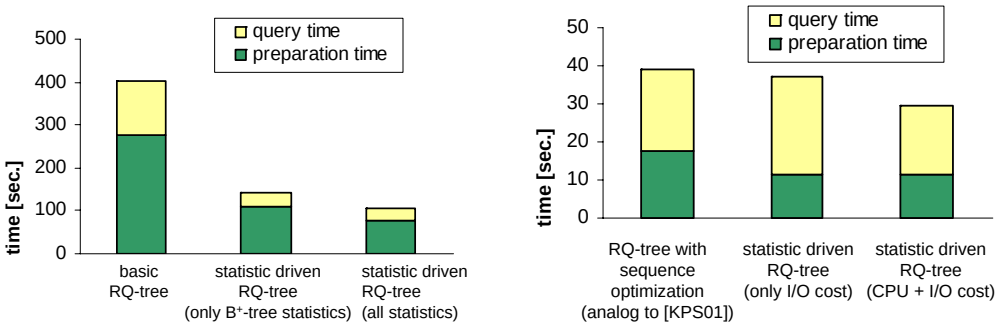


(b) *PLANE* dataset

Figure 6.9: Cost-Based Tile Grouping.

that our new statistic-based approach accelerates both, the basic variant of the RI-tree [KPS00a](*), and a variant which is optimized for efficient handling of interval sequences [KPS01](**). The interval sequence optimized approach (**) merges in a preprocessing step all join partners (fork node ids) of all query intervals into one query interval sequence. Redundant join partners are dropped from the query sequence. Figure 6.9 depicts that we can reduce the number of logical reads approximately by an order of magnitude if we exploit the available statistics. This reduction is achieved without increasing the number of physical reads so that the overall runtime decreases. If we use the statistics, we outperform the simple scanning approach even for the optimum scanning level (cf. Figure 6.8). In all our tests we accelerate the query process by 20% to 150% if we form the extended range scans according to the available statistics.

In the next experiments, we applied the statistic based approach to the



(a) Number of logical/physical reads **(b)** Runtime performance

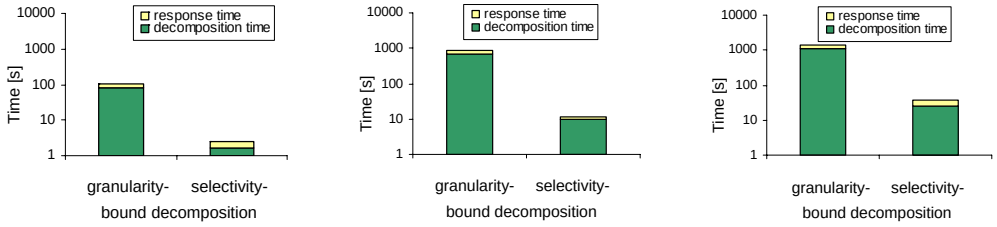
Figure 6.10: Statistic based accelerated RQ-tree on the *CAR* dataset.

RQ-tree (cf. Figure 6.10). Figure 6.10(a) shows that the use of our quantile statistics (cf. Section 6.3) accelerates the RQ-tree by a factor of about 2. A further improvement can be achieved by using the information of the highest and lowest level of stored tiles within the database (cf. Section 6.2), leading to a speed-up factor of almost 3. Figure 6.10(b) depicts the acceleration of the sequence optimized RQ-tree incorporating the same query optimizations as applied to the RI-tree in [KPS01], i.e. redundant join partners (tile ids) are pruned from the query sequence. We compare the statistic driven variant without incorporating the CPU cost of the refinement step with the statistic driven variant including the CPU cost (cf. Section 6.3.2). The first variant considers only the I/O cost and neglects the CPU cost for forming the extended range scan sequences: Figure 6.10(b) shows that this approach leads only to an acceleration in the preparation step, but the overall query time increases due to the expensive refinement process. On the other hand, if we incorporate the CPU cost for the cost estimation, we can achieve an overall speed-up of approximately 30%, even for this highly specialized index structure.

To sum up, similar to the experiments related to the RI-tree, we achieve an acceleration factor of the query process by 0.3 to 3 if we form the extended range scans according to the available statistics, considering both expected I/O-cost and expected CPU-cost.

Statistic based decomposition. In a last experiment, we carried out

different box volume queries on the RI-tree for the $PLANE$ dataset. Figure 6.11 depicts the average runtime for three different boxes, where we moved each box to 10 different locations. As shown in Figure 6.11, our $statistic-based$ decomposition approach can improve the query response behavior up to 10,000%, i.e. by two orders of magnitude, compared to the $granularity-bound$ approach. This speed-up is mainly due to the reduced decomposition time. On the other hand, the query response time does not suffer from the fact that we did not decompose the boxes 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 box. To sum up, our statistic-based decomposition approach is especially adequate for commonly used box volume queries.



(a) box size equal to 0.00002% of data space yielding 0.03% selectivity

(b) box size equal to 0.003% of data space yielding 0.1% selectivity

(c) box size equal to 0.008% of data space yielding 1.0% selectivity

Figure 6.11: Box queries on the $PLANE$ data (decomposition and response time).

6.6 Summary

In this chapter, we have shown how we can accelerate spatial query processing by means of statistics which are available for free, as they are maintained by the cost models belonging to the corresponding spatial index structures. According to our experiments, we achieved speed-up factors of up to two orders of magnitude. The achieved performance is due to the fact that it can dynamically switched between a further use of the index structure and a linear

scan. Our statistic-driven approach adapts the access method continuously variable to the best of these two worlds.

In the next chapter, we show that our statistic-based acceleration approach can successfully be applied to time critical applications, in particular Virtual Reality applications with haptic rendering of complex spatial environments.

Chapter 7

Haptic Exploration of Large Spatial Environments

The efficient management of complex spatial information has become an enabling technology for novel database applications in the area of Multimedia and Virtual Reality (VR). Unfortunately, the integration of modern database systems into human centered VR applications, e.g. haptic- and visual exploration of virtual worlds, fails to achieve the indispensably required interactive response times. In this chapter, we will show that our efficient access methods applied to commercial database systems suffices to provide haptic rendering systems with real-time contact-force computation in very large and complex structured virtual 3D-worlds.

7.1 The "*Sense of Touch*" and Database Systems

In the past decades, the interest in haptic applications has increased enormously. In combination with VR applications, haptic rendering enables to simulate a physical environment in such a way that humans can readily visualize, feel, explore and interact with the objects in the environment. For

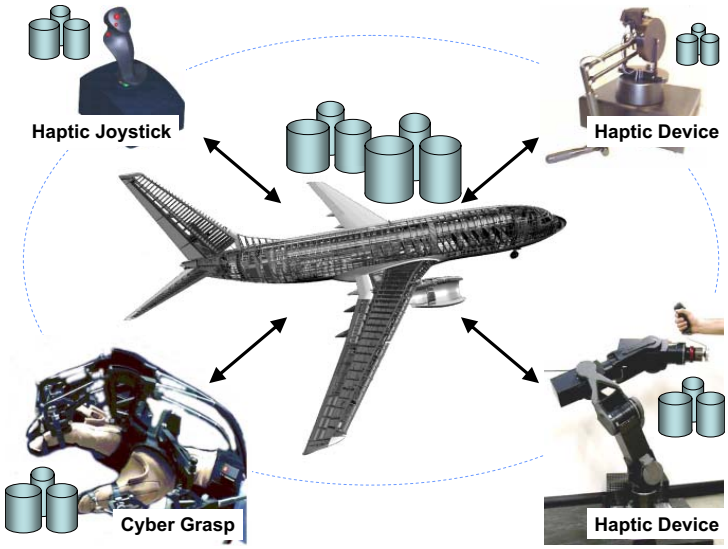


Figure 7.1: Concurrent virtual engineering using different haptic devices.

example, simulation of complex engineering tasks, e.g. installation of an electric bulb within a car, requires more than visual cues, we need to "feel" it. The potential applications of this emerging technology include virtual prototyping, animation, robotic, cooperative design, and education among many others. In all these areas the haptic feedback has a great potential. Figure 7.1 shows a collection of different haptic devices which enable the manipulation of virtual objects and the feedback of the computed forces to the operators.

Generally the environment is given to us as a boundary representation of a set of polyhedral obstacles, each of which is a component part that has designed and modeled within a CAD system. Most of the earlier work has assumed that the virtual environment entirely fits into the main memory [MPT99]. However, this assumption is no longer reasonable. There are several example applications which need the integration of haptic rendering into large scaled virtual environments, including "support of tele-operation for maintenance tasks in space", "simulation of human centered robots" or "haptic walk-through systems". A realistic virtual environment typically consists of thousands of objects which may occupy gigabytes of storage space, especially for high resolution objects. Furthermore, we assume that we have

to provide shared access for multiple users concurrently exploring the same data space (cf. Figure 7.1). In order to fulfill these requirements, we integrate an off-the-shelf database system into a haptic rendering system for the management of the complete environment data. Thereby, we use the substantial advantages of modern database systems, such as logical and physical data independence, concurrency control, recovery and security [Dat99]. In our approach, we employ the object-relational data model, as it is widely accepted and implemented by many database systems. Furthermore, its extensibility is a necessary precondition for the seamless embedding of spatial data types and operations. This work as been published in [KKPR05a].

7.2 Related Work

A lot of work has been done in the field of haptic rendering. A survey is given in [LG98]. To the best of our knowledge, there does not exist any published work that addresses the issue how to combine haptic rendering methods with novel database technologies, so that the force computation can be externalized. For that reason, we separately point out the related work on haptic rendering.

7.2.1 Haptic Rendering

In real-world simulations object models are mostly approximate descriptions of the simulated objects used in the virtual environment. In many rendering algorithms, the geometrical complexity of the particular modeled objects is restricted due to high rendering times. The $Voxmap-PointShell^{TM}$ (VPS) approach [MPT99] and its extensions [RPP+01][MPT06] are very promising due to constant sample rates, independent of the static environment. The real-time capability of the VPS approach qualifies for online haptic interactions between a human operator and a large virtual environment. However, the traditional approach requires that the static environment fits in main memory which limits the size or resolution of the haptic scenario. In this

chapter, we present an externalization of the VPS approach which allows the user to explore a very large static environment but still provides sufficient high haptic rendering frame rates.

7.2.2 Relational Spatial Query Processing

The database integration into the haptic rendering system requires that collision queries are forwarded to the database query engine. In order to achieve the interactive response times required for real-time haptic exploration, the use of efficient spatial access methods based on the relational indexing are indispensable. A general survey on the paradigm of relational index structures can be found in [KPPS03b]. 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. Thereby, the SQL layer of the $ORDBMS$ is employed as a virtual machine managing persistent data.

In our haptic rendering approach, we focused on the acceleration of database queries following the statistic based query processing techniques presented in Chapter 6. By means of the statistic driven acceleration of spatial queries, we try to achieve the performance requirements of the haptic rendering systems.

7.3 Data Model for Haptic Rendering

In the VPS approach [MPT99], the virtual environment consists of objects which can be divided into *dynamic* and *static objects*. *Dynamic objects* can freely move through the virtual space, whereas the *static objects* are fixed in the world coordinate system. Using a haptic device, a user can touch the static objects (static environment) with a dynamic probe, e.g. a virtual clone of the users hand, and the rendered collision forces will be fed back to the

operator. From the haptic point of view, the dynamic objects perform the action, whereas the static environment objects provoke the reaction within the haptic loop. The employed haptic rendering method requires a voxel set (*voxmap*) for the spatial representation of the static objects and a point set (*pointshell*) to represent the dynamic objects.

7.3.1 Static Object Model

The virtual environment can be regarded as a collection of individual 3-dimensional objects, representing a complex and intricate geometric shape, e.g. a completely digitally modeled automobile or aircraft. Thereby the surfaces and solids are designed at a very high precision. This environment is collectively represented by a single spatial occupancy map, called a *voxmap* (volume map). It is created by discretizing the static environment space which is partitioned into regions of free space, and object space. The collection of the discrete volume elements (voxels) of the object spaces builds the *voxmap* (cf. Section 2.1.3).

7.3.2 Dynamic Object Model

The dynamic object is described by a collection of points (*pointshell*) which models its surface. A surface normal vector is assigned to each surface point, pointing to the interior of the object (cf. Section 2.1.4).

7.3.3 Collision Response

The haptic rendering algorithm includes a fast collision detection technique based on probing the *voxmap* with the surface point samples of the *pointshell*. By using the normal vectors of the *pointshell*, an approximate collision force can be computed in constant time for each point-voxel interpenetration. The time consumed to render a single frame depends only on the number of *pointshell* points.

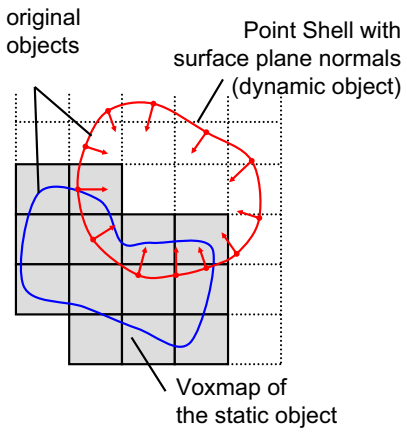
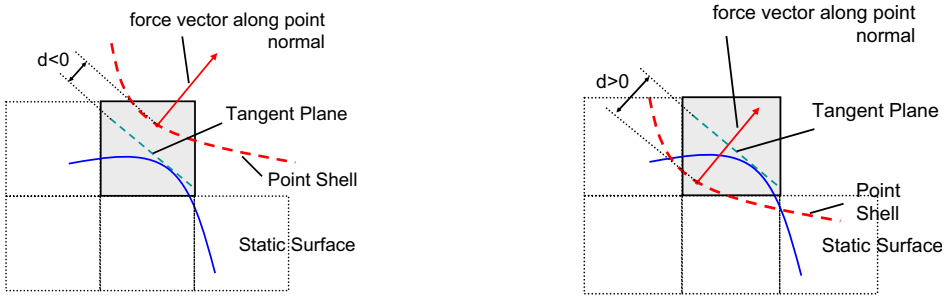


Figure 7.2: Collision detection by probing *voxmap* with *pointshell*.

The interference detection is reduced to point-voxel intersections which can be computed by a plain three-dimensional address computation for each point. If the resolution of the voxel grid and the dynamic point shell are chosen properly, point-voxel intersections are guaranteed to occur in the case of a surface interpenetration. Figure 7.2 [MPT99] depicts a *voxmap* describing the surface of a static object which is probed by a *pointshell* describing the surface of the dynamic object. The associated collision forces are computed by the local interference of a surface point p of the dynamic point shell with a voxel v of the static environment (cf. Figure 7.3(a) and 7.3(b)). The depth of interpenetration d is calculated as the distance from p to the tangent plane $T(p, v)$. This tangent plane is dynamically constructed to pass through the



(a) No surface interpenetration **(b)** Surface interpenetration

Figure 7.3: Force feedback computation.

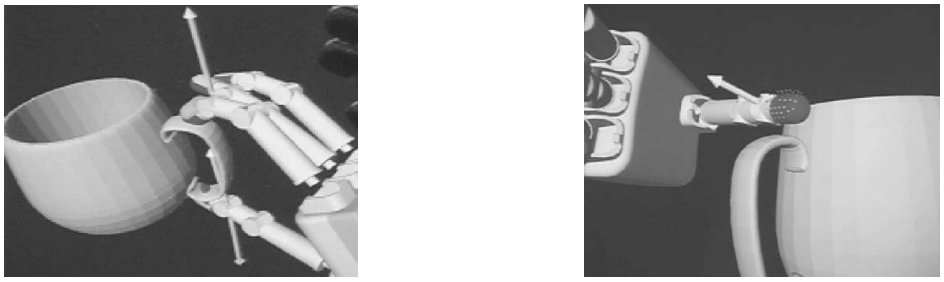


Figure 7.4: Display of the contact forces in a virtual scene.

center of v and to have $n(p)$ as normal vector. If p has not penetrated below the tangent plane, i.e. in the direction to the interior of the static object, we obtain $d < 0$ and produce no local force feedback (cf. Figure 7.3(a)). Contrary, if p is inside of voxel v, and $n(p)$ directs to the corresponding tangent plane of v, we detect a positive interpenetration $d > 0$ (cf. Figure 7.3(b)). According to Hooke's law, the contact force is proportional to d, and the force direction is given by $n(p)$, i.e. the local force vector is determined by interpenetration depth d and the normal vector $n(p)$. The average over all local force vectors of the point shell is transferred to the haptic device to exert the resulting force feedback. In the haptic exploration scene depicted in Figure 7.4, the average force vectors for each fingertip are visualized as arrows pointing towards the effective contact force.

7.4 Relational Embedding of the Static Environment

In this section, we show how to embed the haptic rendering method into a relational schema in order to enable the integration into a commercial database management system. Let us first mention that the Point-Shell representing the dynamic object is small enough, to fit easily into main memory.

The static environment consists of a large set of 3-dimensional objects, each described in a separate CAD-file (object files). The objects of the static environment are stored in a relational table which we call *object table*. It

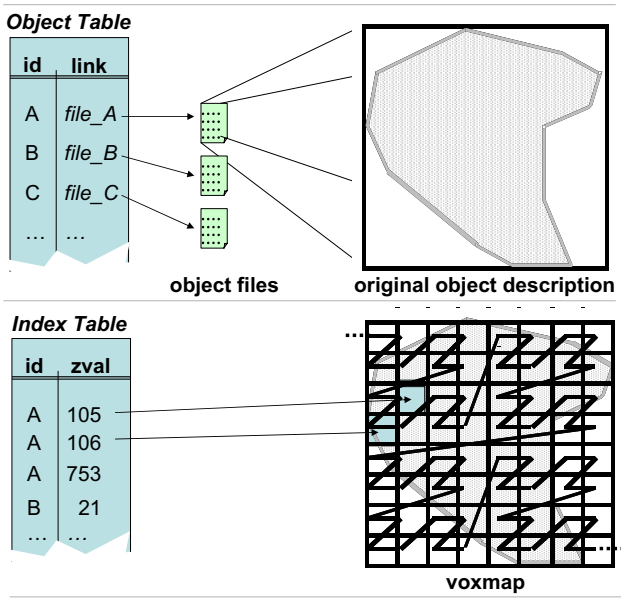


Figure 7.5: Relational embedding of the static environment.

contains a set of tuples (*id*, *link*) where *id* denotes a unique object identifier and *link* refers to the external file containing the complete object as high order surface representation (cf. Figure 7.5).

In order to carry out point-voxel queries efficiently, we propose a simple relational access method which consists of the *object table* and an additional *index table* storing all index data exclusively derived from the *object table* as depicted in Figure 7.5. Each object from the static environment is initially converted into a set of voxels (cf. Section 2.1.3) which are stored with the corresponding object identifier in the *index table*. The table consists of tuples (*id*, *zval*), where the foreign key *id* denotes the identifier of the associated object and *zval* denotes the encoded position of the voxel using the Z-order. We employ the Z-order to achieve a good trade-off between its spatial clustering property and its computational complexity. Due to the linear ordering of the object voxels, the objects can be dynamically indexed by the *zval* attribute using built-in index structures, e.g. a B^+-tree.

7.5 Relational Embedding of the Haptic Rendering Machine

Most ORDBMSs, including Oracle, IBM DB2 or Informix IDS/UDO, provide extensibility interfaces in order to enable database developers to seamlessly integrate custom object types and predicates within the declarative DDL and DML. These interfaces form a necessary prerequisite for the seamless embedding of user-defined spatial objects, functions and access methods into off-the-shelf ORDBMSs. On this basis, we define the haptic query (contact-force computation) which is expressed on top of the SQL engine as follows:

```
SELECT sum(X), sum(Y), sum(Z)
FROM (
   SELECT force_x(PS) AS X, force_y(PS) AS Y, force_z(PS) AS Z
   FROM :POINTSHELL PS
   WHERE EXISTS (
     SELECT 1
     FROM INDEX_TABLE I
     WHERE PS.zval = I.zval)
)
```

Figure 7.6: SQL statement for haptic query processing

The input of this nested SQL query is the *index table* and a collection of points as transient table ($POINTSHELL$) derived from the dynamic object which obey the following schema (id,$zval$,$point$,n_vec). For each query cycle we assume that the 3-dimensional *pointshell* points are mapped to the corresponding $zval$ values associated with the global voxel grid of our static environment. In the projection part of the SQL statement, we use the functions $force\_x()$, $force\_y()$ and $force\_z()$ which are user-defined aggregate functions as provided in the SQL:1999 standard. These functions collect all points of the *pointshell* which intersect any object voxel stored in the *index table* and return the corresponding aggregated contact force vector separated into the x, y and z dimension components. The exist-quantifier used in our SQL statement is necessary to filter out any redundant results.

7.6 Accelerated Query Processing

Our approach aims at reducing the total query cost associated with the built-in B^+-tree. If we assume that our *pointshell* consists of n points, the simple query process as proposed in Section 7.5 leads to n point queries on the *index table*. Thus, we have to navigate n times through the built-in index (B^+-tree) directory. The general idea of our approach is to apply the statistic based query techniques, as presented in the previous chapter, on top of the SQL interface in order to minimize the overall navigational cost of the built-in index. Based on the cost model, we group the n query points into m query ranges, where $m \ll n$ holds. For each query range, the built-in index directory has to be traversed only once, and the corresponding results can be read by a single scan on the leaf level of the built-in index.

At first, we formally introduce two different possibilities to carry out a *pointshell* query, the *point query sequence* and the *range query sequence*. We also consider the cost associated with both query types. In the following, we assume that all points of the pointshell are transformed to the corresponding *zval* values. We start with a straightforward approach of a *pointshell* query, the *point query sequence*.

7.6.1 Point Query Sequence

The *pointshell* query Q leads to an ordered sequence $Seq_{Q,I} = (\langle p_1, ..., p_n \rangle)$ of query points on the index table I, where $p_i < p_{i+1}$ for all $i \in 1, ..., n-1$. For $Seq_{Q,I}$ the following assumptions hold:

- The elements r_i stored in the index are of the same type as p_i. Furthermore, we assume that the elements r_i can be regarded as a linear ordered list $L(I) = \langle r_1, ..., r_N \rangle$ for which $r_1 \leq ... \leq r_N$ holds.
- We assume that the disk blocks b_i of the index obey a linear ordering and fulfill the following property: $r' \leq r'' \Leftrightarrow b(r') \leq b(r'')$, where $b(r)$ denotes the disk block of the index I which contains the entry r.

7.6.2 Range Query Sequence

Consecutive query points of the *point query sequence* $Seq_{Q,I} = (\langle p_1, ..., p_n \rangle)$ can be grouped into subsequences $\langle seq_1, ..., seq_m \rangle$ of the form

$$(\langle \langle p_1, ..., p_{l_1} \rangle, \langle p_{l_1+1}, ..., p_{l_2} \rangle, ..., \langle p_{l_{m-1}+1}, ..., p_n \rangle \rangle)$$

where $l_{i-1} < l_i$ and $1 \leq l_i \leq n$ for all $i \in 1, .., m$, $m < n$. Each subsequence seq_i builds the new query range s_i, which is bounded by the first and last point of seq_i. Thus, the large amount of point queries is reduced to a small sequence $RSeq_{Q,I}$ of query ranges associated with the *pointshell* Q and index table I. When carrying out a range query $s = (p_u, p_v)$ derived from the subsequence $\langle p_u, ..., p_v \rangle$, we traverse the index directory only once and perform a range scan (p_u, p_v) on the leaf-level. Thereby, we read false hits from the index table I which have to be filtered out in a subsequent refinement step.

7.6.3 Cost Based Grouping

For each haptic query, there are a lot of different possibilities to group the *pointshell* points into a range query sequence. We can apply statistics related to the build-in index and the corresponding cost model as presented in Section 6.3 in order to find a cost optimum grouping for the *pointshell* points.

Unfortunately, there are exponentially many grouping possibilities which results in an exponential runtime $O(2^n)$ of an optimum cost-based grouping algorithm, where n denotes the number of *pointshell* points. In this section, we will present an algorithm with a guaranteed worst-case runtime complexity of $O(n)$ which produces a cost optimal range query sequence, helping to accelerate the query process considerably. Our approach closes the gaps between two query points if and only if the estimated cost related to the additional read data is smaller than the estimated navigational cost related to an additional point query following the extended index range scan approach of Section 6.3.2. The grouping algorithm *CBGroup* is presented in Figure 7.7.

Let us note that our cost based grouping method reports the cost optimal

```
CBGroup(PointShell PS, QuantileVector Q){
    Integer left := compute_zvalue(PS.point[1]);
    Integer right := left;
    Integer next := 0;
    Float navcost := estimate_navigational_cost();
    for i=2 to PS.size() do {
      next := compute_zvalue(PS.point[i]);
      if estimate_gap_cost(right,next,Q) ¿ navcost then {
        report(left,right);
        left := next;
      }
      right := next;
    }
    report(left,right);
}
```

Figure 7.7: PointShell Grouping Algorithm *CBGroup*.

solution w.r.t. our selectivity estimation method (cf. Section 6.3.2) because each modification of the reported sequence would increase the overall query cost.

7.6.4 Accelerated *SQL* Query

In order to apply our cost optimal query method on top of the *SQL* engine, we have to rewrite the *SQL* statement of Section 7.5 as depicted in Figure 7.8.

Now, the input of the nested SQL query is the *index table I* and a collection of *zval* intervals covering the pointshell points as transient table *POINTSHELL_INTERVALS* which obey the following schema (*id*, *lower*, *upper*, *exactPointList*). In each query cycle we retrieve the voxels from *I* which are covered by a *zval* interval in a filter step. Subsequently, we collect the voxels which match the *zval* values of the pointshell points covered by the *zval* interval in a refinement step by means of the function *test_Exact()*.

```
SELECT sum(X), sum(Y), sum(Z)
FROM (
   SELECT agg_force_x(PSI,zval_tab) AS X,
          agg_force_y(PSI,zval_tab) AS Y,
          agg_force_z(PSI,zval_tab) AS Z
   FROM :POINTSHELL_INTERVALS PSI,
        TABLE( SELECT DISTINCT I.zval
               FROM INDEX_TABLE I
               WHERE I.zval BETWEEN PSI.lower AND PSI.upper
                 AND test_Exact(I.zval,PSI.exactPointList)) zval_tab
)
```

Figure 7.8: Accelerated SQL statement for haptic query processing

The resulting collection of voxels and the corresponding *zval* interval are delegated to the aggregate functions *agg_force_x()*, *agg_force_y()* and *agg_force_z()* separately computing the force vector component in each dimension.

7.7 Performance Evaluation

In this section, we evaluate the performance of our approach with a special emphasis on the haptic frame rate. The tests are based on a real-world test dataset *CAR* which was provided by our industrial partner, a German car manufacturer, in form of high resolution voxelized 3-dimensional CAD parts. Table 7.1 shows the properties of this dataset.

dataset	#voxels	#objects	size of data space
CAR	$14 \cdot 10^6$	200	2^{33} cells

Table 7.1: Datasets.

The query processing functionality of our approach is implemented on top of the Oracle9i Server using PL/SQL for the computational main memory based programming. All experiments were performed on a Pentium 4/2600 machine with IDE hard drives. The database block cache was set to 500

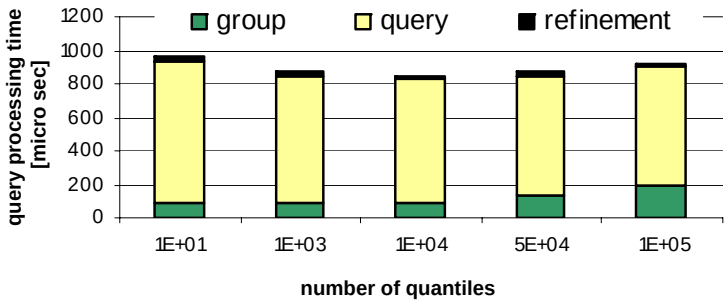


Figure 7.9: Avg. query processing time for different quantile resolutions.

disk blocks with a block size of 8 KB and was used exclusively by one active session.

In the following, we examine the benefits of using range query sequences instead of point query sequences. In order to point out the difference of the data access cost between the simple query and the accelerated query, we removed the aggregate functions from both SQL statements in our experiments. We carried out several *pointshell* queries at different locations and logged the average response times. The query *pointshell* consists of about 300 points.

In a first experiment, we investigated how different resolutions of the quantile vector influence the performance of our approach (cf. Figure 7.9). A low quantile resolution leads to a cheap grouping, but badly estimates the query selectivity. Contrary, if we choose a high quantile resolution, the query is well adjusted to the respective selectivity at the expense of the grouping performance. In our experiments we achieved the best results with a quantile resolution of about 10,000.

Throughout the following experiments we used the best possible quantile resolution of 10,000 for our $CBGroup$ algorithm. Furthermore, we take a comparison approach for generating range scans which does not use any statistical information at all. The comparison algorithm, called $MaxGap$ approach, groups query points into a range query in which the gap between two adjacent query points does not exceed a specified $MAXGAP$ parameter. Note that the setting $MAXGAP = 0$ corresponds to the original point query

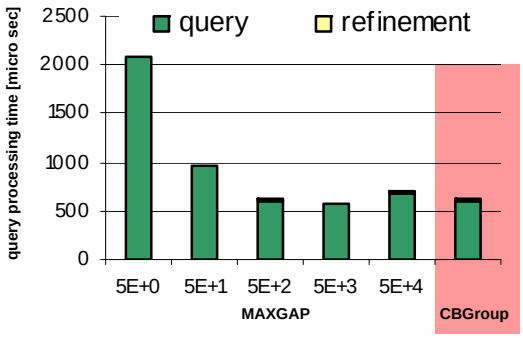
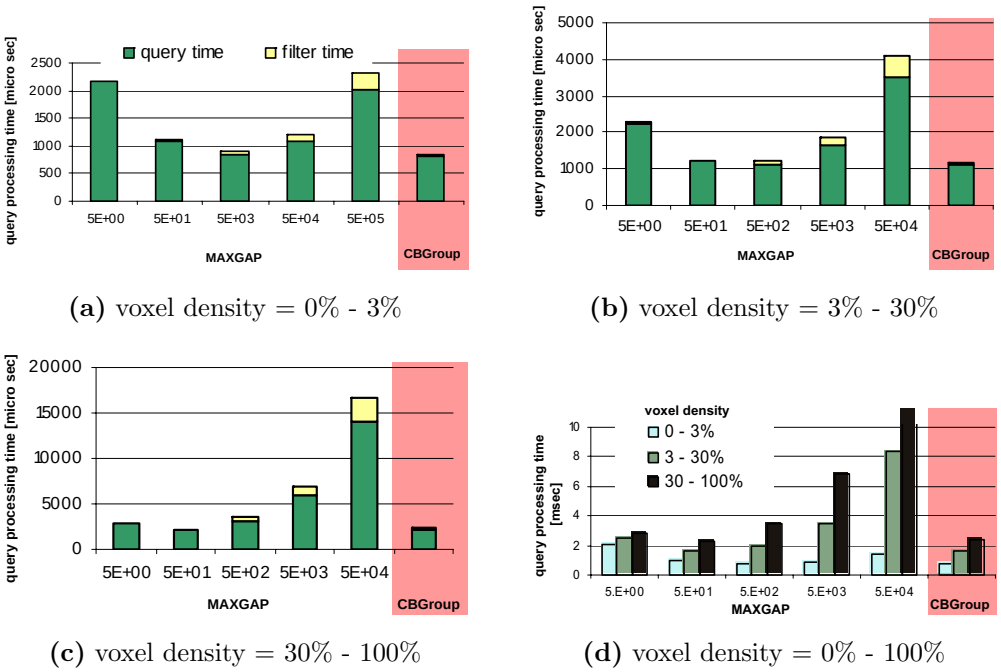


Figure 7.10: Performance of range query sequences.

sequence.

The next experiment compares the runtime of our cost-optimum $CBGroup$ algorithm which corresponds to the best possible runtime achieved by the $MaxGap$ approach. By varying the $MAXGAP$ parameter, we can find a good trade-off between navigational overhead and filter performance, i.e. the number of query ranges should be small while keeping the result size of the filter low. Figure 7.10 shows that using low $MAXGAP$ values results in a low query performance. This is caused by a substantial navigational overhead of the index because we have to carry out many range queries. On the other hand, high $MAXGAP$ values result in a very low filter performance which leads to high I/O and refinement cost. We can observe that a good trade-off between the navigational overhead and filter performance is achieved if we use a $MAXGAP$ value of 500. The results presented in Figure 7.10 also show that our $CBGroup$ approach adapts well to the data characteristics and achieves a performance which is comparable to the $MaxGap$ approach when the best $MAXGAP$ parameter is chosen.

In the last experiment (cf. Figure 7.11), we performed haptic queries in regions having different voxel density. The voxel density denotes the ratio of object voxels to free space voxels. In areas where the voxel density is low, a high $MAXGAP$ value performs best. Although the query ranges are large, which results in low navigational cost, the result set stays small due to low voxel density. With increasing voxel density, high $MAXGAP$ values result in many false hits, leading to high I/O and refinement cost. For all



(a) voxel density = 0% - 3%

(b) voxel density = 3% - 30%

(c) voxel density = 30% - 100%

(d) voxel density = 0% - 100%

Figure 7.11: Average query performance dependent on the voxel density.

voxel density settings, our cost-based grouping algorithm adapts to the voxel density automatically as shown in Figure 7.11d. Independent of the voxel density, CBGroup achieves a performance that is very close to the respective best MAXGAP parameter which in fact differs for varying voxel densities. Obviously, the *CBGroup* algorithm locally adapts the grouping to the data distribution.

7.8 Summary

The experiments show that the approved main memory Voxmap-PointShell method can be indeed successfully externalized and integrated into a fully-fledged database management system. This application clearly demonstrates the advantage of our cost-optimum access method which groups different independent point queries together to larger range scans. In a broad experimental evaluation based on a real-world test dataset, we have demonstrated

the enormous acceleration of the query process. We obtain even a frame rate of the haptic rendering loop which is sufficient for many haptic rendering applications [MPT99].

Chapter 8

Cost-Based Approximation of Complex Spatial Objects

In the previous chapters, we have presented efficient solutions for spatial collision queries on voxelized objects. The proposed methods are based on adjusting the query to the characteristics of the distribution of the datasets by means of simple statistics. In this chapter, we propose to use similar statistical information in order to reduce the complexity of the database objects by building more adequate object approximations. The concept proposed in this chapter is published in [KKPR04a].

Three-dimensional CAD applications require efficient and scalable database solutions to cope with rapidly growing amounts of dynamic data. Such applications include the digital mock-up of vehicles and airplanes, virtual reality applications or haptic simulations in virtual product environments (cf. Chapter 7). For instance, the "777" from Boeing was completely digitally designed and assembled. It consists of about three million parts, having very complex shapes.

We assume that the spatial objects are conservatively be approximated by a set of voxels, i.e. cells of a grid covering the complete data space. This is a common and successful approach since spatial selection queries can be easily expressed as an intersection of the corresponding voxel sets. However,

for high resolution objects the voxel sets representation can result in a very large amount of primitives which have to be managed and can become very expensive w.r.t. memory space and indexing cost. For example, some parts of the "777" from Boeing are composed of several millions of voxels. Although the voxels can further be grouped together to intervals, the number of the resulting voxel intervals still remains very high. In order to overcome the high cardinality of the voxel based representation which makes an efficient processing of the objects difficult, we introduce a cost-based decomposition algorithm for linearized high-resolution spatial objects. This decomposition helps to range between the two extremes of one-value approximations and the use of unreasonably many approximations. Our approach takes compression algorithms for the effective storage of decomposed interval representations of spatial objects and access probabilities of these decompositions into account. This work has been published in [KKPR04a].

8.1 Related Work

First, we will shortly discuss different aspects related to an effective decomposition of complex-structured spatial objects for efficient query processing.

Approximations of extended objects generally consist of either one or several simple spatial primitives such as minimal bounding boxes which are often used for one-value approximations [BKSS94][GG98]. Although providing the minimal storage complexity, one-value approximations of spatially extended objects often are far too coarse. This is in particular the fact for many spatial applications where GIS or CAD objects feature a very complex and fine-grained geometry.

In contrast, approaches which use multi-value approximations, i.e. approximations which are composed of several spatial primitives, can achieve a better approximation than a single rectangle. In the case of a very accurate approximation, the number of primitives can become very high. For instance, Gaede [Gae95] pointed out that the number of z-value intervals representing a spatially extended object exponentially depends on the granularity of the

grid approximation. Furthermore, the extensive analysis given in [MJFS96] and [FJM97] shows that the asymptotic redundancy of an interval-based decomposition is proportional to the surface of the approximated object. Thus, in the case of highly resolved huge parts, e.g. wings of an airplane, the number of intervals can become unreasonably high which results in too many intersect verifications for any spatial query procedure.

A promising solution for a good trade-off between these conflicting objectives may be found somewhere in between one-value and multi-value object approximations. 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 its effect on efficient query processing. The presented empirically derived root-criterion suggests to decompose a polygon consisting of n vertices into $O(\sqrt{n})$ many simple approximations. 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 chapter, in contrast, we will present an analytical cost-based decomposition approach which can be used for 2D and 3D objects. It takes the cost of the multi-step query processing method, i.e. the filter and refinement cost, into account.

8.2 Approximation of Rasterized Spatial Objects

Objects having a very complex structured shape often need to be represented with high resolution in order to pattern their complex structure as accurate as possible. We assume objects which are rasterized with a given resolution as depicted in Figure 8.1.

Definition 8.1 (rasterized objects) *Let O be the domain of all object identifiers and let $id \in O$ be an object identifier. Furthermore, let N^d be the domain of d-dimensional points. Then, we call a pair $O_{voxel} = (id, \nu_1, ..., \nu_n) \in$*

$O \times 2^{\mathrm{N}^d}$ *a d-dimensional rasterized object. We call each of the* ν_i *an object voxel, where* $i \in 1, ..., n$.

Generally, a rasterized d-dimensional object (cf. Figure 8.1(b)) consists of a set of d-dimensional points which can be naturally ordered in the one-dimensional case. If d is greater than 1, such an ordering does not longer exist. By means of space filling curves $\rho : \mathrm{N}^d \rightarrow \mathrm{N}$, all multidimensional rasterized objects are mapped to a set of integers (cf. Figure 8.1(c)). As a principal design goal, space filling curves achieve good spatial clustering properties since voxels in close spatial proximity are encoded by contiguous integers which can be grouped together to intervals. Examples for space filling curves include the lexicographic-, Z- or Hilbert-order, whereas with the Hilbert-order the least intervals per object are generated [FR89][Jag90], but it is also the most complex linear ordering. As a good trade-off between redundancy and complexity, we use the Z-order throughout this chapter.

In form of *voxel sets*, high resolution spatial objects may consist of several hundreds of thousands of voxels. Applying space filling curves in order to linearize the voxelized objects, we achieve a sequence of intervals which is marginally smaller. For instance, one of our engineering test datasets composes about 18 million voxels results in about 9 million intervals using the z-order based linearization. The resulting sequence of intervals, representing a high resolution spatially extended object, often consists of very short intervals connected by short gaps. Experiments suggest that both gaps and intervals obey an exponential distribution [KPPS03a]. In order to overcome this obstacle, it seems promising to pass over some "small" gaps in order to obtain much less intervals, which we call *interval container*.

In the following, the geometry of a spatial object is assumed to be described by a sequence of voxels.

8.2.1 Interval Container

An interval container is a covering of one or more ρ-order-values, i.e. integer values resulting from the application of a space filling curve ρ to a rasterized

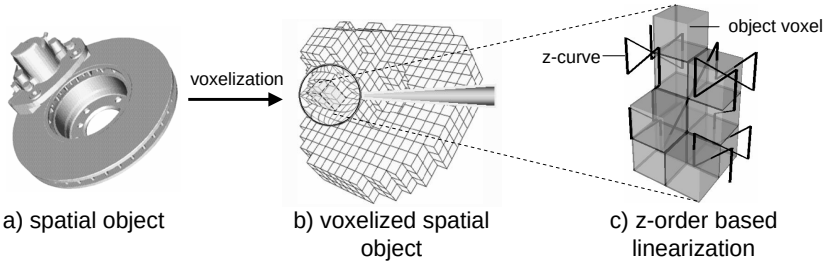


Figure 8.1: Voxelized spatial object

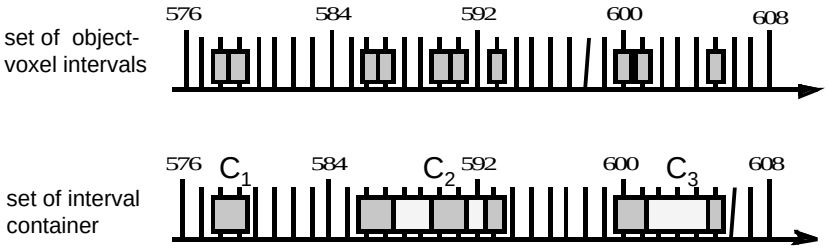


Figure 8.2: Interval container sequence

object $(id, \nu_1, ..., \nu_n)$, where the interval container may contain integer values which are not in the set $\{\rho(\nu_1), ..., \rho(\nu_n)\}$ (cf. Figure 8.2).

Definition 8.2 (interval container) *Let $(id, \nu_1, ..., \nu_n)$ be a rasterized object and $\rho : \mathrm{N}^d \rightarrow \mathrm{N}$ be a space filling curve. Furthermore, let $W = \{(l, u), l \leq u\} \subset \mathrm{N}^2$ be the domain of intervals and let $b_1 = (l_1, u_1), ..., b_n = (l_n, u_n) \in W$ be a sequence of intervals with $u_i + 1 < l_{i+1}$, representing the set $\rho(\nu_1), ..., \rho(\nu_n)$. Moreover, let $m \leq n$ and let $i_0, i_1, i_2, ..., i_m \in \mathrm{N}$ such that $0 = i_0 < i_1 < i_2 < ... < i_m = n$ holds. Then, we call $O_{cont} = (id, \langle\langle b_{i_0+1}, ..., b_{i_1}\rangle, ..., \langle b_{i_m-1}, ..., b_{i_m}\rangle\rangle)$ an interval container sequence of cardinality m, and each of the $j = 1, ..., m$ groups $I_j = \langle b_{i_{j-1}+1}, ..., b_{i_j}\rangle$ of O_{cont} an interval container.*

Intuitively, an *interval container* covers consecutive z-value intervals, where there is at least one gap of one z-value between adjacent intervals. Next, we will define some useful operators on interval containers which we use throughout this thesis. Table 8.1 summarizes the operators for the interval

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

Table 8.1: Operators on interval containers.

container $C_I = \langle (l_r, u_r), ..., (l_s, u_s) \rangle$. The byte sequence $B(C_I)$ corresponds to a sequence of voxels within $H(C_I)$ along the space filling curve ρ, transformed into a bit sequence of length $L(C_I)$, where "1" denotes an object voxel and "0" denotes free space voxels. Table 8.2 demonstrates the values of these operators for the interval containers in our example shown in Figure 8.2. Note that the operators defined on interval containers can naturally also be applied to simple intervals. Let $b = (l, u)$ be an interval, then $H(b) = b$, $L(b) = u - l + 1$, $D(b) = 1$, $G(b) = 0$ and $B(b) =' 1_b'^{L(b)}$.

In the next section, we discuss how the I/O cost required to load the exact content of the interval containers, i.e. the byte sequence $(B(C_I))$, can be drastically reduced by applying compression techniques.

Operators	C_1	C_2	C_3
$H(C_x)$	[578,579]	[586,593]	[600,605]
$L(C_x)$	2	8	6
$D(C_x)$	1	$\frac{5}{8}$	$\frac{3}{6}$
$G(C_x)$	0	2	3
$B(C_x)$	00110000_b	$00110011\ 01000000_b$	11000100_b

Table 8.2: Operators for the interval containers C_1, C_2 and C_3 of the example given in Figure 8.2.

8.2.2 Compression of Interval Containers

In this section, we motivate the use of packers by showing that $B(C_I)$ contains patterns. Therefore, $B(C_I)$ can efficiently be shrunk by using data compressors. Furthermore, we discuss the properties which a suitable compression algorithm should fulfill. In the following, we give a brief presentation of a new effective packer. It exploits gaps and patterns included in the byte sequence $B(C_I)$ of our interval container C_I.

Patterns. To describe a rectangle in a 2D vector space, we only need four numerical values, e.g. two 2-dimensional points. In contrast to the vector representation, an enormous redundancy might be contained in the corresponding voxel sequence of an object. An example is shown in Figure 8.3. As space filling curves, in particular the Z-order, enumerate the data space in a structured way. Such "structures" can be found 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 interval, i.e. $O_{cont} = (id, \langle C_I \rangle)$, and survey its byte representation $B(C_I)$ in a hex-editor. We can notice that some byte sequences occur repeatedly. For more details about the existence of patterns in $B(C_I)$ we refer the reader to [Kun02]. We will now discuss how these patterns can be used for the efficient storage of interval containers.

Compression Rules. A voxel set belonging to an interval container C_I

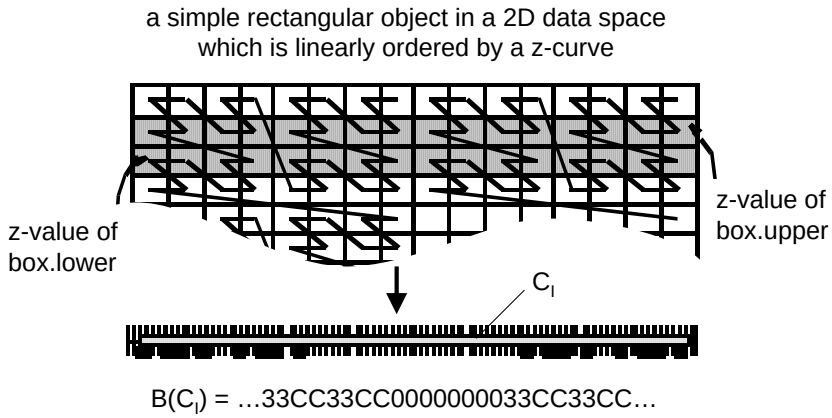


Figure 8.3: Pattern derivation by linearizing a rasterized object using a space-filling curve (Z-order).

can be materialized in many different ways. A good materialization should consider two "compression rules":

- Least possible secondary storage should be occupied.
- Least possible time should be needed for the (de)compression of $B(C_I)$.

A good query response behavior is based on the fulfillment of both aspects. The first rule guarantees that the I/O cost is relatively small whereas the second rule is responsible for low CPU cost. A good behavior related to an efficient retrieval and evaluation of $B(C_I)$ depends on the fulfillment of both rules.

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

Spatial compression techniques. In this section, we look at a new specific compression technique which we call *Quick Spatial Data Compressor* ($QSDC$). It is designed for storing the byte sequence of an interval container in a compressed way. According to our experiments, the new data compressor outperforms popular standard data compressors such as $BZIP2$ [BW94].

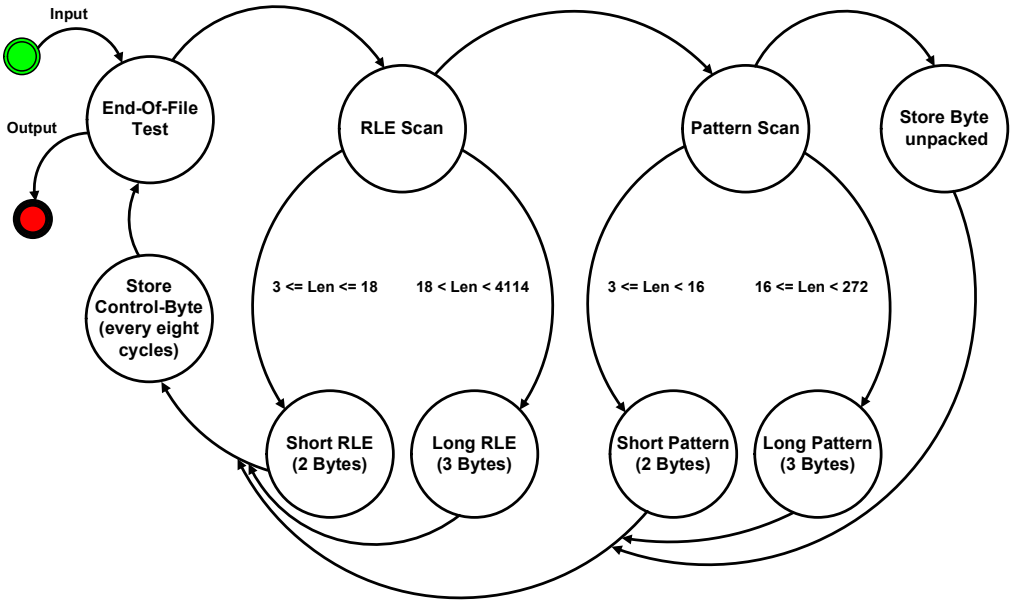


Figure 8.4: Flow diagram of QSDC compression algorithm.

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 the Huffman compression does. *QSDC* is a derivation of the *ZLIB* technique [LZ77]. However, it compresses data in only one pass and much faster than other Lempel-Ziv based compression schemes.

QSDC operates on two main memory buffers. The compressor scans an input buffer for patterns and gaps (cf. Figure 8.4). It replaces the patterns with a two- or three-byte compression code and the gaps with a one- or two-byte compression code. Then it writes the code to an output buffer. *QSDC* packs an entire byte sequence 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.

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. For more details we refer the reader to [Kun02], where it was shown that $QSDC$ is more suitable for spatial query processing than $ZLIB$ [LZ77] due to the higher (un)pack speed and an almost as high compression ratio.

8.3 Cost-Based Approximation

Now, we will show how to build interval containers in a reasonable way, trying to speed up spatial intersection queries.

8.3.1 Grouping Rules

For each object, there exist a lot of different possibilities to group the voxel intervals into interval containers. The question 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 containers should be small.
- The approximation error of the interval containers should be small.
- Interval containers should allow an efficient evaluation of the contained voxel intervals.

The first rule assures that the number of index entries is small, as the conservative representations of the interval containers are stored in appropriate index structures, e.g. the RI-tree. The second rule guarantees that many unnecessary candidate tests, i.e. false hits, can be omitted, as the number and size of gaps included in the interval containers is small. Finally, the third rule ensures that a candidate can be refined efficiently. A good query

response behavior results from an optimum trade-off between these grouping rules.

A common approach is to determine the optimal grouping solution with respect to accuracy and redundancy of the resulting approximation of the actually considered object. However, other factors like "regions which are frequently covered by queries" could be very valuable to build promising approximations. Our grouping algorithm is based on the following parameters:

- An average query distribution function QDF which is used to compute the access probability $P(C_I, QDF)$ of an interval container C_I.
- The cost model related to the used index structure CST_{IDX}.
- The cost model related to the evaluation of the compressed container content CST_{PAC}.

8.3.2 Query Distribution Function QDF

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 Tokyo, are much more frequently inquired than less dense areas. Furthermore, often small selective queries are posted. From the grouping point of view, it would be unadvisable to allow large gaps within interval containers covering such frequently asked regions.

The assumed *query distribution function* QDF can be successfully applied to our grouping algorithm. First, we transform the interval queries into a two-dimensional normalized data space $D := \{(l, u) \in [0, 1]^2 : l \leq u\}$, an example for two query intervals $Q_1 = (l_1, u_1)$ and $Q_2 = (l_2, u_2)$ is shown in Figure 8.5. We start with normalizing the coordinates of our query intervals to ensure that all data lies within the two-dimensional cuboid. Therefore, an interval $Q = [l, u]$ corresponds to the point (l, u) with $l \leq u$ (cf. Figure 8.5). To each of these two-dimensional points $Q = (l, u)$, we assign a numerical

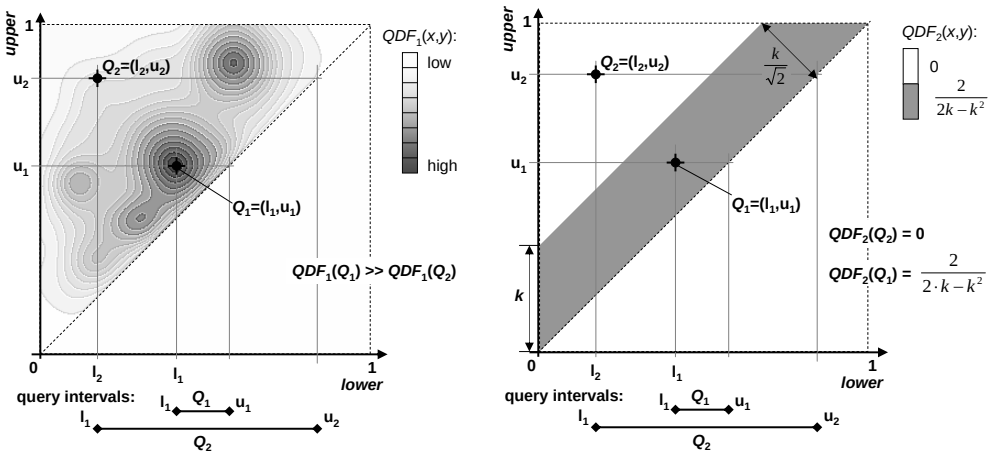


Figure 8.5: Query Distribution Functions $QDF(x, y)$.

value $QDF(Q)$, where $0 \leq QDF(Q) \leq 1$ holds and which has the following property:

$$\forall (a,b) \in [0,1]^2, a \leq b : P((l,u)) = \int_{l=a}^{l=b} \int_{u=l}^{u=b} QDF((l,u))\, du\, dl,$$

where $P(Q)$ denotes the probability that the query interval Q is completely covered by the interval (a, b), i.e. $0 \leq a \leq l \leq u \leq b \leq 1$. 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 8.5 shows two different query distribution functions. A potential query Q_2 is very unlikely in Figure 8.5(a) and does not occur at all in Figure 8.5(b). On the other hand, query Q_1 is very likely in both cases.

In the following, we assume a very simple query distribution function as depicted in Figure 8.5(b). 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

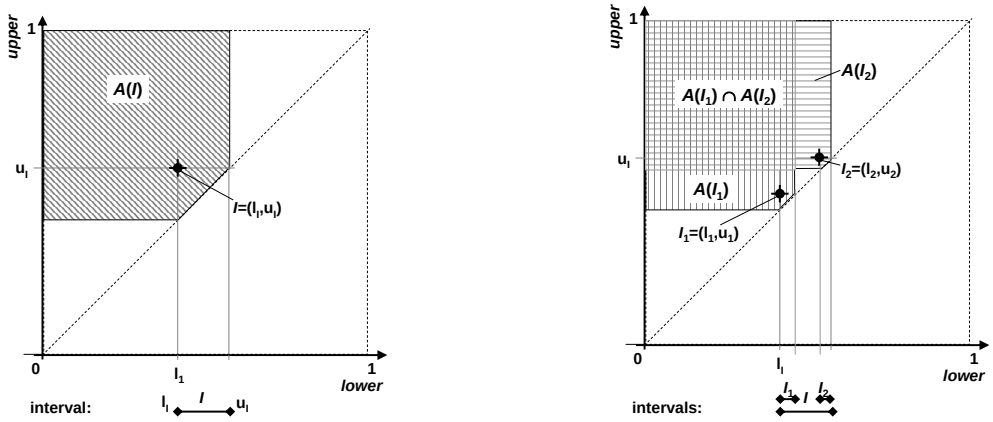


Figure 8.6: Computation of access probabilities of interval containers.

the possible query objects. For the computation of the access probability, we only consider query intervals whose extensions do not exceed k. Let us denote the restricted area D^*. As we assume that the query intervals are equally distributed within the restricted area D^*, the function $QDF(l, u)$ is equal to $\frac{2}{2 \cdot k - k^2}$ for all positions $(l, u) \in D^*$ and otherwise zero. In our experiments, we used this simple query distribution function which is specified by the parameter k.

8.3.3 Access Probability

The access probability $P_{acc}(I)$ related to an interval $I = H(C_I)$ of an interval container C_I denotes the probability that an arbitrary query object has an intersection with the interval $I = (l_I, u_I)$. All possible query intervals that intersect an interval I_1 are visualized by the shaded area $A(I)$ in Figure 8.6. The area displays all intervals whose lower bounds are smaller or equal to u_I and whose upper bounds are larger or equal to l_I. These query intervals are exactly the ones that have a non-empty intersection with I. The probability that the interval I is intersected by an arbitrary query interval is

$$P_{acc}(I) = \int_{l \in A(I)} \int_{u \in A(I)} QDF((l, u))\, du\, dl.$$

8.3.4 Cost Model

In order to evaluate the compressed information of an intersected interval container C_I, we have to retrieve it from disk first. Thus, the expected access cost consists of two parts:

- the access probability and
- the cost of the evaluation.

The evaluation cost heavily depend on the used compression algorithm. The compression ratio mainly influences the I/O cost, i.e. the cost required to load the compressed information from disk. Furthermore, we have to take the CPU cost required to decompress the information into account which also depends on the used compression algorithm. For each compression algorithm we provide statistics, i.e. a look-up table LUT, by means of which we estimate the I/O cost and CPU cost related to an evaluation of an interval container. Roughly speaking, the evaluation cost depends on the length of our container interval $H(C_I)$ and on the used packer. Figure 8.7 depicts an empirically derived look-up table for two packers. Let $c_{load}^{I/O}$ be a factor relating to the I/O cost, c_{decomp}^{cpu} be a factor relating to the CPU cost of the decompression and let c_{test}^{cpu} be a factor relating to the CPU cost of the intersection test. Then, the evaluation cost $cost_{eval}$ can be estimated by the following formula:

$$cost_{eval} = |C_I|_{compr} \cdot c_{load}^{I/O} + |C_I| \cdot (c_{decomp}^{cpu} + c_{test}^{cpu}),$$

where $|C_I|_{compr}$ denotes the size of the compressed information of C_I and $|C_I|$ denotes the size of the uncompressed information of C_I.

To sum up, the expected access cost related to an interval container object C_I can be computed as follows:

$$cost(C_I) = P(C_I) \cdot cost_{eval}(C_I, LUT).$$

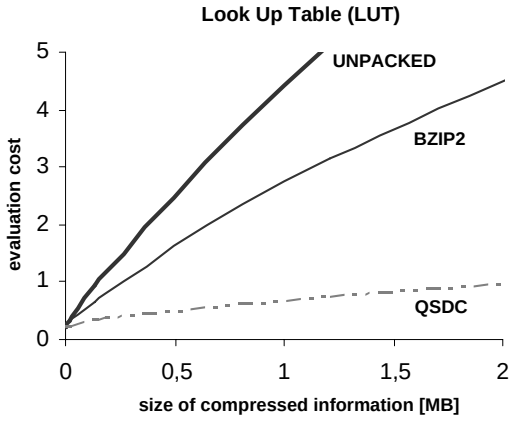


Figure 8.7: Look-Up table for different Packers

8.3.5 Decomposition Algorithm

Orenstein [Ore89] introduced the size- and error-bound decomposition approach. Our first grouping rule "the number of interval containers 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 containers should be small". For fulfilling both rules, we introduce the following top-down decomposition algorithm for interval containers, called *CoDec* (cf. Figure 8.8). *CoDec* is a recursive algorithm which starts with an approximation $O = (id, \langle C_I \rangle)$, i.e. we approximate the interval sequence object by one single interval. In each step of our algorithm, we look for the gap g within the interval approximation C_I which seems most promising to split C_I at this gap into two interval containers C_{left} and C_{right}, incorporating the query-distribution function QDF. From all possible splits[1], we choose the one that yields the highest decrease in the cost according to our cost model. We carry out the split along this gap if the average query cost caused by the decomposed interval containers is smaller than the cost caused by our input interval container C_I. The interval containers which are reported by the *CoDec* algorithm are stored in the database and no longer taken into account in further recursion steps. Data compressors which have a shallow

[1]We only try to split at real gaps. Splitting an object interval would obviously not yield any performance gain.

```
LUT: look-up table with packer specific cost

CoDec(C_I, QDF, LUT){
   cost_comp := P(C_I, QDF) · cost_eval(C_I, LUT);
   container_pair := split(C_I,QDF);
   C_left := container_pair.left;
   C_right := container_pair.right;
   cost_dec := P(C_left, QDF) · cost_eval(C_left, LUT) +
               P(C_right, QDF) · cost_eval(C_right, LUT);
   if cost_comp > cost_dec then
      CoDec(C_left,QDF,LUT);
      CoDec(C_right,QDF,LUT);
   else
      report(C_I);
   end if;
}
```

Figure 8.8: Decomposition Algorithm *CoDec*.

LUT curve, as e.g. *PACKER2* in Figure 8.7, result in an early stop of the *CoDec* algorithm, generating a small number of interval containers.

Our experimental evaluations suggest that the above grouping algorithm yields results which are very close to an optimal decomposition for many data compression techniques and data space resolutions.

8.4 Intersection Detection Based on Interval Containers

We first present two rather obvious lemmas which state whether two interval containers intersect or not, based on relatively little information. The first lemma can be used as a filter for detecting non-intersecting interval sequence objects.

We speak of "overlapping" interval containers if their hulls intersect. Respectively, if two overlapping interval containers C_I and $C_{I'}$ contain at least one interval which intersect, i.e.

$$\exists b \in C_I, \exists b' \in C_{I'} : intersect(b, b'),$$

we speak of "intersecting" interval containers.

Lemma 8.1 (non-intersecting interval containers) *Let $C_I = \langle b_1, .., b_n \rangle$ and $C_{I'} = \langle b'_1, .., b'_{n'} \rangle$ be two interval containers. Then, the following statement holds:*

$$\neg overlap(C_I, C_{I'}) \Rightarrow \neg intersect(C_I, C_{I'}).$$

Proof. *At first we convert the statement as follows:*

$$\neg overlap(C_I, C_{I'}) \Rightarrow \neg intersect(C_I, C_{I'}) \Leftrightarrow intersect(C_I, C_{I'}) \Rightarrow overlap(C_I, C_{I'}).$$

Then,

$$intersect(C_I, C_{I'}) \Rightarrow \exists (b_i, b'_j) \in b_1, .., b_n \times b'_1, .., b'_{n'} : intersect(b_i, b'_j) = true.$$

Let $b_i = (l_i, u_i)$, $b'_j = (l'_j, u'_j)$, $H(C_I) = (l, u)$ and $H(C_{I'}) = (l', u')$, then follows Definition 4.3:

$$l \leq l_i \leq u'_j \leq u' \wedge l' \leq l'_j \leq u_i \leq u$$

$$\Rightarrow l \leq u' \wedge l' \leq u \Rightarrow intersect((l, u), (l', u'))$$

holds which proves that C_I and C'_I overlaps. □

Let us note that we cannot pinpoint any intersecting interval sequence objects by means of Lemma 8.1, as

$$overlap(C_I, C'_I) \Rightarrow intersect(C_I, C'_I)$$

does not hold. Thus, a refined evaluation of the intersect predicate is necessary when two interval containers overlap. However, in the case where both interval containers have maximum density, we can do without this refinement step.

Lemma 8.2 (intersecting interval containers) *Let* $C_I = \langle b_1, .., b_n \rangle$ *and* $C_{I'} = \langle b'_1, .., b'_{n'} \rangle$ *be two interval containers. Then, the following statement holds:*

$$(D(C_I) = 1 \wedge D(C_{I'}) = 1 \wedge overlap(C_I, C_{I'})) \Rightarrow intersect(C_I, C_{I'}).$$

Proof. *According to Definition 8.2 and the definitions of the operators given in Table 8.1:* $D(C_I) = 1 \Rightarrow n = 1$ *and* $D(C'_I) = 1 \Rightarrow n' = 1$. *Then, the following statement holds:*

$$overlap(C_I, C_{I'}) \Rightarrow intersect(b_1, b'_1) \Rightarrow intersect(C_I, C_{I'}).$$

□

Lemma 8.2 shows that we can sometimes pinpoint whether two interval-container pairs intersect based on relatively little information. Unfortunately, as in most cases the precondition of Lemma 8.2 does not hold, we will not be able to apply it very often. Nevertheless, there are some more information given by the interval container operators (cf. Table 8.1). In the next section, we will explain how these operators can be successfully used to detect intersecting interval containers without accessing the covered intervals more often.

8.5 Fast Intersection Test for Interval Containers

For the intersect predicate, it suffices to find a single interval of a database object which intersects an interval of the query object in order to report the database object. Obviously, it is desirable to detect such intersecting interval pairs as early as possible in order to avoid unnecessary refinement tests. In this section, we present an optimization aiming at this goal. As we assume that the query objects are approximated in the same way as the database objects, the determination of intersecting interval pairs approximated by

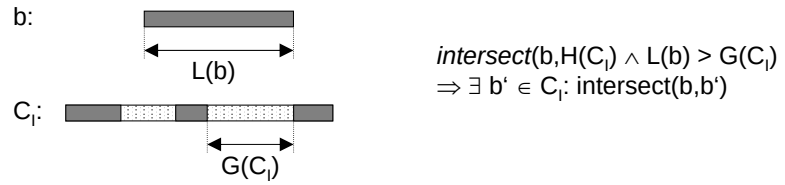
interval containers without examining the exact interval information can be successfully used as fast second filter step. If this filter step determines an intersecting interval container pair $(C_I, C_{I'})$, i.e. $H(C_I)$ *intersects* $H(C_{I'})$, all remaining candidate tests of the database object can obviously be skipped. Thus, this filter step acts as an additional filter between the first filter step and the refinement step.

In the following, we present the fast intersection test which is entirely based on aggregated information of the interval containers.

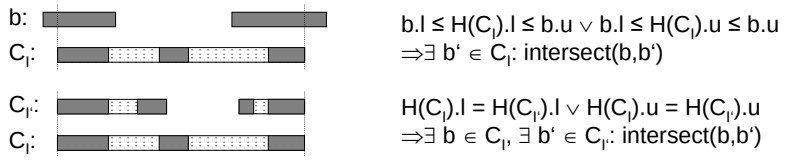
8.5.1 Fast Intersection Tests

We will now discuss in which cases we can decide whether two overlapping interval containers C_I and $C_{I'}$ intersect each other without accessing their exact content. Let us note that an interval container C_I with density equal to 1 exactly represents the single interval b contained in C_I. Consequently, when we use interval $b \in C_I$ instead of C_I, we assume that $D(C_I) = 1$. If any of the following three conditions holds, then two interval containers intersect:

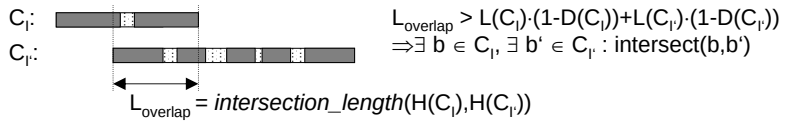
- If the interval b (or the interval container C_I with density of 1) is longer than the maximum gap between two intervals contained in the other interval container $C_{I'}$, then the two interval containers intersect (cf. Figure 8.9(a)).
- If the interval b (or an interval container C_I with density of 1) overlaps the start or end point of the other interval container $C_{I'}$, then the interval containers intersect. This is due to the fact that any interval container ends and starts with an interval (cf. Figure 8.9(b)). Furthermore, if both interval containers start or end at the same point, then they intersect.
- If the summarized gap lengths of both interval containers exceeds the length of their overlapping, then the two interval containers necessarily intersect. The length of the summarized gaps of an interval container C_I can be easily computed by $L(C_I) \cdot (1 - D(C_I))$ (cf. Figure 8.9(c)).



(a) detection based on maximal gap



(b) detection based on starting and end points



(c) detection based on densities

Figure 8.9: Fast Intersection Detection on Interval Containers.

Let us note that we carry out this fast intersection test for all overlapping interval containers before testing the exact interval intersection. If one of these fast intersection tests yields true, the intersection routine returns true without testing the data stored in the interval containers.

8.5.2 Priority Based Intersection Tests

If none of these fast-intersection-tests yields true, it is beneficial to order the relevant test candidate pairs of two objects. As soon as an intersection between two intervals belonging to two objects can be detected, we know that these two objects intersect, and thus, all other intersection candidates with respect to these two objects do not need to be evaluated any more. Consequently, promising intersection candidate pairs should be preferred tested. First, we need to define the *minimal intersection length* L_{min} between two interval containers, i.e. the minimal summarized intersection lengths of the intervals contained in the interval containers.

Definition 8.3 (minimal intersection length) *Let C_I and $C_{I'}$ be two overlapping interval containers. Furthermore, let $L_{gap}(C_I) = L(C_I) \cdot (1 - D(C_I))$ denote the summarized gap length of interval container C_I. Then, the minimal intersection length L_{min} is defined as follows:*

$$L_{min}(C_I, C_{I'}) = intersection\_length(H(C_I), H(C_{I'})) - (L_{gap}(C_I) + L_{gap}(C_{I'})).$$

Note that this measure can also receive negative values. If the minimal intersection length has a positive value, the interval containers definitely intersect according to the third fast-intersection-test condition (cf. Figure 8.9(c)). Otherwise, for $L_{min}(C_I, C_{I'}) < 0$, the interval container pairs may be intersect. Generally, the higher the minimal intersection length between two interval containers the higher the chance that they intersect. Let $(C_I, C_{I'})$ and $(C_J, C_{J'})$ be two overlapping interval container pairs. If $L_{min}(C_I, C_{I'}) > L_{min}(C_J, C_{J'})$ holds, then the pair (C_I, C'_I) intersect with a higher probability than the pair (C_J, C'_J). The closer L_{min} converge 0, the closer the summarized gap length of both interval containers to the length of their overlapping, and thus, the smaller the chance that all gaps of both containers are completely within the overlapping area.

After the fast-intersection-test, we propose to rank the remaining candidate pairs $(C_I, C_{I'})$ of two objects in descending order of their minimal intersection length $L_{min}(C_I, C_{I'})$ before refining the interval containers and carrying out the expensive interval-intersection tests. Thus, interval container pairs having the highest minimal-intersection-length value are tested first which increases the probability for an early intersection detection.

8.6 Experiments

In this section, we evaluate our approximation technique with respect to the achieved performance of the RI-tree with a special emphasis on the various data compression techniques introduced in Section 8.2.2. We exploit different grouping algorithms GRP in combination with various data compression techniques DC. For the compression of the byte sequence $B(C_I)$ of an in-

terval container C_I, we used the data compressors (DC) explained in the following table:

compr. type	description
NOOPT:	The $B(C_I)$ is unpacked.
BZIP2:	$B(C_I)$ is packed according to the BZIP2 approach.
ZLIB:	$B(C_I)$ is packed according to the $ZLIB$ approach.
OPTRLE:	$B(C_I)$ is packed with a simple run-length encoding according to the approach in [KPPS03a].
QSDC:	$B(C_I)$ is packed according to the $QSDC$ approach.

Table 8.3: Data compressors.

Furthermore, we grouped voxel intervals into interval containers depending on two grouping algorithms GRP:

MaxGap. This grouping algorithm tries to minimize the number of interval containers while not allowing that a maximum gap $G(C_I)$ of any interval container C_I 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 Section 8.3.1, namely a small number of interval containers and a high accuracy, i.e. small gaps are included in each of these containers.

CoDec. We grouped the interval containers according to our cost-based grouping algorithm $CoDec$ (cf. Section 8.3.5), where we chose the query distribution parameter $k = 10^{-5}$ (cf. Section 8.3) and used a look-up table for each packer. The look-up table was created by experimentally determining the average cost for evaluating an interval container C_I dependent on the length of its byte sequence.

The grouping based on $MaxGap(DC)$ does not depend on DC, whereas $CoDec(DC)$ takes the actual data compressor DC into account for performing the grouping.

In order to support the first filter step of $GRP(DC)$, we have implemented the RI-tree [KPS00a][KPS01] on top of the Oracle9i Server using

PL/SQL for most of the computational main memory based programming. The evaluation of the byte sequence routines (exact intersection evaluation) 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.

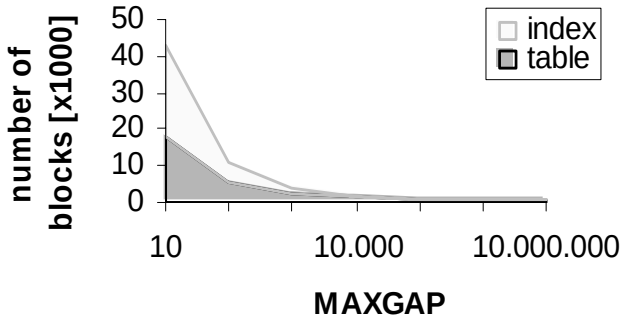
8.6.1 Test Datasets

The tests are based on two test data sets CAR and $PLANE$. These test datasets were provided by our industrial partners, a German car manufacturer and an American plane producer, in form of high resolution voxelized three-dimensional CAD parts. The CAR dataset consists of approximate 14 million voxels and 200 parts, whereas the $PLANE$ dataset consists of about 18 million voxels and 10,000 parts. The CAR data space is of size 2^{33} and the $PLANE$ data space is of size 2^{42}. In both cases, the *Z-curve* was used as a space filling curve to build the voxel intervals.

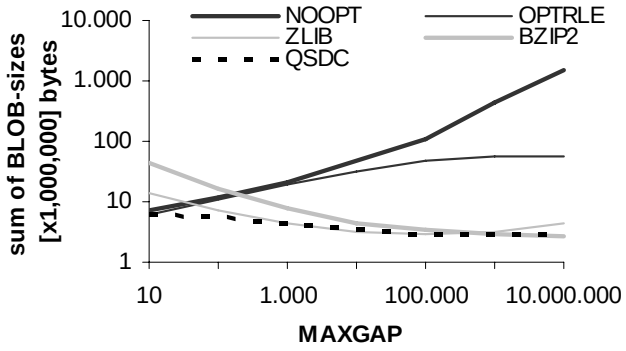
8.6.2 Storage Requirements

First we look at the storage requirements of the RI-tree on the $PLANE$ dataset. In Figure 8.10(a), the storage requirements for the index, i.e. the two B^+-trees underlying the RI-tree, as well as for the complete interval-container representations 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 interval containers. With increasing $MAXGAP$ parameters, the number of disk blocks used by the index dramatically decreases hand in hand with the number of interval container objects, and at high parameter values, i.e. large gaps and intervals, they yield no significant contributions to the overall sum of used disk blocks any more.

Figure 8.10(b) shows the different storage requirements for the com-



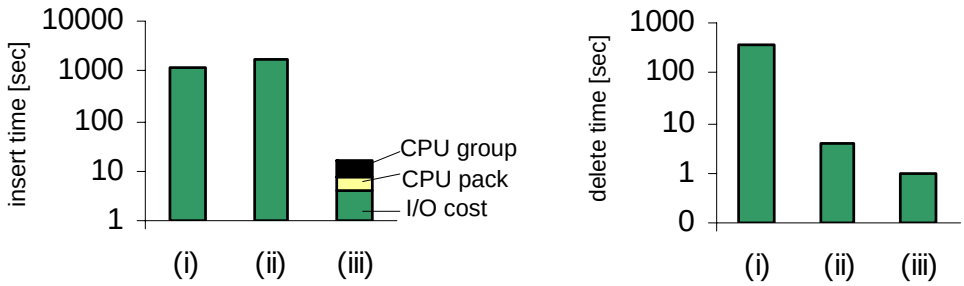
(a) Index & $B(C_I)$ for MaxGap (QSDC)



(b) $B(C_I)$ for MaxGap(DC)

Figure 8.10: Storage requirements for the RI-tree (PLANE).

pressed data 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 [KPPS03a]. 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 compressed data 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.



(a) insert-operation **(b)** delete-operation

Figure 8.11: Update operations for the RI-tree (CAR). (i) one interval container per interval; (ii) one interval container per object; (iii) interval containers grouped by CoDec(QSDC).

8.6.3 Update Operations

In this section, we will investigate the time needed for updating complex spatial objects in the database. For most of the investigated application ranges, it 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 8.11(a) shows that inserting all objects into the database takes very long if we store the numerous object 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 $CoDec(QSDC)$ 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 compressed objects in the database.

Obviously, the delete operations are also carried out much faster for our $CoDec(QSDC)$ approach, as we have to delete much less disk blocks (cf. Figure 8.11(b)).

8.6.4 Query Processing

In this section, we want to turn our attention to the query processing by examining different kinds of intersection queries. In addition to the pure intersection query, we will call it *boolean intersection query*, we also investigated *intersection volume queries* additionally reporting the intersection volume. While the query optimizing concepts of Section 8.5 can be successfully applied to the *boolean intersection queries*, we cannot use them for the *intersection volume queries*. Comparing the performance results of these two query types shows the effect of our optimization concept.

The figures presented in this paragraph depict the average result obtained from intersection queries where we have taken every part from the CAR data set and the 100 largest parts from the PLANE dataset as query objects.

Experiments with MaxGap:

In Figure 8.12 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 refinement 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 8.13 it is shown in which way the different data space resolutions influence the query response time. Generally, the higher the resolution the

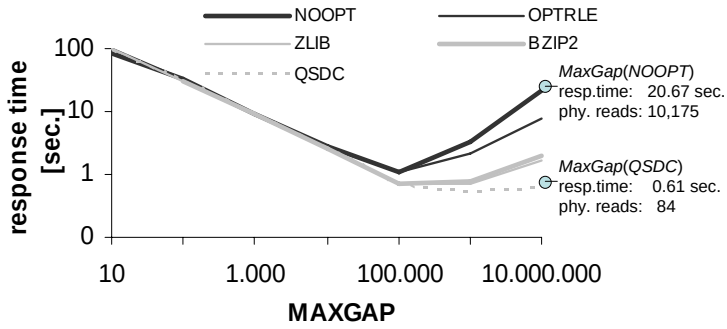


Figure 8.12: MaxGap(DC) evaluated for boolean intersection queries on the RI-tree (PLANE).

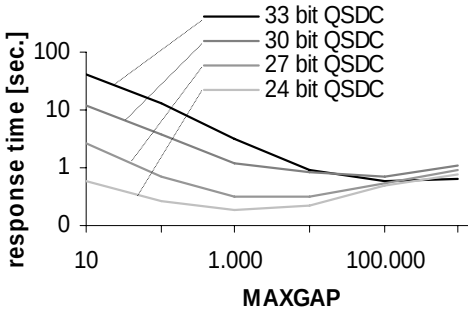


Figure 8.13: MaxGap(QSDC) evaluated for boolean intersection queries for the RI-tree using different resolutions (CAR).

slower 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 8.14 it is illustrated that at small $MAXGAP$ values the number of different object candidates resulting from the first filter step is only marginally higher than the number of 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 8.5 are especially useful for small $MAXGAP$ parameters.

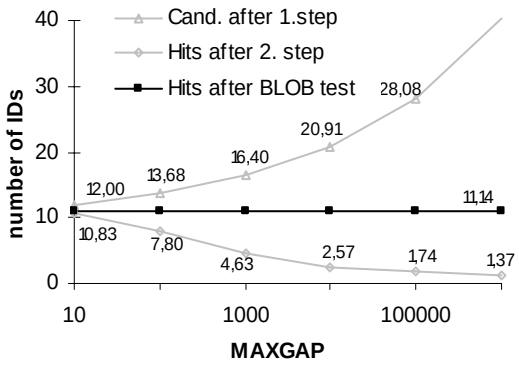


Figure 8.14: Object candidates and result sets for boolean intersection queries on the RI-tree (CAR).

Experiments with CoDec:

Figure 8.12 shows that for packed data the optimum $MAXGAP$ value is higher than for unpacked data. Furthermore, Figure 8.13 demonstrates that for increasing resolutions the optimum $MAXGAP$ also increases. We will now experimentally show that the $CoDec$ algorithm produces object decompositions which yield almost optimum query response times for varying index structures, compression techniques and data space resolutions.

Table 8.4 depicts the overall query response time for boolean intersection queries and intersection volume queries for the RI-tree based on the $CoDec$ algorithm.

	NOOPT	BZIP	QSDC	RI-tree without CoDec
number of containers	24,453	16,063	15,468	9,289,569
Overall Runtime* [s]	1.35	0.71	0.55	135.01
Overall Runtime** [s]	2.42	1.23	0.92	∞ (not applicable)

Table 8.4: $CoDec(DC)$ evaluated for Boolean intersection* and intersection volume** queries for the RI-tree ($PLANE$).

We can see that for boolean intersection queries this grouping delivers results quite close to the minimum response times depicted in Figure 8.12. Furthermore, we notice that the $CoDec(QSDC)$ approach outperforms the tra-

ditional RI-tree [KPS01] by a factor of 180 for boolean intersection queries on the $PLANE$ dataset. For intersection volume queries, the RI-tree [KPS00a] is not applicable due to the enormous amount of generated join partners. On the other hand, the $CoDec(QSDC)$ approach yields interactive response times even for such queries. The $CoDec$ algorithm adapts to the optimum $MAXGAP$ parameter for varying compression techniques by allowing larger gaps for packed data, i.e the number of generated container objects is smaller in the case of packed data.

In Table 8.5 it is shown that the query response times resulting from the $CoDec$ algorithm for varying resolutions are almost identical to the ones resulting from a grouping based on an optimum $MAXGAP$ parameter (cf. Figure 8.13).

	33 bit	30 bit	27 bit	24 bit
Overall Runtime [s]	0.64	0.7	0.29	0.22

Table 8.5: $CoDec(QSDC)$ evaluated for Boolean intersection queries for the RI-tree with different resolutions (CAR).

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

8.7 Summary

In this chapter, we introduced a new generic approach for accelerating spatial query processing for relational index structures. We presented interval containers as a new concept and showed how we can efficiently store them by means of data compression techniques. In particular, we presented a quick spatial data compressor $QSDC$ in order to emphasize those packer characteristics which are important for efficient spatial query processing, namely good compression ratio and high unpack speed. Furthermore, we proposed a cost-based decomposition algorithm for complex spatial objects, called $CoDec$.

CoDec takes the decompression cost of the interval containers and their access probabilities into account. This decomposition algorithm is applicable for different data space resolutions and compression algorithms. We showed in a broad experimental evaluation that our new approach, i.e. the combination of *CoDec* and *QSDC*, accelerates the Relational Interval-tree by up to two orders of magnitude.

Chapter 9

Join Queries for Complex Spatial Objects

Modern database applications, including computer-aided design (CAD), medical imaging, molecular biology or Multimedia Information Systems, impose new requirements on efficient spatial query processing. One of the most common query types in spatial databases is the spatial join. In this chapter, we concentrate us on the intersection join, as it is the most important join predicate for complex spatial objects [GG98]. It retrieves all object pairs from two given datasets that satisfy the spatial-intersection predicate, i.e. all pairs of intersecting objects are reported. A usual spatial join example of 2D geographical data is "find all cities which are crossed by a river". In the automobile industry, spatial join processing of complex 3D high-resolution objects is also required, e.g. to support efficient processing of queries like "find all engine parts which intersect the car body". Therefore, an efficient processing of spatial joins is indispensable.

9.1 Introduction

In many applications, *GIS* or *CAD* objects feature a very complex and fine-grained geometry. An important new requirement for large objects, including

cars or planes, is a high approximation quality. High resolutions yield a high approximation quality but result in high efforts in terms of identifying candidate pairs in a spatial join process. In this chapter, we follow the approach of Chapter 8 where spatial objects are conservatively approximated by a set of voxels, i.e. cells of a grid covering the complete data space. By means of space filling curves, an extended object is represented by a set of intervals which we call *voxel intervals*. In the last chapter, we have seen that in particular for high-resolution data, where the number of resulting intervals still remains very high, further grouping of the intervals can significantly speed up intersection queries. For the join processing, we adopt the approach where voxel intervals are grouped into *interval containers* (cf. Section 8.2) which are approximations of interval sequences, comprising some aggregate information of the covered intervals. This chapter comprises the approaches published in [KKPR04b] and [KKPR05c].

In contrast to the last chapter, where we focused on building approximations based on an assumed query-distribution function, and a cost model for single intersection queries supported by the RI-tree, in this chapter, we turn our attention to spatial join queries. In Section 9.5, we first present a simple join algorithm, the *nested-loop join*. It is based on a combination of two decomposition methods: the cost-based decomposition as presented in Chapter 8 and the statistic driven query-decomposition approach presented in Section 6.4.

In Section 9.6, we will introduce a *sort-merge-based join* procedure which also uses statistics in order to build adequate object approximations for an efficient join processing. In opposition to the nested-loop join algorithm it does not need a full table scan on the one relation for each query object of the other relation. It tries to join only those objects sharing the same location in space.

9.2 Related Work

At first, we will shortly discuss related work on efficient spatial join processing of complex spatial objects.

Spatial Join. Numerous spatial join algorithms have been proposed over the last decade. Most of them rely on the paradigm of multi-step query processing [BKSS94]. A fast filter step excludes all objects that cannot satisfy the join predicate. The subsequent refinement step is applied to the join candidate pairs which are returned from the filter step. Thereby, the main focus of research is on the filter step which is applied to geometric object approximations. On the basis of the availability of indices for processing the filter step, spatial join methods operating on two relations can be classified into three classes:

- Class 1: Index on both relations
- Class 2: Index on one relation
- Class 3: No indices

Common solutions for the spatial join methods of *Class 1* are the algorithms based on matching the two spatial access methods in order to identify the sets of potential join candidates during the traversal of the index structures. The most prominent approach is based on a hierarchical matching of two R-trees as presented in [BKS93b] and [HJR97]. This method synchronously traverses both R-trees from top to bottom and builds for each step the sets of potential join candidates based on the rectangle entries of the corresponding directory pages. Given to directory pages only if the rectangles of both directory pages match the join predicate. Both sets of data objects (in)directly referenced by the directory pages have to be considered for the remaining join process. Recently, a join algorithm for interval data in relational databases has been proposed [EHS04]. This approach is based on matching two RI-trees by identifying pairs of RI-tree nodes representing sets of intervals which are potential join candidates.

In the last few years, the international research community has focused on methods of *Class 2* and *Class 3*. A simple *Class 2* approach is the index nested-loop, where each tuple of the non-indexed relation is used as query applied to the indexed relation. More efficient solutions are presented in [PD96][PRS99][LR96]. These algorithms are based on building an index for the non-indexed relation in order to reduce the problem to the join where both relations are indexed (*Class 1*). For spatial join algorithms of *Class 3*, initially no indices are available which could be used to improve the query performance. Several partitioning techniques have been proposed which partition the tuples into buckets and then use either hash-based or sweep-line techniques, e.g. the spatial-hash join [LR96], the Partition-Based Spatial-Merge join (PBSM) [PD96] or the Scalable Sweeping-Based Spatial Join (SSSJ) [APR+98]. The latter approaches work well for relative simply shaped 2D objects which can be well approximated by their minimal bounding boxes. In contrast to these approaches, our approach deals with very complex 3D objects, where the minimal bounding box is a rather poor approximation. In order to use the PBSM or the SSSJ with decomposed objects, some modifications has to be done, as for instance duplicate elimination.

9.3 Cost Model

For our decomposition algorithm we take the estimated join cost between an interval container C_I and a join-partner relation T into account. Let us note that T can be either one of the two join relations R or S or any temporary table containing derived information from the original relations R and S. The overall join cost $cost_{join}$ for an interval container C_I and a join-partner relation T are composed of two parts, the *filter cost* $cost_{filter}$ and the *refinement cost* $cost_{refine}$:

$$cost_{join}(C_I, T) = cost_{filter}(C_I, T) + cost_{refine}(C_I, T).$$

Filter Cost. The filter cost $cost_{filter}(C_I, T)$ can be computed by the expected number of interval containers $C_{I,T}$ of the join partner relation T. We

penalize each intersection test by a constant c_f which reflects the cost related to the access of one interval container $C_{I,T}$ and the evaluation of the join predicate for the pair $(H(C_I), H(C_{I,T}))$:

$$cost_{filter}(C_I, T) = N_{cont}(T) \cdot c_f,$$

where $N_{int}(T)$ (number of voxel intervals) $\geq N_{cont}(T)$ (number of interval containers) $\geq N_{obj}(T)$ (number of objects) holds for the join-partner relation T. The value of the parameter c_f depends on the used system.

Refinement Cost. The cost of the refinement step $cost_{refine}$ is determined by the selectivity of the filter step. For each candidate pair resulting from the filter step, we have to retrieve the exact geometry $B(C_I)$ in order to verify the intersection predicate. Consequently, our cost-based decomposition algorithm is based on the following two parameters:

- Selectivity σ_{filter} of the filter step.
- Evaluation cost $cost_{eval}$ of the exact geometries.

The refinement cost of a join related to an interval container C_I can be computed as follows:

$$cost_{refine}(C_I, T) = N_{cont}(T) \cdot \sigma_{filter}(C_I, T) \cdot cost_{eval}(C_I).$$

In Section 8.3 it was shown how to calculate the evaluation cost $cost_{eval}$ of interval containers approximating the spatial objects. In the following, we show how to estimate the selectivity of the filter step σ_{filter}. We use simple statistics of the join-partner relation T to estimate the selectivity $\sigma_{filter}(C_I, T)$. In order to cope with arbitrary interval distributions, histograms can be employed to capture the data characteristics at any desired resolution (cf. Section 4.5.1). The selectivity $\sigma_{filter}(C_I, T)$ related to an interval container C_I can be determined by using an appropriate interval histogram $IH(T, \nu)$ of the join partner relation T. Based on $IH(T, \nu)$, we compute a selectivity estimated by evaluating the intersection of C_I with each bucket span $b_{i,\nu}$.

$$\sigma_{filter}(C_I, T) = \frac{\sum_{i=1}^{\nu} \frac{overlap(H(C_I), b_{i,\nu})}{\beta} \cdot n_i}{\sum_{i=1}^{\nu} n_i},$$

where *overlap* returns the intersection length of two intersecting intervals, and 0 if the intervals are disjoint.

Note that long intervals may span multiple histogram buckets. Thus, in the above computation, we normalize the expected output to the sum of the number n_i of intervals intersecting each bucket i rather than to the original cardinality n of the relation T.

Join Cost. To sum up, the join cost $cost_{join}(C_I)$ related to an interval container C_I and a join-partner relation T can be expressed as follows:

$$cost_{join}(C_I, T) = N_{cont}(T) \cdot (c_f + \sigma_{filter}(C_I, T) \cdot cost_{eval}(C_I)).$$

9.4 Decomposition Algorithm

Based on the formulas for join cost related to an interval container C_I and a join-partner relation T, we propose a cost optimum decomposition algorithm $CoDecJ$ in order to build the interval containers. In this section, we will show how the decomposition algorithm $CoDec$ presented in Section 8.2 can be easily adapted to produce decompositions helping to accelerate join queries considerably. Note that like the algorithm $CoDec$, $CoDecJ$ is a greedy algorithm with a guaranteed worst-case runtime complexity of $O(n)$.

The top-down decomposition algorithm called $CoDecJ$ is depicted in Figure 9.1. $CoDecJ$ is a recursive algorithm which starts with an interval container C_I, initially covering the complete object. In each step of our algorithm, we look for the longest remaining gap. We carry out the split at this gap if the estimated join cost caused by the decomposed intervals is smaller than the estimated cost caused by our input interval C_I. The expected join cost $cost_{join}(C_I, T)$ can be computed as described above. Data compressors which have a high compression rate and a fast decompression method result in an early stop of the $CoDecJ$ algorithm, generating a small number of interval containers. Let us note that the inequality "$cost_{comp} > cost_{dec}$" in Figure 9.1 is independent of $N_{cont}(T)$, and thus, $N_{cont}(T)$ is not required any more during the decomposition algorithm.

```
CoDecJ(C_I, IH(T,ν), T){
    cost_comp := cost_join(C_I, T, IH(T,ν));
    container_pair := split_at_maximum_gap(C_I);
    C_left := container_pair.left;
    C_right := container_pair.right;
    cost_dec := cost_join(C_left, T, IH(T,ν)) + cost_join(C_right, T, IH(T,ν));
    if cost_comp > cost_dec then
      CoDecJ(C_left,IH(T,ν),T);
      CoDecJ(C_right,IH(T,ν),T);
    else
      report(C_I);
    end if;
}
```

Figure 9.1: Decomposition Algorithm $CoDecJ$.

In the next section, we turn our attention to the processing of two different join algorithms. Exemplarily, we explain in detail how our decomposition algorithm can help to accelerate a variant of the *nested-loop join* and the *sort-merge join*. Let us note that the main focus of this chapter is not the presentation of new join algorithms, but the presentation of a theoretically sound and practically relevant object decomposition concept which helps to accelerate known join algorithms on complex spatial objects linearized with high precision.

9.5 Nested-Loop Based Join Processing

In the following, we assume that we have to join relation R with relation S containing complex spatial objects in form of voxel interval sequences. Our nested-loop join algorithm will be processed in two phases, the *preprocessing phase* and the *join phase*, as depicted in Figure 9.2.

Preprocessing Phase. For each object obj_S in relation S we apply the decomposition function $CoDecJ$ (cf. Figure 9.1) which builds the interval

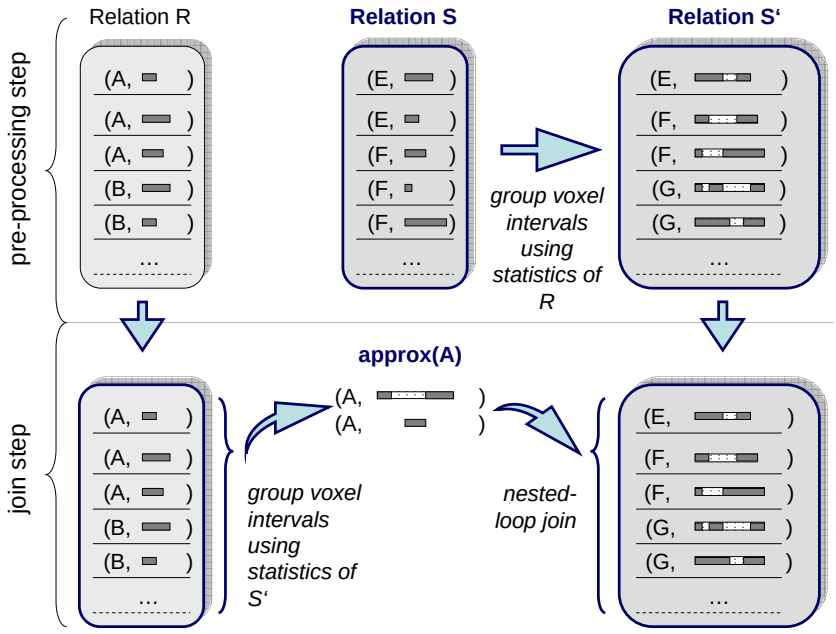


Figure 9.2: Nested-Loop Based Join Processing.

containers. This grouping algorithm takes the statistics of the data distribution with respect to relation R, i.e. the interval histogram $IH(R, \nu)$, into account. Then the resulting interval containers of each object are materialized in an auxiliary relation S'.

Note that we assume that the objects obj_R of relation R will be accessed only once. Thus, there is no need to materialize the interval containers of obj_R in the database as done for the objects in relation S. Furthermore, we assume that one object completely fits in memory so that its interval containers can be built on-the-fly during the join phase.

Join Phase. The join phase is performed in a $nested-loop$ fashion. In an outer loop we decompose each object obj_R from R by means of $CoDecJ$ into interval containers. This time, we apply the data distribution statistics of relation S', i.e. the interval histogram $IH(S', \nu)$. In the inner loop, we test each object obj_S for intersection with object obj_R, calling the boolean function $intersect()$.

The nested-loop join algorithm is depicted in Figure 9.3. The func-

```
R relation (id,voxel-interval);   // objects of relation R
S relation (id,voxel-interval);   // objects of relation S
S' temporal relation (id,interval-container);

NL-join(R, S){
   // preprocessing phase:
   for each object obj_S in S do
      obj_S' := CoDecJ(obj_S.C_I, IH(R,ν), R);
      store(obj_S') in S';
   end for;
   // join phase:
   result_set := ∅;
   for each object obj_R in R do {
      obj_R' := CoDecJ(obj_R.C_I, IH(S',ν), S');
      for each object obj_S' in S' do {
         if intersect(obj_R', obj_S') then
            result_set := result_set∪(obj_R'.id, obj_S'.id);
         end if;        end for;
   end for;
}
```

Figure 9.3: Nested-Loop Join Algorithm *NL-join*.

tion intersect($obj_{R'}$, $obj_{S'}$) checks whether two objects $obj_{R'}$ and $obj_{S'}$ intersect. They intersect iff there is at least one interval-container pair ($obj_{R'}.C_I$, $obj_{S'}.C'_I$) which intersects. As soon as an intersection between two objects is detected, the remaining tests according to these two objects can be skipped. The intersection test of an interval-container pair is performed in two steps: In a filter step the pair is tested with respect to their hulls. If the result of the filter step is positive, i.e. the hulls intersect, a subsequent refinement step verifies the intersection with respect to the exact geometric object representations. Note that before loading the exact voxel-interval information for the refinement, we usually can apply the fast-intersection-test methods presented in Section 8.5 in order to detect true hits.

Before evaluating two interval containers for intersection, we have to load the compressed byte code of the exact geometric representations from disk and decompress it. Certainly, it is important that the size of the compressed interval-containers 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 only the objects of the inner relation S' are in a compressed form, since the objects of R does not influence the I/O cost very much. Furthermore, a fast decompression algorithm is required to evaluate the byte sequence $B(C_I)$ of the interval container I_C quickly.

9.6 Sort-Merge Based Join Processing

The join algorithm introduced in this section is based on the worst-case optimal interval join algorithm described in [APR+ 1998] and on the cost-based decomposition approach described in Section 9.4. We again consider R and S as input relations. The starting point of this algorithm is like that of the nested-loop join algorithm (cf. Section 9.5), i.e. each object is preliminarily approximated by one interval container covering the complete voxel-interval sequence of the object. In contrast to the nested-loop join algorithm, our sort-merge join algorithm is performed in plane-sweep fashion according to the preliminary interval containers. We process the interval containers in ascending order of their starting points, i.e. $H(C_I).lower$. Note, for the prior sorting of the interval containers we assume that we can access the hulls $H(C_I)$ without accessing the detailed object description, i.e. the entire content of the interval containers. As we cannot assume that the sweep-line status completely fits in memory, we additionally use two auxiliary relations R' and S' to hold the actual sweep-line status on disk.

In analogy to the nested-loop join, we apply the algorithm CoDecJ (cf. Figure 9.3) in order to adjust the object approximations to the data distribution of the respective join-partner relation. Again, for the computation of the data distribution, we use interval histograms with the exception that we perform the decomposition in two steps in which we employ two different

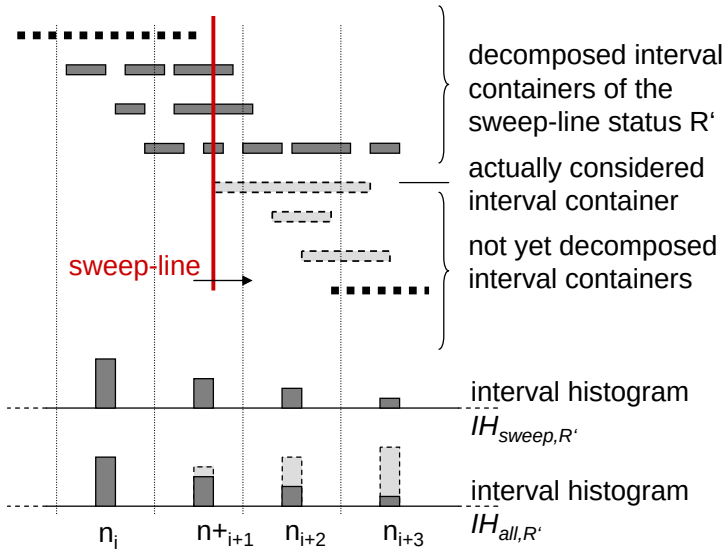


Figure 9.4: Interval containers from relation R and the corresponding interval histograms $IH_{sweep,R'}$ and $IH_{all,R'}$.

interval histograms for each dataset. The interval histograms $IH_{sweep,R'}$ and $IH_{sweep,S'}$ represent the data distribution within the actual sweep-line status. The other interval histograms $IH_{all,R'}$ and $IH_{all,S'}$ represent the overall data distribution derived from R and S in conjunction with the actual sweep-line status. In the following, we assume that all interval histograms have the same resolution ν, so that their bucket borders are congruent. An example is shown in Figure 9.4.

During the join processing, we try to estimate the filter selectivity for each actual considered interval container as precisely as possible. For those objects which have already been processed, we take the exact interval distribution into account which is retrieved from the actual sweep-line status, i.e. from R' respective S'. For the contribution of the remaining objects, where no decomposition was performed yet, we take the distribution of the corresponding undecomposed objects into account which can be derived from R and S.

Our sort-merge join algorithm consists of two phases where the second phase in turn consists of three steps which are performed for each object.

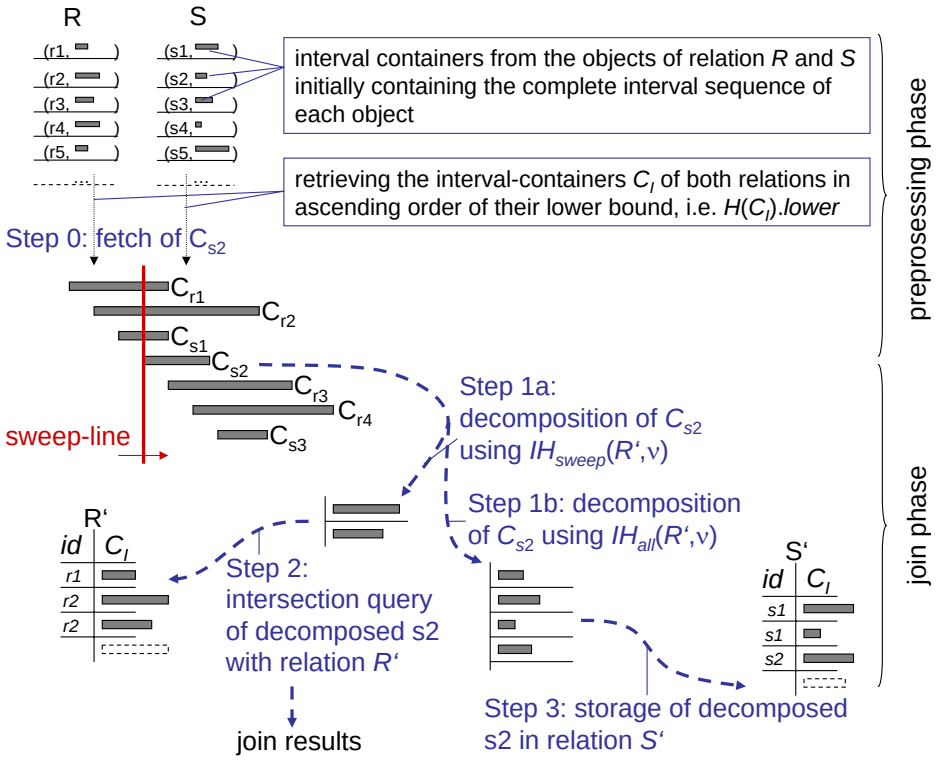


Figure 9.5: Two-Phase Sort Merge Join.

The complete join algorithm described in the following is depicted in Figure 9.5.

1. Phase (Preprocessing Phase). Initially, we gather the statistics about the data distribution of R and S and store them in the two interval histograms $IH_{all,R'}$ and $IH_{all,S'}$, respectively. Thereby, $IH_{all,R'}$ is generated from the data of R and $IH_{all,S'}$ from the data of S. Next, we order all preliminary interval containers of both relations R and S according to their starting point, i.e. $H(C_I.lower)$. Let us note that sorting the objects is rather cheap if we assume that we have comparatively few objects but rather complex ones.

2. Phase (Join Phase). We apply a plane sweep algorithm to walk through the sorted lists of interval containers of both relations R and S. The event points of this algorithm are the starting points of the interval

containers of both relations. Each encountered interval container C_I from relation S (R) is now processed according to the following three steps:

- Step 1: C_I is decomposed, based on the data distribution of the actual sweep-line status by applying $IH_{sweep,R'}$ ($IH_{sweep,S'}$) and stored in a temporary list Q. In the same step, we decompose C_I applying the statistics $IH_{all,R'}$ ($IH_{all,S'}$) and buffer the result in another temporary list D.

- Step 2: The resulting decomposed interval containers of Q are used as query objects for the relation R' (S'). We report all objects having an interval container C'_I stored in R' (S') which intersects at least one of the decompositions of C_I. These intersection queries can be efficiently carried out as outlined in Section 8.5.

- Step 3: The decomposed interval containers of D are stored in the temporary relation S' (R'). In order to keep the relations R' and S' small, we delete all interval containers C'_I from R' and S' where $H(C'_I.upper)$ is smaller than $H(C_I.lower)$, i.e. all interval containers which are certainly not accessed anymore. Finally, we have to update the interval histograms $IH_{all,S'}$ and $IH_{sweep,S'}$ ($IH_{all,R'}$ and $IH_{sweep,R'}$).

Let us note that our algorithm *CoDecJ* considers in each step i the i-th longest gap g_i independent of the chosen histogram. We suggest to compute the object decompositions for the respective interval histograms in parallel. This approach guarantees that we consider each gap only once.

Similar to the nested-loop join algorithm, the presented sort-merge join does not require any duplicate elimination. Furthermore, the main memory footprint of our sort-merge join algorithm is negligible because we do not keep the sweep-line status in main memory. Even if we kept it in main memory, the use of suitable data compressors would lead to a small main memory footprint.

9.7 Experimental Evaluation

In this section, we evaluate the performance of our approach with a special emphasis on different decomposition algorithms in combination with various data compression techniques DC. We used the following data compressors: no compression ($NOOPT$), the $BZIP2$ approach [Sew06] and the $QSDC$ approach. Furthermore, we decomposed object voxels into interval containers according to two decomposition algorithms $MaxGap$ and $CoDecJ$.

MaxGap (cf. Section 8.6). This decomposition algorithm tries to minimize the number of interval containers while not allowing that a maximum gap $G(C_I)$ of any interval container C_I exceeds a given $MAXGAP$ parameter. By varying this $MAXGAP$ parameter, we can find the optimum trade-off between the first two opposing decomposition rules, namely a small number of interval containers and a small approximation error of each of these intervals. A one-value interval approximation is achieved by setting the $MAXGAP$ parameter to infinite.

CoDecJ. We decomposed the linearized objects according to our cost-based decomposition algorithm $CoDecJ$, presented in Figure 9.1, where we set the resolution of the used interval histograms to 100 buckets.

Let us note that the decomposition based on $MaxGap(DC)$ does not depend on DC or any statistical information about the data distribution, whereas $CoDecJ(DC)$ takes the actual data compressor DC and the actual data distribution into account for performing the decomposition.

The refinement-step evaluation of the *intersect()* routine was delegated to a DLL written in C. All experiments were performed on a Pentium 4/2600 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 two test datasets CAR (3D CAD data) and $SEQUOIA$ (subset of 2D GIS data representing woodlands derived from the SEQUOIA 2000 benchmark [SFGM93]). The characteristics of both

datasets are summarized in Table 9.1. The first test dataset was provided by our industrial partner, a German car manufacturer, in form of high resolution rasterized three-dimensional CAD parts.

dataset	#voxels	#objects	size of data space
CAR (3D)	$14 \cdot 10^6$	200	2^{33} cells
$SEQUOIA$ (2D)	$32 \cdot 10^6$	1100	2^{34} cells

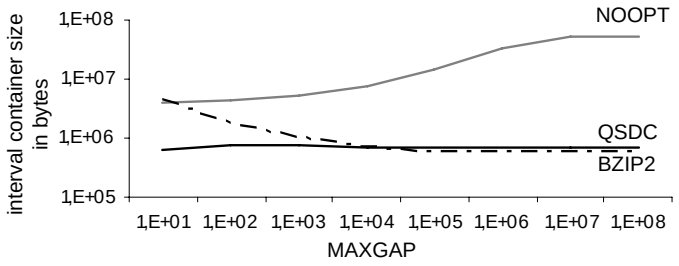
Table 9.1: Test data sets.

Both datasets are linearized by means of a space filling curve in Z-order. They consist of many short intervals and short gaps and only a few longer ones.

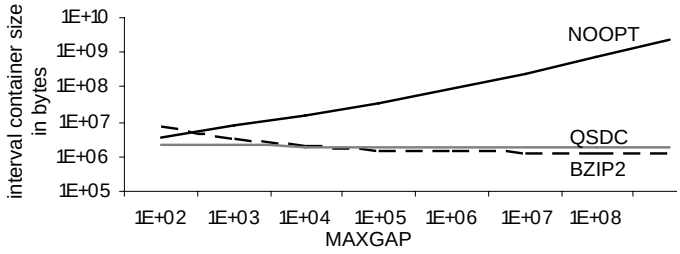
9.7.1 Compression Techniques

Figure 9.6 shows the different storage requirements of the materialized interval containers with respect to the different data compression techniques. For high $MAXGAP$ values, the $BZIP2$ approach yields very high compression rates, i.e. compression rates up to 1:100 for the $SEQUOIA$ dataset and 1:500 for the CAR dataset. Note that the higher compression rates for the CAR dataset are due to fact that it is a 3D dataset, whereas the $SEQUOIA$ dataset is a 2D dataset. This additional dimension leads to an enormous increase of the size of the materialized interval containers, making suitable compression techniques indispensable. On the other hand, due to a noticeable overhead, the $BZIP2$ approach occupies even more secondary storage space than $NOOPT$ for small $MAXGAP$ values. Contrary, the $QSDC$ approach yields good results over the full range of the $MAXGAP$ parameter. Using the $QSDC$ compression technique, we achieve low I/O cost for storing (Step 3 in two-phase sort-merge join) and fetching (Step 2 in two-phase sort-merge join) the interval containers which drastically enhances the efficiency of the join process.

In the following, we want to investigate the runtime behavior of our two decomposition-based join algorithms, the nested-loop join and the two-phase



(a) SEQUOIA



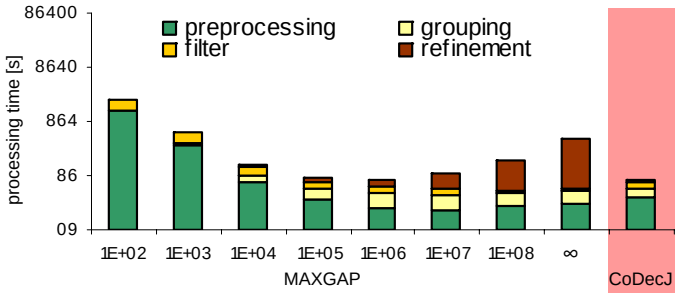
(b) CAR

Figure 9.6: Storage requirements for the interval containers.

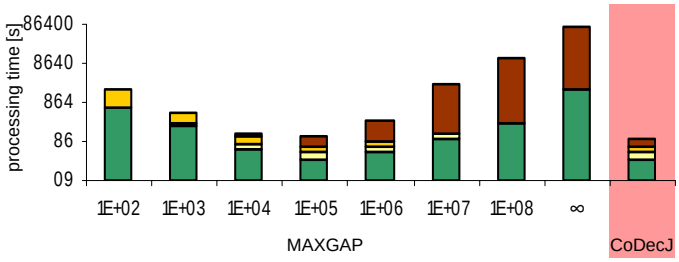
sort-merge join. We performed the intersection join queries over two relations, each containing approximately a half of the parts from the CAR dataset. We took care that the data of both relations have similar characterizations with respect to the object size and distribution. Similarly, the intersection join is performed on parts of the $SEQUOIA$ dataset which is divided into two relations, consisting of deciduous-forest and mixed-forest areas.

9.7.2 Performance Evaluation for the Nested-Loop Join

In Figure 9.7 it is shown in which way the response time for the intersection join query, including the preprocessing step, depends on the $MAXGAP$ parameter using the $QSDC$ compression (cf. Figure 9.7(a)) and no compression (cf. Figure 9.7(b)). The figures depict the overall contributions of the preprocessing phase of the on-the-fly decomposition (cf. Figure 9.2) and of the filter and refinement step. If we use small $MAXGAP$ parameters, we need a lot of time for the filter step, whereas the refinement step which is



(a) *QSDC* compression

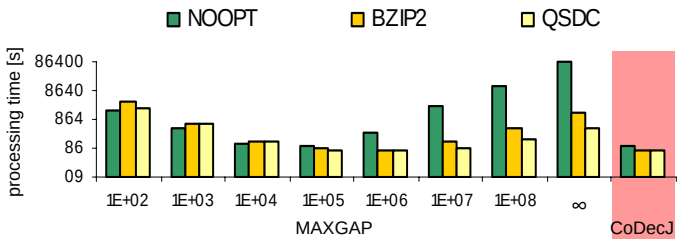


(b) *NOOPT* (no compression)

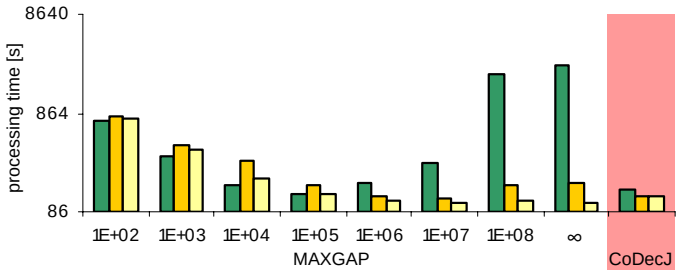
Figure 9.7: *MaxGap* and *CoDecJ* evaluated for intersection joins on the *CAR* dataset.

influenced by the size of the byte sequence of the interval container is relatively cheap. On the other hand, for high $MAXGAP$ values we can see that the refinement step is very expensive in contrast to the filter step which shows very little contribution. Due to the fact that the performance mainly depends on the I/O cost, the preprocessing step shows a similar performance behavior as the pure join. We can observe that for both compression cases the *CoDecJ* approach exceeds the best $MAXGAP$ approach with respect to both compression variants. The marginally higher preprocessing cost of the *CoDecJ* algorithm result from the computation of the cost-estimations required for the decomposition.

Figure 9.8 illustrates how the overall join runtime depends on the different grouping techniques for both datasets, *CAR* (cf. Figure 9.8(a)) and *SEQUOIA* (cf. Figure 9.8(b)). For packed data, the optimum $MAXGAP$ value is higher than for unpacked data, i.e. $MAXGAP = 10^5$ for *NOOPT* and $MAXGAP = 10^6$ for *BZIP2* and *QSDC*. The *CoDecJ* algorithm pro-



(a) *CAR* dataset



(b) *SEQUOIA* dataset

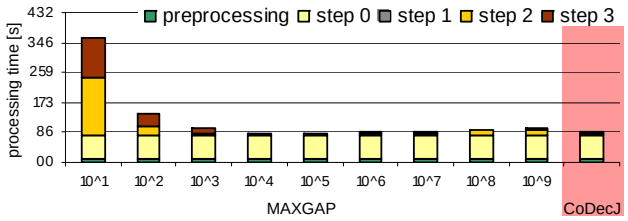
Figure 9.8: Overall join performance for different packers.

duces for both datasets object decompositions which yield almost optimum join response time for varying compression techniques. It adapts to the optimum $MAXGAP$ parameter for varying compression techniques by allowing greater gaps for packed data, i.e the number of generated interval containers is smaller in the case of packed data.

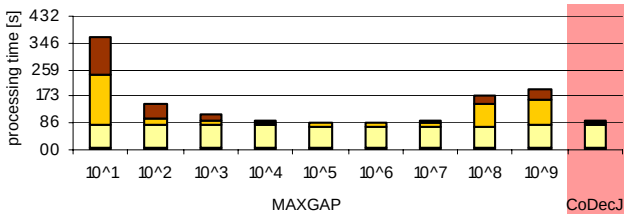
To sum up, the *CoDecJ* algorithm produces object decompositions which yield the optimal trade-off between the filter and refinement cost for both high-resolution datasets.

9.7.3 Performance Evaluation for the Two-Phase Sort-Merge Join

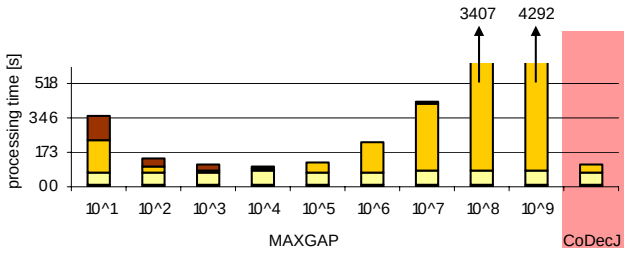
In Figure 9.9 and Figure 9.10 it is shown how the response time for the intersection join, including the preprocessing step, depends on the $MAXGAP$ parameter if we use no cache, i.e. the temporary relations R' and S' are not kept in main memory. The preprocessing time, i.e. the time for the creation



(a) *QSDC*



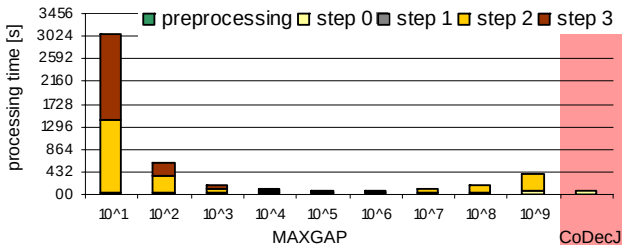
(b) *BZIP2*



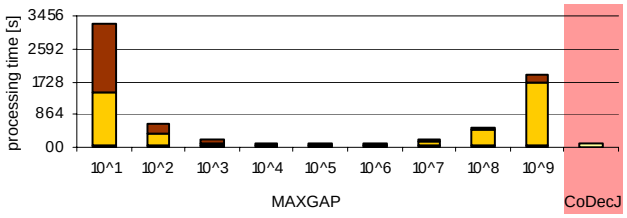
(c) *NOOPT*

Figure 9.9: *MaxGap* and *CoDecJ* grouping on *SEQUOIA* for different compression algorithms (main memory cache disabled).

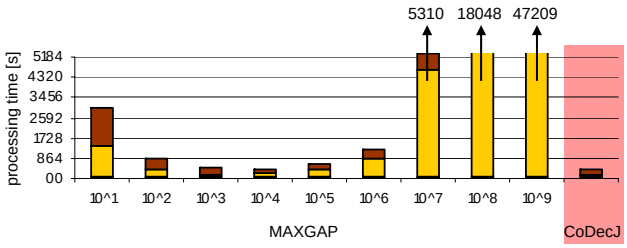
of the statistics, is negligible. Step 0 of the join phase, i.e. the loading of the exact object descriptions, is rather high and almost constant w.r.t. a varying $MAXGAP$ parameter. On the other hand, Step 1, the statistic-based decomposition of our interval containers is very cheap for our *CoDecJ* algorithm, and for the *Maxgap* approach it is not needed. Step 2, i.e. the actual intersection query, heavily depends on the used $MAXGAP$ value and the applied compression algorithm. For small $MAXGAP$ values, we have rather high cost for all compression techniques as the number of used query interval containers is very high. For high $MAXGAP$ values, we only have high cost if we use the *NOOPT* compression approach. On the other hand, if we use our *QSDC* approach, the actual cost for the intersection queries stay low,



(a) *QSDC*



(b) *BZIP2*



(c) *NOOPT*

Figure 9.10: *MaxGap* and *CoDecJ* grouping on *CAR* for different compression algorithms (main memory cache disabled).

as we have rather low I/O cost and can efficiently decompress the interval containers. If we use the *BZIP2* approach for high *MAXGAP* values, we also have low I/O cost but higher CPU cost than for the *QSDC* approach. Due to these rather high CPU cost, the *BZIP2* approach performs worse than the *QSDC* approach. The incidental cost for Step 3, i.e. the storing of the decomposed interval containers in temporary relations, can be explained similar to the cost for Step 2. Note that the cost for Step 2 and Step 3 are smaller if we allow a higher main memory footprint.

For *MAXGAP* values around 10^6, our join algorithm works most efficiently for the *QSDC* and *BZIP2* compression approaches. Our *CoDecJ*-

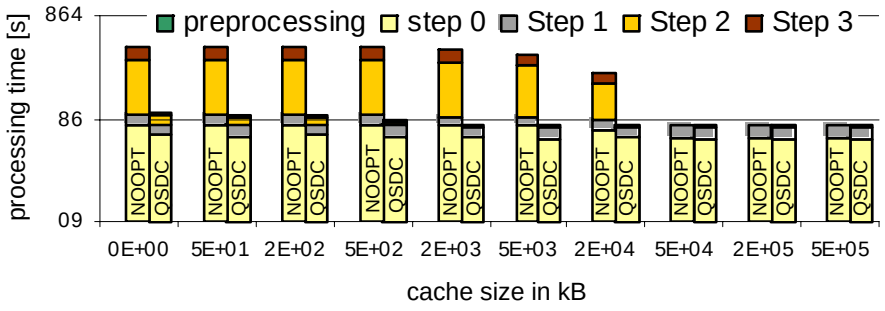


Figure 9.11: Sort-merge join performance with $CoDecJ$ algorithm for different cache sizes of the sweep-line status (CAR dataset).

based decomposition yields results quite close to these optimum ones. For the $NOOPT$ approach, the best possible runtime can be achieved for $MAXGAP$ values around 10^4. Again, the runtime of our join based on the $CoDecJ$ algorithm is close to this optimum one, justifying the suitability of our grouping algorithm.

Figure 9.11 demonstrates for the CAR dataset how the runtime of the complete join algorithm depends on the available main memory. We keep as much as possible of the sweep-line status in main memory instead of immediately externalizing it. The figure shows that for uncompressed data Step 2 and Step 3 (cf. Figure 9.5) are very expensive if the available main memory is limited. If we use our $CoDecJ$ algorithm without any compression, we need about 50 MB or more to get the best possible runtime. If we use $CoDecJ$ in combination with the $QSDC$ approach, we only need about 2 MB to get the best runtime. The two optimum runtimes are almost identical because one of the main design goals of the $QSDC$ was high unpack speed. Already with a main memory footprint of 0 KB, i.e. the sweep-line status cache is disabled, the $QSDC$ approach achieves runtimes close to the optimum ones, demonstrating a high compression ratio of the $QSDC$.

Figure 9.12 for the CAR dataset and Figure 9.13 for the $SEQUOIA$ dataset show the influence of the available main memory for one-value interval approximations, i.e. $O_{cont} = (id, C_I)$, and container approximations formed by our $CoDecJ$ algorithm. The one-value interval approximations

produce more false hits resulting in higher refinement cost. One-value interval approximations of uncompressed data cannot be kept in main memory even if allowing a main memory footprint of up to 1.5 GB. Furthermore, the figures demonstrate the superiority of the $QSDC$ approach compared to the $BZIP2$ approach, independent of the available main memory. This superiority is due to the high (un)pack speed of the $QSDC$ and a comparable compression ratio.

To sum up, our cost-based decomposition algorithm $CoDecJ$ together with our $QSDC$ approach can significantly speed up the nested-loop join and the sort-merge join while keeping the required main memory small. For reasonable main memory sizes, we achieve an acceleration by more than one order of magnitude for the $SEQUOIA$ dataset and by more than two orders of magnitude for the CAR dataset compared to the traditionally used non-compressed one-value approximations. Let us note that the sort-merge join outperforms the nested-loop join only with the $SEQUOIA$ dataset. The reason is the different characteristics of the datasets, in particular different dimensionality and density of the object locations. In contrast to the $SEQUOIA$ objects, the objects of the CAR dataset are all very close to each other which leads to a significant overlap of the one-value approximations of all objects. As a result, almost all objects intersect the sweep-line at the same time, and thus, they are nearly simultaneously in the sweep-line status.

9.8 Summary

In this chapter, we introduced a new approach for efficient processing of spatial intersection joins over high-resolution objects. In our approach, it is assumed that there is no spatial index available. In particular, we presented solutions for two joins, one for the nested-loop join and one for the sort-merge join. Both join procedures are based on fast filter steps performed on object approximations and an expensive refinement step. We used the concept of interval containers for the object approximation. The core of our approach

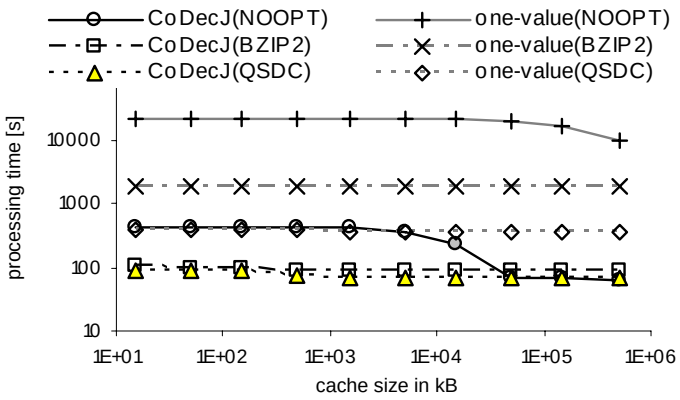


Figure 9.12: Overall sort-merge join performance for different cache sizes of the sweep-line status (CAR dataset).

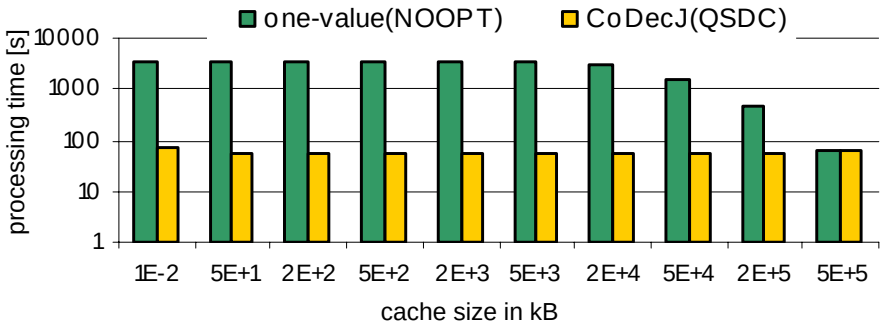


Figure 9.13: Sort-merge join performance for different cache sizes of the sweep-line status ($SEQUOIA$ dataset).

is a cost-based decomposition algorithm, building the object approximations in a for the corresponding join process convenient way. The decomposition algorithm takes cost of the filter and refinement step of the join procedure into consideration. The cost model takes the actual data distribution reflected by statistical information into account. In a broad experimental evaluation on real-world datasets, we demonstrated the efficiency of our both spatial join algorithms for complex spatial objects.

Chapter 10

Distributed Spatial Join Processing

In many different application areas, e.g. space observation systems or engineering systems of world-wide operating companies, there is a need for an efficient distributed intersection join in order to extract new and global knowledge. A solution for carrying out a global intersection join is to transmit all distributed information from the clients to a central server, leading to high transfer cost. In this section, we present a new distributed intersection join for interval sequences of high-cardinality which tries to minimize these transmission cost.

Our approach is based on a suitable probability model for interval intersections which is used on the server as well as on the various clients. On the client sites, we group intervals together into *interval containers* (cf. Section 8.2.1) based on this probability model. These 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 us to cut down the transmission cost considerably which is proven by our experimental evaluation based on synthetic and real-world test datasets.

10.1 Introduction

In this section, we consider interval data or, more generally, interval-sequence data residing on different, independently working computers which are connected to each other via local or wide area networks (LANs or WANs). Applications of this type of data comprise distributed mobile networks, sensor networks or vehicle manufacturers, where the development agencies are located at different places, distributed all over the world. For instance, international companies such as Daimler-Chrysler AG have some development agencies which are located in Europe, Asia, and the USA. In the areas of digital mock-up of vehicles and airplanes [KPPS03a], haptic simulations in virtual product environments [MPT99] or engineering data management as well as in the areas of two-dimensional GIS and environmental information systems [MP94], the locally collected data can only, with great difficulty, be transmitted to a central site and centrally joined there. Meeting the need of all these application ranges, in this chapter, we will present a distributed interval intersection join which extracts global knowledge while taking limited bandwidth and security aspects into account.

Considerable work has been done in the area of distributed data management [ÖV99], for instance in the area of Distributed Data Mining (DDM) [KC00]. Generally, distributed databases constitute a very important and emerging research area which crucially depends on efficient query processing.

In the following, we present a distributed algorithm which detects intersecting interval sequences, e.g. interval sequences representing complex spatial objects, residing on different local clients. Note that determining pairs of intersecting interval sequences located at the same local site is a rather straightforward task which can be handled independently by the corresponding local clients (cf. Section 9). These locally determined result sets can easily be combined with the global result set determined by the distributed intersection join. The approaches presented in this chapter has been published in [KKPR05b].

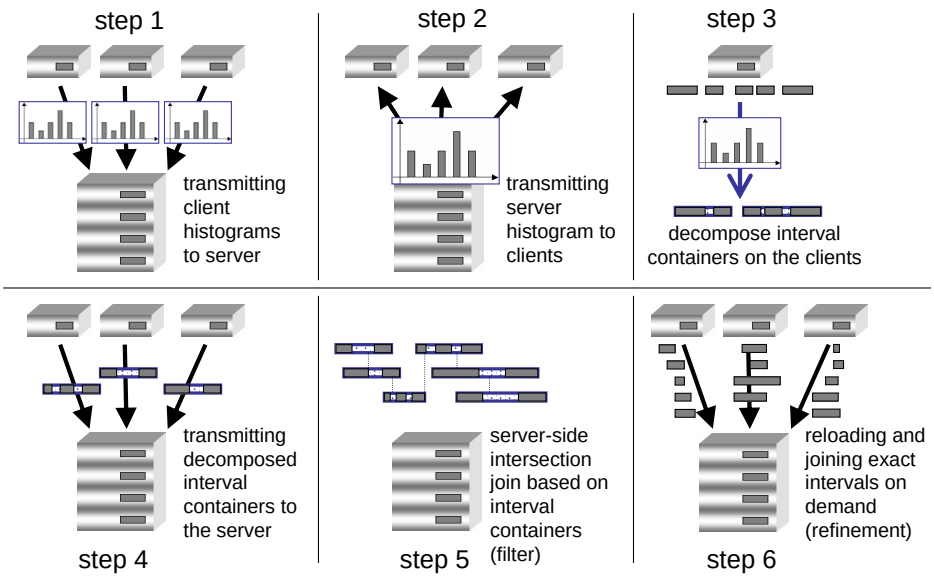


Figure 10.1: Distributed Intersection Join on Interval Sequences.

10.1.1 Concept of the Distributed Join Processing

We start with shortly sketching the complete join process. The distributed join process is performed in multiple steps as depicted in Figure 10.1.

Step 1. At first, all clients collect statistical information, e.g. interval histograms, reflecting the interval distributions of the data residing at their own local site. Then, the clients send this statistical information to the server.

Step 2. At the server site, the local client statistics are merged into a global statistic, reflecting the interval distribution of all local clients. This global statistic is sent back to each client.

Step 3. Each client groups the original intervals belonging to the same voxel-interval sequence together to interval containers. This grouping process is decisively based on the data distribution of the join partners residing on the other local clients reflected by the global statistic minus the own local client statistic. The resulting statistic is not only used for the grouping process but also for a fast filter step on the client sites.

Step 4. If by means of this statistic a global intersection of an interval container cannot be ruled out, the hull of the interval container along with additional aggregated information, i.e. the density of the interval container (cf. Table 8.1) and the number of bytes required for sending the corresponding (compressed) exact information (cf. Section 8.2.2), is sent to the server.

Step 5. The server detects all overlapping interval containers based on their hulls. We say that two interval containers overlap iff their hulls intersect (cf. Section 8.4). Based on the intersection length of the hulls and the density of the interval containers, the server computes a probability that the interval containers intersect, i.e. the probability that they contain at least one pair of intervals that intersect. The pairs of overlapping interval containers are ranked in ascending order according to a cost model, in particular a combination of the determined probability value and the estimated transmission cost of the exact information.

Step 6. Finally, the server iteratively refines the top-listed pairs by fetching the exact information from the local clients, i.e. the (compressed) original voxel intervals. Interval container pairs which belong to objects from which we already know that they intersect, do not have to be refined. Thereby, we can enormously save on the overall transmission cost.

In Section 8.5 we have shown that we can sometimes pinpoint whether two interval container pairs intersect, based on relatively little information, in particular the operators of interval containers given in table 8.1. Unfortunately, as in most cases, the preconditions allowing to detect the intersections in that way does not hold, we will not be able to apply it very often. Nevertheless, it is still helpful if we can predict how probable an intersection of overlapping interval containers might be.

10.2 Intersection Probability

Since we want to integrate this probability model into the first filter step of our join algorithm and we do not want to neglect completely the CPU cost,

its computation should be rather cheap. The probability model introduced in this section is easy to compute and will be applied in various different forms throughout our approach. The model is equal to the $coin-toss$ experiment, i.e. it is a *Bernoulli* experiment. It assumes that the cells within an interval container occupied by object voxels are equally distributed[1].

Theorem 10.1 (intersection probability) *Let C_I and $C_{I'}$ be two interval containers with densities $D = D(C_I)$ and $D' = D(C_{I'})$. Furthermore, let $L = intersection\_length(H(C_I), H(C_{I'}))$. Then, the probability $P(C_I, C_{I'})$ that the two interval containers C_I and $C_{I'}$ intersect is equal to:*

$$P(C_I, C_{I'}) = 1 - (1 - D \cdot D')^L.$$

Proof. *Let x be one of the cells in the overlapping area of both interval containers C_I and $C_{I'}$. Obviously, the probability that this cell is covered by a voxel interval contained in C_I is equal to the density D. Subsequently, the probability that two intervals b and b' from C_I and $C_{I'}$, respectively, intersect at the point x is $P_x = D \cdot D'$. The probability that either x or another cell $y \neq x$ is covered by intervals b and b' from C_I and $C_{I'}$ 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 b and b' share at least one point can be computed as follows:

$$P(b, b') = \sum_{i=0}^{L-1} (D \cdot D' \cdot (1 - D \cdot D')^i)$$

$$= D \cdot D' \cdot \frac{1 - (1 - D \cdot D')^L}{1 - (1 - D \cdot D')} = 1 - (1 - D \cdot D')^L.$$

□

Note that Lemma 8.1 and 8.2 can be derived from the above theorem by setting the overlapping length L to 0 (Lemma 8.1) or setting D and D' to

[1] We neglect the fact that the interval containers represent (parts of) spatial objects and that in this case the corresponding intervals covered by the containers tend to form no equally distributed groups.

1 (Lemma 8.2). Similar to the above reasoning, we are going to derive the probability that one interval container C_{I_0} intersects at least one of n other interval containers $C_{I_1}, .., C_{I_n}$.

Theorem 10.2 (combined intersection probability) *Let C_{I_0} be an interval container that intersects with n other interval containers $C_{I_i}, i \in 1..n$ with a probability of $P(C_{I_0}, C_{I_i})$. Then, the total probability $P(C_{I_0})$ that C_{I_0} intersects with at least one of the other interval containers can be computed by*

$$P(C_{I_0}) = 1 - \prod_{i=1}^{n}(1 - P(C_{I_0}, C_{I_i})).$$

Proof. *The probability $P'(C_{I_0})$ that none of the interval containers $C_{I_1}, .., C_{I_n}$ intersect with C_{I_0} is*

$$P'(C_{I_0}) = (1 - P(C_{I_0}, C_{I_1})) \cdot ... \cdot (1 - P(C_{I_0}, C_{I_n})).$$

Consequently, the total probability $P(C_{I_0})$ that C_{I_0} intersects with at least one of the other interval containers is $P(C_{I_0}) = 1 - P'(C_{I_0})$. □

This intersection probability model can be used for building appropriate object approximations and helps us to minimize the cost of the refinement step. How we can exploit our probability model in detail will be shown in the next two sections.

10.3 Client-Side Approximation of Interval Sequences

The central question is how to group the interval sequence of a client object into interval containers, serving as suitable object approximations. We first introduce probability histograms which are used to estimate the intersection probability of interval containers. Secondly, we present our cost model which takes the probability histograms as well as the transmission cost into account.

Finally, we present a cost-based grouping algorithm $CoDecDJ$, a variant of the $CoDecJ$ algorithm (cf. Section 9.4), which aims at minimizing the overall transmission cost.

10.3.1 Local Intersection Probability

We utilize simple statistics of the interval sequence objects to estimate the probability $P(C_I)$ that any interval container C_I intersects with at least one other interval container located on a different client. In order to cope with arbitrary interval distributions, histograms can be employed to capture the data characteristics at any desired resolution. The expected intersection probability $P(C_I)$ can be determined by using an appropriate intersection probability histogram which reflects aggregated information over all interval sequence objects distributed over all local clients.

Definition 10.1 (intersection probability histogram) *Let* $IB = [0, max \in \mathrm{N}]$ *be a domain of voxel-interval bounds. Let the natural number* $\nu \in \mathrm{N}$ *denotes the resolution, and* $\beta_\nu = \frac{max}{\nu}$ *be the corresponding bucket size of the histogram. Let*

$$\beta_{i,\nu} = [1 + (i-1) \cdot \beta_{\nu,1} + i \cdot \beta_\nu)$$

denote the span of bucket $i, i \in 1, .., n$. *Let further* DB *be a database of interval sequence objects and the function* $\Omega(o_j, b_{i,n})$ *denotes the sum*

$$\Omega(o_j, b_{i,n}) = \sum_i (intersect_{length}(i, b_{i,n})$$

over all intervals i *of the interval sequence object* o_j. *Then,* $\Psi(DB, n) = (n_1, .., n_\nu) \in \mathrm{N}^\nu$ *is called the intersection probability histogram on* DB *with resolution* ν *iff for all* $i \in 1, .., \nu$*:*

$$n_i = 1 - \prod_{\forall o_j \in DB} (1 - \frac{\Omega(o_j, b_{i,\nu})}{\beta_\nu}).$$

In the above definition, we map an interval sequence object to ν interval containers congruent to the histogram buckets, each having a density

$\frac{\Omega(o_j,b_{i,\nu})}{\beta_\nu}$. This density corresponds to the probability that one point $x \in \beta_{i,\nu}$ is intersected by the interval sequence object of o_j. Theorem 10.2 shows that the value n_i in Definition 10.1 reflects the probability that $x \in \beta_{i,\nu}$ is intersected by at least one interval sequence object of the domain DB.

10.3.2 Global Intersection Probability

All local clients send their own intersection probability histogram to the server. The server computes for each client C^j a specific global intersection probability histogram Ψ^j.

Definition 10.2 (global intersection probability histogram) *Let DB_1, .., DB_m be the datasets of m different local clients with congruent intersection probability histograms $\Psi(DB_s, \nu) = (n_1, s, .., n_{\nu,s}), s \in 1, .., m$. Then, the global intersection probability histogram $\Psi^j(\cup_{s=1..m,s\neq j} DB_s, \nu) = (n_1^j, .., n_\nu^j)$ for the client C^j can be computed as follows:*

$$n_i^j = 1 - \prod_{s=1,s\neq j}^{m} (1 - n_{i,s}).$$

Similar to the argumentation following Definition 10.1, the value n_i^j of Ψ^j in Definition 10.2 reflects the probability that $x \in b_{i,\nu}$ is intersected by at least one interval sequence object located at a client C^s where $s \in 1, .., m, s \neq j$.

For the computation of the expected intersection probability $P(C_I)$, we apply our probability model of Section 10.2 in combination with the intersection probability histograms $\Psi(DB, \nu)$. For each histogram bucket $i \in 1, .., \nu$ we separately compute the combined intersection probability $P(C_I)$ (cf. Theorem 10.2) according to an interval container C_I by applying the global intersection probability histogram (cf. Defintition 10.2):

$$P^\Psi(C_I) = 1 - \prod_{i=1}^{\nu} (1 - D(C_I) \cdot n_i(DB))^{overlap(H(C_I)),b_{i,\nu}},$$

where $n_i(DB)$ denotes the accumulated interval density described by the i^{th} bucket $b_{i,\nu}$ of the global intersection probability histogram (cf. Definition 10.2).

10.3.3 Cost Model

The approximation quality has a significant influence on the performance of the multi-step join process. If we adjust the approximation quality too low, for example by taking one-value approximations, the filter step is not very selective, thus, many exact object information has to be requested from the server. On the other hand, if we choose very accurate approximations, the initial transmission cost for sending the aggregated information of the interval containers to the server is very high.

The overall join cost $cost_{join}$ related to an interval container C_I are composed of two parts, the filter cost $cost_{filter}$ and the refinement cost $cost_{refine}$:

$$cost_{join}(C_I, \Psi^j) = cost_{filter}(C_I) + cost_{refine}(C_I, \Psi^j).$$

Filter cost. The filter cost $cost_{filter}(C_I)$ related to an interval container C_I depends mainly on the cost required to transmit the aggregated information of C_I to the server. Furthermore, transmission includes the necessary identifier of C_I, the hull $H(C_I)$ and the density $D(C_I)$. The total size of each transmitted entity for the filter step is constant, thus, we penalize each transmission by a constant c_{trans} which reflects the transmission cost related to one interval container.

Refinement cost. The refinement cost related to C_I depends on whether the server asks for the exact information of C_I during the join process or not. Obviously, the probability that the server asks for the exact information depends on the probability whether C_I intersects at least one interval container or not. Thus, we can estimate the refinement cost as follows:

$$cost_{refine}(C_I, \Psi) = P^{\Psi}(C_I) \cdot cost_{trans}(C_I),$$

where $cost_{trans}(C_I)$ denotes the cost required to transmit the interval container C_I from client C^j to the server.

```
CoDecDJ(C_I, Ψ^j(DB, ν)){
    cost_comp := cost_join(C_I, Ψ^j(DB, ν));
    container_pair := split_at_maximum_gap(C_I);
    C_left := container_pair.left;
    C_right := container_pair.right;
    cost_dec := cost_join(C_left, Ψ^j(DB, ν)) + cost_join(C_right, Ψ^j(DB, ν));
    if cost_comp > cost_dec then
      CoDecDJ(C_left, Ψ^j(DB, ν));
      CoDecDJ(C_right, Ψ^j(DB, ν));
    else
      report(C_I);
    end if;
}
```

Figure 10.2: Decomposition Algorithm $CoDecDJ$.

10.3.4 Grouping Algorithm

Our *cost-based grouping* algorithm $CoDecDJ$, depicted in Figure 10.2, is a greedy approach which is performed in top-down fashion. It starts with a one-value approximation of the input interval sequence, i.e. all intervals are grouped into one large interval container C_I. At first, we search the largest gap of C_I and split it at this gap into two smaller interval containers $C_{I_{left}}$ and $C_{I_{right}}$. As long as the estimated accumulated transmission cost of the resulting interval containers are smaller than the cost according to the un-split interval container C_I, the algorithm is applied recursively to both interval containers $C_{I_{left}}$ and $C_{I_{right}}$.

10.4 Server-Side Join Algorithm

The server-side join algorithm is based on the paradigm of multi-step query processing [BKSS94]. First, we detect all overlapping objects, i.e. overlapping interval containers of the objects. In order to decide whether an over-

lapping object pair intersects, it suffices to detect one intersecting interval-container pair of this object combination. Consequently, all remaining intersection tests according to these two objects can be discarded and the corresponding transmission cost can be saved. Therefore, it would be promising to carry out a fast-intersection test (cf. Section 8.5) for all overlapping interval containers before accessing and testing the exact geometry for any interval container pair. As the preconditions according to the fast-intersection-tests may be too restrictive for many test candidates and does not hold very often, it would be desirable to rank overlapping interval containers according to their intersection probability.

10.4.1 Ranked Refinement Based on Join Probability

Following the discussions above, an overlapping interval-container pair $(C_I, C_{I'})$ should be top ranked if the following conditions are fulfilled:

- the intersection probability of $(C_I, C_{I'})$ is high and
- the cost required to transmit $(C_I, C_{I'})$ from the clients to the server is low.

If the intersection probability of $(C_I, C_{I'})$ is high, then this pair seems to have an intersection and we can expect that further intersection tests between the corresponding objects can be discarded. If the transmission cost of $(C_I, C_{I'})$ is rather small, the overall transmission rate can be significantly reduced, provided that we perform the intersection test for the interval container pair $(C_I, C_{I'})$ as early as possible.

We compute the ranking value for an interval pair $(C_I, C_{I'})$ as follows:

$$rank(C_I, C_{I'}) = (1 - P(C_I, C_{I'})) \cdot (cost_{transmit}(C_I) + cost_{transmit}(C_{I'})).$$

Thereby, the intersection probability $P(C_I, C_{I'})$ between two overlapping interval containers is computed according to Theorem 10.1. Note that all transmitted information is stored on the server site. Therefore, for each interval container C_I which has already been transmitted from a local client to the server, the cost attribute $cost_{transmit}(C_I)$ is set to zero.

10.5 Experiments

In this section, we evaluate the performance of our approach with a special emphasis on the overall transmission cost which is measured in bytes. All experiments were performed on a Pentium 4/2600 machine with IDE hard drives.

10.5.1 Test Datasets

The tests are based on a test dataset CAR which consists of 200 3D CAD objects (voxelized with high-resolution) provided by our industrial partner, a German car manufacturer. These voxelized objects have been linearized via a space filling curve, leading to 200 interval sequence objects. Each of these objects consists of approximately 50,000 voxel intervals. Furthermore, we used an artificial test dataset ART consisting of 1,024 interval sequence objects, each represented by 10,000 voxel intervals. The objects are equally distributed in a range of $[0..2^{27}\text{-}1]$ and the gap lengths inside an object follow a normal distribution. During the experiments, the objects of both test datasets were equally distributed on the available clients.

10.5.2 Grouping

We used two different grouping strategies for forming the interval containers. The $MaxGap$ approach tries to minimize the number of interval containers while not allowing that a maximum gap $G(C_I)$ of any interval container C_I exceeds a given $MAXGAP$ parameter. By varying this $MAXGAP$ parameter, we can find the optimum trade-off between accuracy and redundancy. A one-value interval approximation is achieved by setting the $MAXGAP$ parameter to infinite. If the parameter is set to zero, each interval container is identical to one voxel-interval. Furthermore, we used the decomposition algorithm $CoDecDJ$, where we set the resolution of the used histograms to 10,000 buckets by default.

10.5.3 Client-Side Grouping

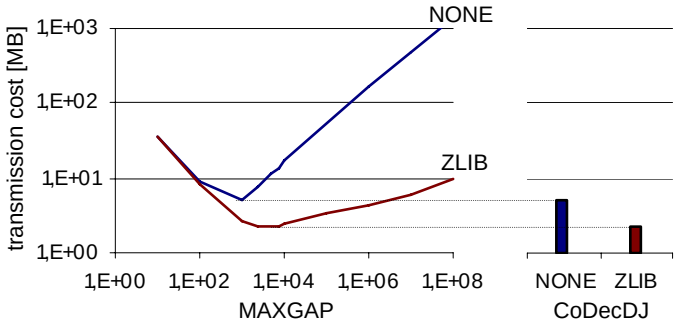
In a first set of experiments, we compare our different client-side grouping strategies to each other. Figure 10.3 shows that for the $MaxGap$ approach we have rather high transmission cost when using too small or too large $MAXGAP$ parameters. If the parameter is too small, many hulls have to be transmitted. On the other hand, if the parameter is very high, the filter selectivity is very bad, leading to high transmission cost during the refinement step of the server-side join algorithm. Furthermore, it is shown that by applying suitable packers for compressing the exact information of the interval containers, these cost can dramatically be reduced. Note that our $CoDecDJ$ approach does not produce higher transmission cost than the "optimal" $MaxGap$ approach that is independent of whether a packer ($ZLIB$ [LZ77]) is used or not (denoted by $NONE$).

In a next experiment, we investigated the dependency of the different grouping approaches for a varying number of clients. Here, we transmitted the exact information of the interval containers in a compressed way. Figure 10.4 shows that our $CoDecDJ$ approach yields optimum results, independent of the number of used clients. Again, for low $MAXGAP$ values, the number of hulls sent to the server is quite large and dominates the overall transmission cost. High $MAXGAP$ values lead to a very small number of interval containers per object, thus, almost all join candidates have to be refined.

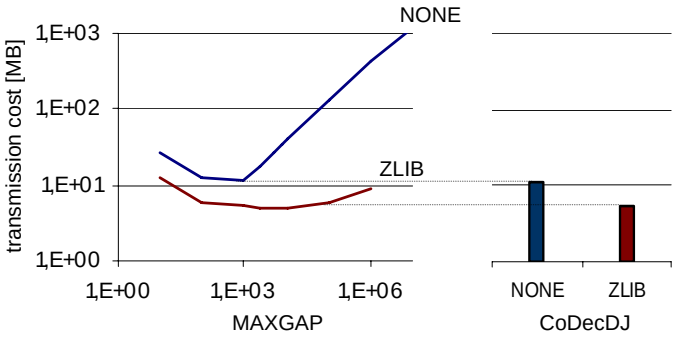
10.5.4 Server-Side Join

In Figure 10.5 it is shown how the intersection probability based ranking function (cf. Section 10.4.1) influences the transmission cost of the refinement step. Therefore, we compare our cost-based ranking approach (CBR) with the following methods which differ in the order in which the join candidates are refined:

- Ordered exclusively by the intersection probability (PBR),



(a) CAR



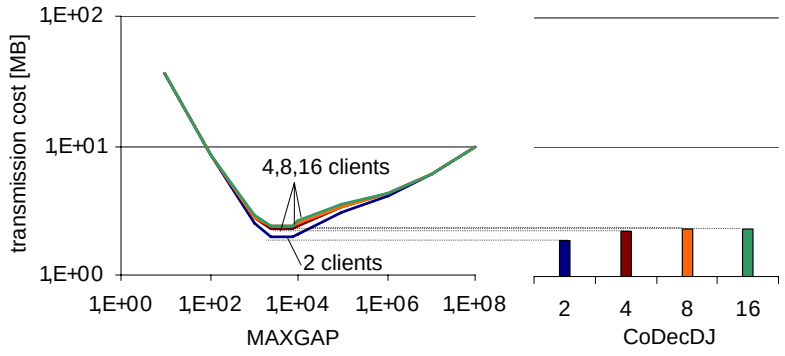
(b) *ART*

Figure 10.3: Grouping strategies using (un)compressed data equally distributed on 4 local clients.

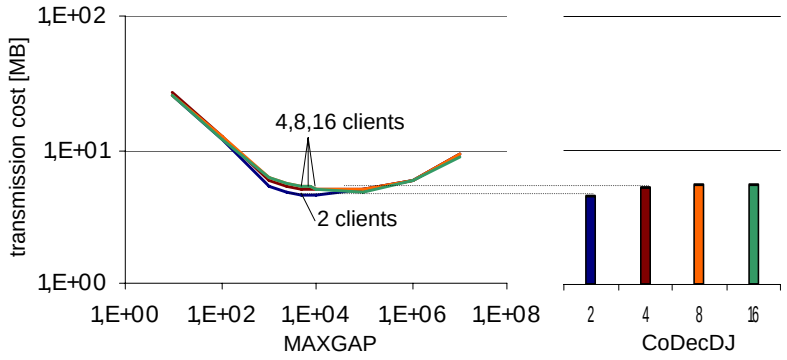
- Ordered exclusively by the transmission cost (LBR),

- Ordered in a randomized order (RND).

This experiment shows that our approach achieves the lowest transmission cost as well as the lowest number of transmission requests. Consequently, our cost-based ranking approach produces the smallest additional communication overhead. Note that we made similar results for the CAR dataset, a varying number of clients, and if we transmit the exact information uncompressed.



(a) CAR

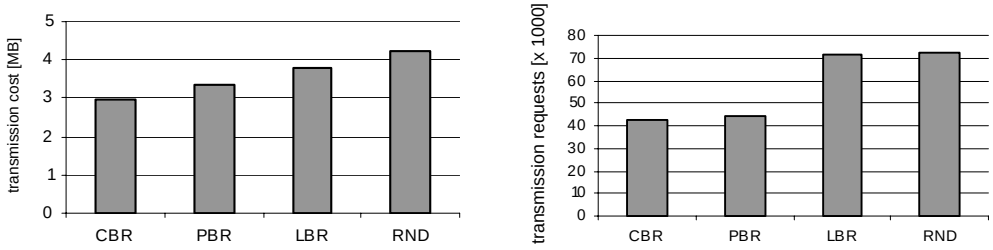


(b) *ART*

Figure 10.4: Different grouping strategies on the two datasets (compressed with *ZLIB*) which were equally distributed on 2, 4, 8 and 16 local clients.

10.6 Summary

In this chapter, we presented an intersection join for distributed complex spatial objects represented by interval sequences. The objects are assumed to be distributed on clients located at different sites. The intersection join is executed at a central server which is connected to all clients via local or wide area networks. The main goal of this approach is to minimize the client-server-communication cost incurred by the server side join process. Our proposed solution is based on generating approximations of the interval sequence data which are transmitted from the clients to the server site for a filter step. In contrast to existing solutions, e.g. error-bound approaches, our statistic driven proposal achieves a good trade-off between the communication cost



(a) transmission cost **(b)** transmission requests

Figure 10.5: Different join strategies (4 Clients, $ZLIB$, $CoDecDJ$, ART).

of the filter and the refinement step. It adapts automatically to different client-server characteristics, e.g. different datasets, varying number of clients or the used compression technique. Another contribution of this work is a cost based strategy for the refinement step. The experiments show that our approach leads to a speed-up of more than one order of magnitude compared to the use of one-value approximations or the use of no approximations at all.

Part III

Enhanced Similarity Search on Time Series

Chapter 11

Introduction

In this part, we introduce the new concept of threshold-based similarity search for time series databases. In Part II we used interval sequence objects 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 "unusual" 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.

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, e.g. stock marketing, astronomy, environmental analysis, molecular biology, and pharmacogenomics. As a consequence, a lot of research work has recently focused on similarity search in time series databases. Similarity search on time series can be classified into the following three objectives: *similarity in time*, *similarity in shape* and *similarity in change* which have been already discussed

in detail in Section 3.3.

In this part, we present efficient and effective similarity search algorithms for time series with special emphasize on *similarity in time*. As mentioned above, we use sequences of intervals in order to approximate our objects. This time we consider time intervals instead of space intervals. A particular interval sequence representation of a time series object is associated with a given amplitude threshold.

11.1 Overview of Related Work

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 (cf. Section 3.3), the type of application (cf. Section 3.4) and the form of representation used for the time series objects (cf. Section 3.7); a survey is given in [KCMP01]. In the following, we will review existing solutions for efficient similarity search on time series and give reasons why they do not suit for threshold-based similarity search.

11.1.1 Measuring Similarity

The most prominent (dis)similarity measure for time series is the Euclidean distance. 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) can fix this problem [KCMP01]. Using DTW to measure the distance between two time series t_1 and t_2, each value of t_1 may be matched with any value of t_2. However, the Euclidean distance as well as DTW are obviously not applicable to threshold-based similarity search because they take the absolute values of the entire time series curve into account rather than tightly focusing on the time series characteristic at "relevant" amplitudes.

11.1.2 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 that way, 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 that lower bound the desired similarity-distance measure applied to the original time series. Nevertheless, the proposed representations are only applicable for the Minkowski metrics or time warping similarity measures but do not support threshold-based similarity computations.

11.1.3 Further Approximation Techniques

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 are defined on this bit level representation that lower bounds the Euclidean distance and DTW. Though, this type of time series representation 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 (cf. Chapter 14), does not allow the user to define the threshold at query time.

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

11.2 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 11.1 (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 $x_i \in \mathrm{R}$ 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. As already mentioned, missing values, i.e. values between two measurements, are estimated by means of interpolation. From the large range of appropriate solutions for time series interpolation, in this section we assume that the time series curves are supplemented by linear interpolation, the most prevalent interpolation method for time series (cf. Section 3.2.2). Throughout the rest of this thesis, $x(t) \in \mathrm{R}$ denotes the (interpolated) time series value of time series X at time $t \in T$.

11.3 Threshold Based Similarity Measure

Time series are usually very large, containing several 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 their mining process does not need the entire course of the time series. Vague "qualitative" course information like whether "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 a specific amplitude spectrum, in particular amplitudes that 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 11.1 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 exceeds the specified threshold τ. We can observe that the time interval sequences derived from $T2$ and $T3$ differs marginally. In contrast, the time series $T1$ shows quite different characteristics caused by the ECG-aberration which indicates the heart disease.

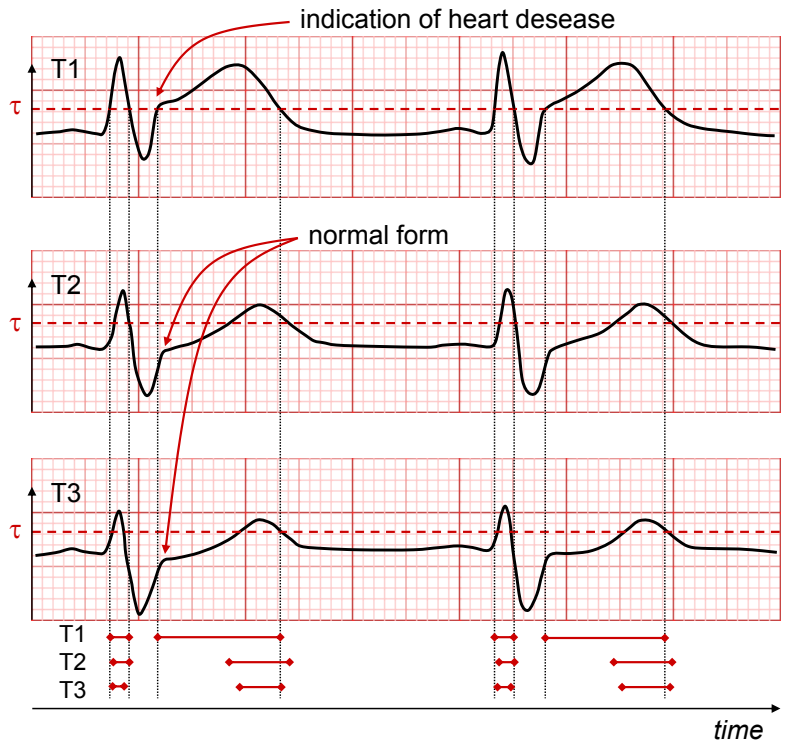


Figure 11.1: Threshold-based detection of risk patients for heart diseases.

The applicability of threshold based time series analysis can be also demonstrated by the example depicted in Figure 11.2. 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 17.2. 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 at the significant amplitude instead of taking the entire time series into account. In fact, we observed in our experiments that we achieve for this dataset the best classification accuracy when considering only the significant amplitude value. The achieved classification accuracy by far outperforms the Euclidean distance in consideration of the classification accuracy.

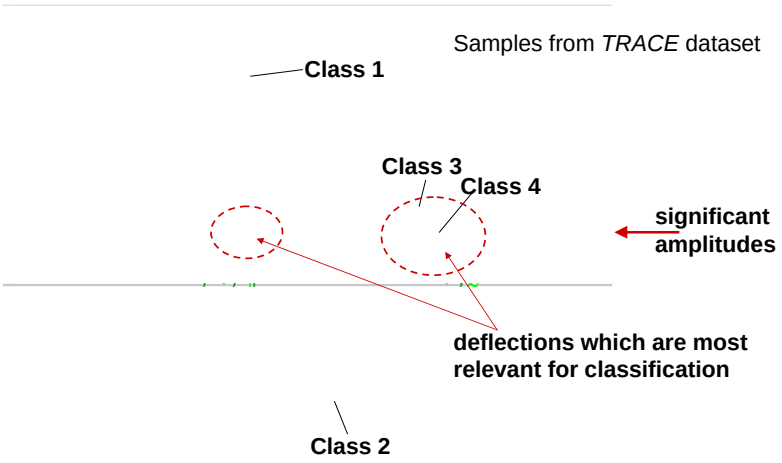
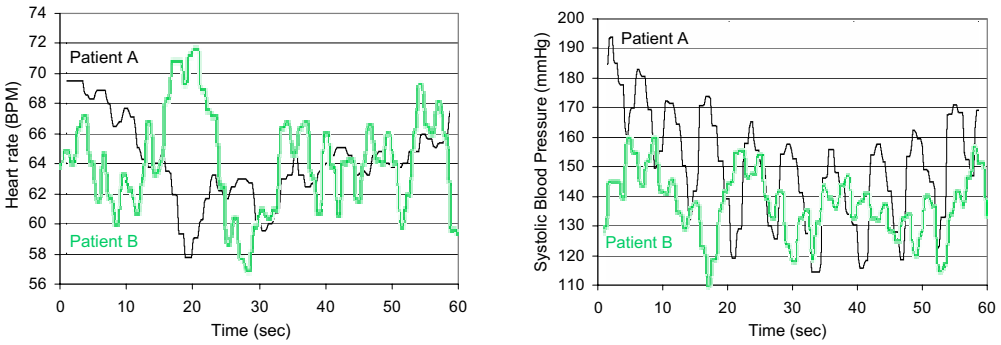


Figure 11.2: Threshold-based classification of time series.

11.3.1 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 at similar time intervals are above or below the given threshold τ. Thus, time series can be considered as similar, even if their absolute values are considerably different, as long as they have similar time frames during which the time series exceeds the specified query threshold τ. Then, the processing of queries like "retrieve all pairs of sequences of ozone concentration which are above the critical threshold of $50\mu g/m^3$ at 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.



(a) heart rate **(b)** systolic blood pressure

Figure 11.3: Patients heart rate and systolic blood pressure after drug treatment.

11.3.2 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 pharma 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 11.3, and the data were stored within a database. In our example, the recorded data of Patient A shows an immediate effect on the drugs and differs significantly 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 w.r.t. certain thresholds which may be significant for the observed attributes.

Application 2 The amount of time series data, e.g. derived from environment observation centers, increases 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 focused by many European research projects in the 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 which exist between meteorological attributes, e.g. humidity, and environmental attributes, e.g. particulate matter (PM_{10}). To know 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 which exceed the threshold $\tau_1 = 75\mu g/m^3$ at a similar time as the temperature reaches the threshold $\tau_2 = 25°C$" could be very valuable. An example is depicted in Figure 11.4, 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 kind of similarity is provided by threshold queries. Obviously, the increasing amount of data to be analyzed represents a big challenge for methods supporting threshold queries efficiently.

Application 3 The analysis of gene expression data is important in molecular biology for understanding gene regulation and cellular mechanisms. 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 of being "up". Genes with similar interval sequence have a

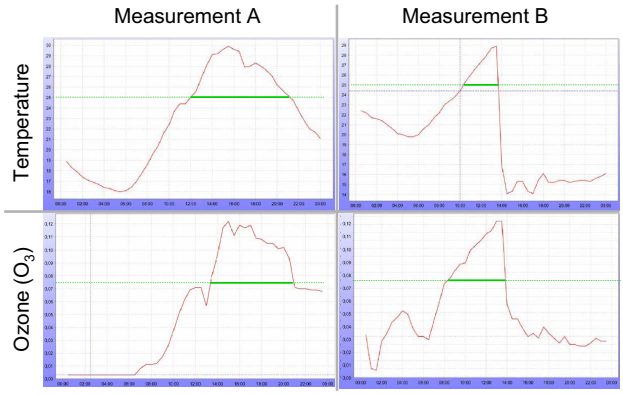


Figure 11.4: Detection of associations between different environmental and climatical attributes.

similar "up" and "down" pattern.

11.3.3 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 preserve 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]. But the compressed information does not support the computation of the threshold-based similarity. The compressed representa-

tions have to be decompressed before we are able to compute the similarities.

11.3.4 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.
- We propose a semi-supervised time series analysis framework adapted from threshold-based similarity measures.

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

The remainder starts with a discussion about several different similarity measures for interval data in Chapter 12. Then, Chapter 13 formally introduces our novel similarity measure and proposes a new query type called threshold query. In Chapter 14 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 15. Based on our new similarity measure, we introduce in Chapter 16 a semi-supervised time series analysis approach. The effectiveness and efficiency of our methods are evaluated in Chapter 17.

Chapter 12

Similarity-Distance Measures for Intervals

The similarity between two objects is usually measured by the distance between the attributes or features describing them. Since the distance is usually a quantitative relation between two points in space, we have to map the attributes or features into a multi-dimensional space, called feature space. The problem is to extract or identify the relevant object features which are required to measure the similarity and which constitute a well-defined feature space. For example, it is obvious that the similarity between two points in time can be measured by their distance in time. But how can we measure the similarity between two time intervals? In this chapter, we will discuss several similarity distances defined on different attributes of intervals.

Figure 12.1 shows two intervals, 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, initially we consider the first three attributes l, u and μ. Based on these three attributes, we can define nine basic distances between the two intervals, as depicted in Figure 12.1. In the following, we will briefly discuss several interval similarity distances which are based on the nine basic distances. A detailed discussion is given in [Joh06].

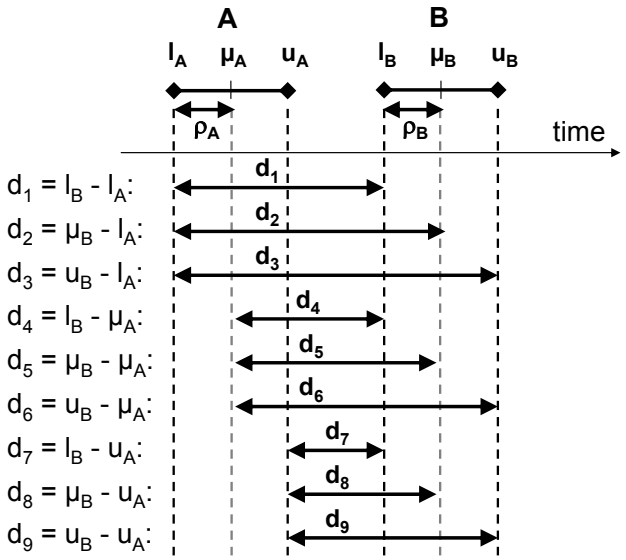


Figure 12.1: The nine basic distances between two intervals A and B.

In the remainder of this chapter, we need the following two notions: Given two intervals A and B. We call B "*upper interval*" and A "*lower interval*" iff the midpoint μ_B of interval B is greater than the midpoint μ_A of interval A.

12.1 Midpoint Measure

The Midpoint measure denotes the distance between the midpoints of two intervals (d_5). An example is given in Figure 12.2(a). This is a simple way of measuring the distance. A disadvantage of this method is that it cannot distinguish between intervals which have different widths but share the same

(a) (b)

Figure 12.2: Interval distance measured by Midpoint measure.

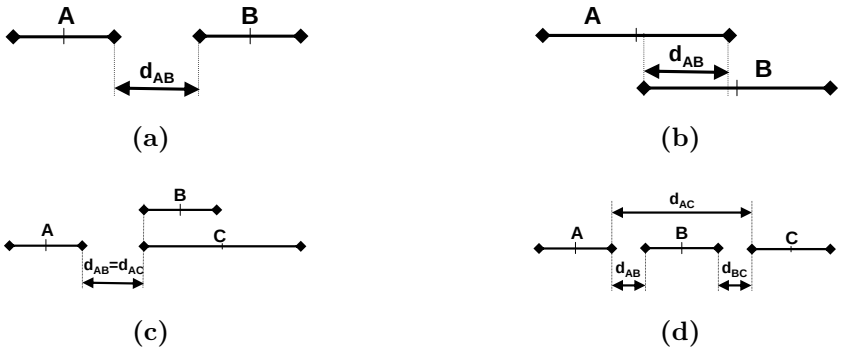


Figure 12.3: Interval distance measured by Gap measure.

midpoint. As shown in the example given in Figure 12.2(b), the intervals B and C have an equal distance to A. Since this distance measure does not take the extension of the intervals into account, it does not intuitively reflect the similarity between intervals.

12.2 Gap Measure

This distance is determined by the gap between two disjoint intervals (d_7). In the case where the two intervals are disjoint, this measure is easy to define (cf. Figure 12.3(a)). However, it becomes more difficult when the intervals overlap. Overlapping intervals could be regarded as being at zero distance from one another since there is no gap. Alternatively, we can consider the absolute distance from the upper bound of the lower interval to the lower bound of the upper interval. Then, the distances between two intervals, shown in Figures 12.3(a) and 12.3(b), could be treated as equal. Since the distance between "equal" intervals is not zero, neither of the two alternatives is appealing for measuring the similarity between intervals.

Intuitively, a larger intersection would argue for a higher similarity. According to the Gap measure, we can describe the distance between two intervals as the lower bound of the upper interval minus the upper bound of the lower interval. One disadvantage of the Gap measure is, when two intervals having the same lower bound, they have the same distance to a lower interval,

regardless of their extensions. In the example shown in Figure 12.3(c), both intervals B and C would have the same similarity to interval A, irrespective of the difference in their length. Another drawback of the Gap measure is that the triangle inequality does not hold, in particular when the distances between intervals are positive as shown in Figure 12.3(d). Anyway, the fact that the Gap measure can result in negative distances disqualifies it from being a well-defined distance measure fulfilling the metric properties.

12.3 Ratio Gap Measure

The disadvantage of the Gap measure not being a metric can be removed by using a ratio rather than a sum measure. The Ratio Gap measure is defined as

$$d_{AB} = \frac{|\mu_A - \mu_B|}{\rho_A + \rho_B}.$$

This measure of course guarantees non-negative values because both, the numerator and denominator are necessarily non-negative. If two intervals are equal, the distance between them is zero. But any pair of intervals with the same midpoint will have a zero distance between them, even if they differ in length.

12.4 Total Distance

The next measure, the *Total Distance*, takes the distance from the extreme of one interval to the extreme of the other interval into account. This is the distance from the lower bound of the lower interval to the upper bound of the upper interval (cf. Figure 12.4(a)). Unlike the Gap measure, there are no immediate complications if the intervals overlap. However, if one interval is completely contained within another interval (cf. Figure 12.4(b)), it is not clear how the *Total Distance* is to be defined. Nevertheless, like the Gap measure, this distance measure suffers from the fact that the length of the intervals are ignored in some cases. As shown in the example in Figure

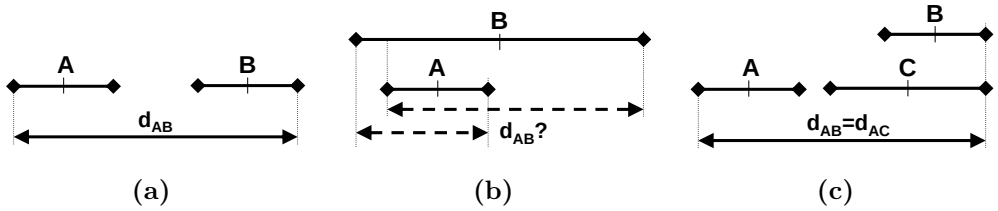


Figure 12.4: Interval distance measured by *Total Distance*.

12.4(c), both intervals B and C of different length have the same distance to interval A. The drawback is that the two closest bounds of two intervals contribute nothing to the distance. An interval pair that touch each other can have the same distance than another interval pair that is widely separated if the extreme ends of both pairs have the same distance.

12.5 Plus-Minus Measures

The next two possibilities relate to the corresponding bounds of two intervals. The Lower Bound measure which is the distance from the lower bound of an interval to the lower bound of the other interval (cf. Figure 12.5(a)) or alternatively the Upper Bound measure which is the distance from the upper bound of an interval to the upper bound of the other interval could be considered. The Lower Bound measure is easily defined for situations as represented in Figure 12.5(a). If the absolute value of the difference between lower bounds is maintained, we can get situations which are not very intuitive as shown in Figure 12.5(b). The strict Lower Bound distance between the interval A and the interval B is the same as the distance from interval B to C. This equivalence holds despite the fact that B's midpoint is closer to A's midpoint than to C's, and that A and B overlap each other while B and C are distinct. The Upper Bound measure can be defined similarly. This time we take the upper bounds of the intervals into account as shown in Figure 12.5(c). Of course, the Upper Bound measure has the same disadvantage as the Lower Bound measure. The main drawback of both distances is that they are concentrated on only one bound of the intervals. Let us consider

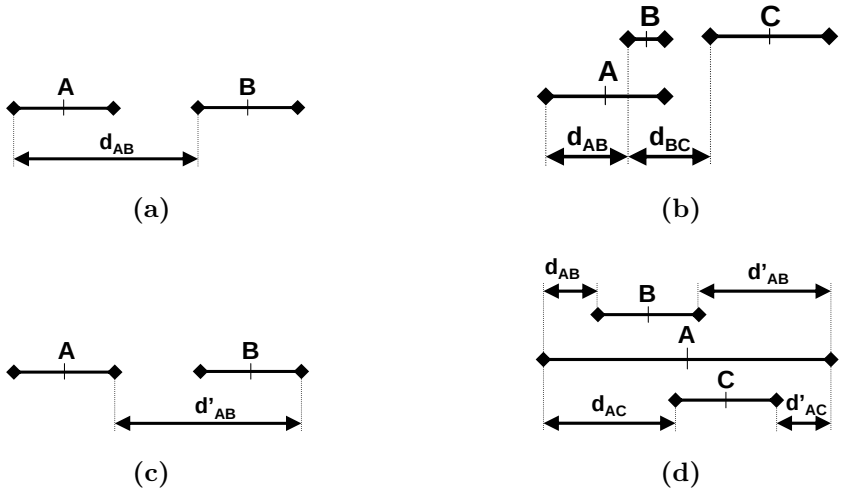
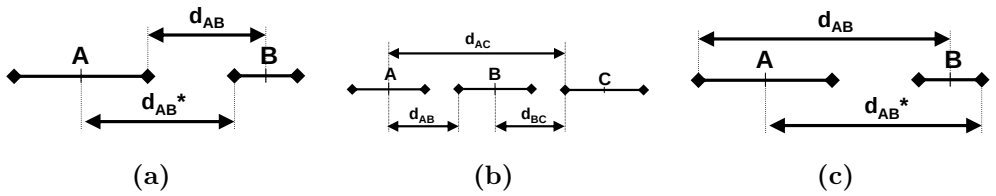


Figure 12.5: Interval distance measured by Plus-Minus measure.

the example given in Figure 12.5(d). Here, the intervals B and C have the same length and the same distance w.r.t. the midpoint of interval A but on opposite sides. It seems reasonable that both should be judged equidistant from the interval A. However, using a measure based on lower bounds, B is considered closer to A, i.e. $d_{AB} < d_{AC}$, while applying the same measure on the upper bounds, C is considered closer, i.e. $d_{AB} > d_{AC}$.

12.6 Mid-Near/Mid-Far Measures

The discussion in the previous section is relevant to another set of measures which take the distance from the midpoint of one interval to one of the endpoints of another interval into account. There are two pairs of measures of this type which can be categorized as "midpoint to nearer endpoint" (Mid-Near measure) and "midpoint to further endpoint" (Mid-Far measure), respectively.



(a) (b) (c)

Figure 12.6: Interval distance measured by Mid-Near / Mid-Far measure.

12.6.1 Mid-Near Measure

In this section, the two nearer endpoint measures, referred to as the Mid-Near measures, are considered. The two relevant measures are illustrated in Figure 12.6(a). These models are closely related to the Gap measure considered in Section 12.2. The same problem encountered in the Gap measure of obtaining negative distances appear also in the Mid-Near measure. In fact, in none of the presented cases, the metric axioms were satisfied. The distance between identical intervals will be $-\rho_X = -(\mu_X - l_X)$, and only zero if their lengths are zero. In contrast to the Gap measure, the symmetry condition is not fulfilled anymore. Finally, the triangle inequality does not hold in the same way as in the Gap measure. Figure 12.6(b) illustrates that the distance from A to C exceeds the summed distance of A to C via B, i.e. $d_{AC} > d_{AB} + d_{BC}$.

12.6.2 Ratio Mid-Near Measures

A possible modification to the measure identified above is to move to a ratio measure rather than an absolute difference measure. This approach is suggested by the parallels between the present measure and the Gap measure. In that instance, the adoption of a ratio measure solved the positivity problem and set up equivalence classes based on shared midpoints. The ratio Mid-Near measure can be computed by one of the following ratios:

$$d_{AB} = \frac{|\mu_A - \mu_B|}{\rho_B},$$

or

$$d_{AB} = \frac{|\mu_A - \mu_B|}{\rho_A}.$$

The metric properties of both of these measures will be the same and both will bear close resemblance to the Ratio Gap measure described in 12.3. Clearly, the distance between an object and itself will be zero. However, non-identical intervals will also have zero distance from one another if they have the same midpoint. Furthermore, like the Mid-Near measure this distance measure is not symmetric.

12.6.3 Mid-Far Measures

The other category of measures which combine a midpoint and an endpoint is called Mid-Far measure. It involves the farther rather than nearer endpoint as depicted in Figure 12.6(c). In the previous section, the relationship between the Gap measure and the Mid-Near measures were noted. There is a similar relationship between the Mid-Far measures and the *total distance*. The metric properties are also similar to those of the *total distance*. The distance between an object and itself will not be zero, except in the case where its length is equal to zero. Contrary to the *total distance*, here the symmetry does not hold.

The discussions so far have involved the nine basic interval distances. However, there are more complex distance measures to consider, an overlap based distance measure and the Minkowski metric which is the most popular one.

12.7 Overlap Measure

This distance measure takes the overlap between two intervals, i.e. the amount of points commonly shared by the two intervals into account. Initially we define the overlap $d_{overlap}$ between two intervals as follows:

$$d_{overlap}(A, B) = \min\{u_A, u_b\} - \max\{l_A, l_B\}.$$

Note, that $d_{overlap}$ results in a negative value, iff both intervals do not intersect.

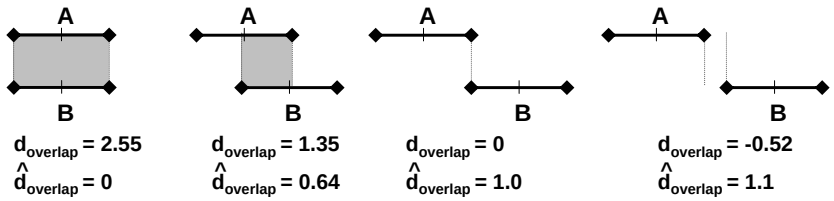


Figure 12.7: Examples of the overlap based interval distance measure.

Usually large intervals have a higher overlap than small intervals. In order to avoid that pairs of large intervals tend to be more similar than pairs of small intervals, we have to normalize the overlap by the length of both intersecting intervals. Furthermore, our distance measure should also be able to assess the similarity between intervals which have no intersection. For this reason we have to incorporate the gap between the intervals. Generally, our distance measure should fulfill the following conditions:

- The distance between equal intervals should be zero.
- The distance between non-equal intervals should be larger than zero.
- The larger the overlap between two intervals relative to their extensions, the smaller their similarity distance.
- The larger the gap between two intervals relative to their extensions, the higher their similarity distance.

All these properties are fulfilled with the following overlap based similarity distance $\hat{d}_{overlap}$ which is defined as follows:

$$\hat{d}_{overlap}(A,B) = 1 - \frac{d_{overlap}(A,B)}{2 \cdot \rho_A + 2 \cdot \rho_B - \max\{0, d_{overlap}(A,B)\}}.$$

Examples of the overlap based similarity distance between two intervals are depicted in Figure 12.7. If the two intervals A and B are equal, the distance $\hat{d}_{overlap}(A,B)$ is equal zero and increases with increasing displacements of the both intervals. As long as both intervals overlap, $\hat{d}_{overlap}(A,B)$ is between 0 and 1. Otherwise, the distance exceeds 1 and increases with increasing extension of the gap between both intervals.

12.8 Minkowski Metric

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.

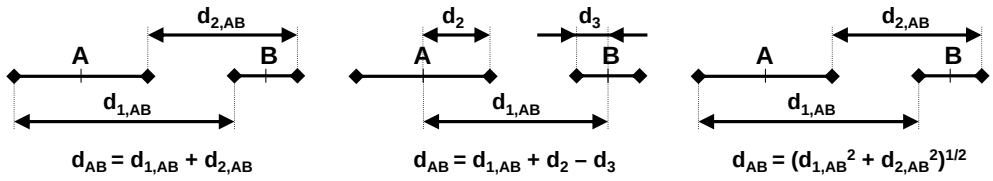
Since using intervals, there are just two mutually independent (orthogonal) axes so that the Minkowski metric can be simplified to

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

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 will consider the three most common Minkowski-metric parameter values $r = 1$, 2 and ∞. The case where $r = 1$ is referred to us as the *city-block metric* which has also been called the *Manhattan metric*[1]. 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* or *dominance metric*. Under this model, only the largest of the component distances contributes to the overall distance.

Now, with this background, specific models can be examined. In the following, we will discuss more closely three different Minkowski models:

[1]The various names are intended to represent the fact that this metric simply adds the component distances together to obtain an overall distance. It is the same as traveling some distance in a city, first traveling along a North-South street and then along an East-West street.



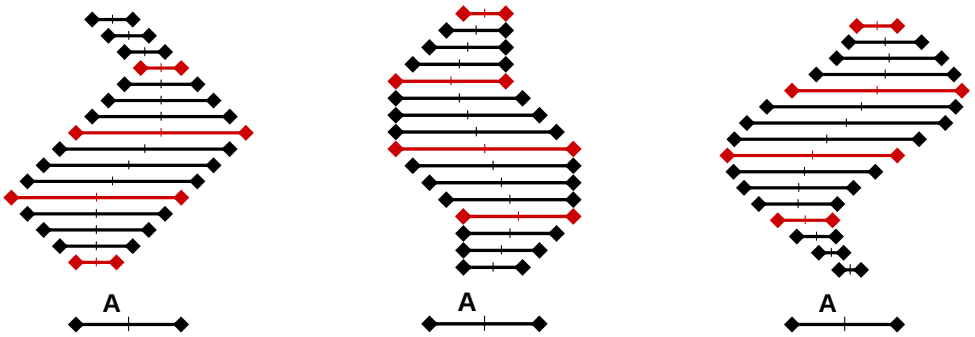
(a) (b) (c)

Figure 12.8: Interval distance measured by Minkowski-Metric.

- The Manhattan metric based on endpoints of intervals (cf. Figure 12.8(a)),
- the Manhattan metric based on the midpoint and latitude of intervals (cf. Figure 12.8(b)) and
- the Euclidean distance based on endpoints (cf. Figure 12.8(c)).

Note that it makes no difference whether the endpoints of the intervals or the midpoint and length attributes are used for the Euclidean distance. This is due to the fact that the space which is spanned by the endpoints of the intervals can be transferred into the space spanned by the midpoint and latitude by a 45° clockwise rotation and proportional scaling by a factor of $\frac{1}{\sqrt{2}}$. Since the Euclidean distance is rotation invariant and the two spaces differ by a proportional rescaling which makes no difference to comparative judgements, it would not make any difference using the endpoints or the midpoints and latitudes.

The advantage of the Minkowski-metrics against the nine basic distance measures is that two parameters that completely define an interval are taken into consideration and not only one parameter. This fact restricts the degree of variance of interval pairs having similar distance. Figure 12.9 sketches for the interval A all intervals having the same distance to A w.r.t. our three Minkowski metrics. Obviously, the parameters of the equidistant intervals are not independent from each other. Intervals that differs in length from A are more centered to A. Contrary, intervals which length is close to the length of A are more displaced. This observation corresponds to the intuitive



(a) Manhattan-Metric based on endpoints.

(b) Manhattan-Metric based on midpoint and latitude.

(c) Euclidean Distance based on endpoints.

Figure 12.9: Equi-distant intervals for different Minkowski-Metrics.

meaning of interval similarity. However, as similarity is a subjective notion, we have to evaluate the various similarity-distance measures empirically.

Nevertheless, the main advantage of the Minkowski metrics against the other distance measures is that the metric properties, i.e. positive definite, symmetry and triangle inequality, are fulfilled. We can use these metric properties for our purpose, in particular for accelerating the query process (cf. Chapter 14). The triangle inequality allows us to apply spatial access methods, like the R^*-tree, in order to speed up similarity queries on intervals.

Chapter 13

Threshold Based Similarity Search

In this chapter, we will formally introduce the novel concept of threshold queries. We consider one-dimensional (univariate) time series represented by a sequences of N value-time pairs $< (x_1, t_1), .., (x_N, t_N) >$ as defined in Section 11.2.

13.1 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 13.1 (Threshold-Crossing Time Intervals) *Let* $X = \langle (x_i, t_i) \in \mathrm{R} \times T : i = 1..N \rangle$ *be a time series and* $\tau \in \mathrm{R}$ *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.

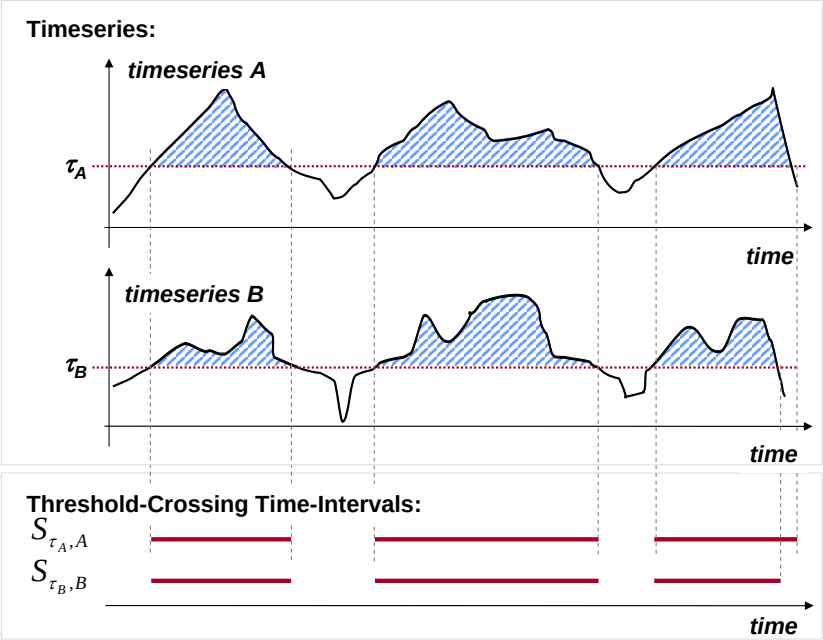


Figure 13.1: Threshold-Crossing Time Intervals.

The example shown in Figure 13.1 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 will 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.

13.2 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 12.1. Following the discussion in Chapter 12, we choose the Euclidean distance based on endpoints 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 will use the Euclidean distance throughout this thesis. An empirical comparison of

the different similarity measures is given in Chapter 17.

Definition 13.2 (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* $d_{int} : (T \times T) \times (T \times T) \rightarrow \mathrm{R}$ *which is defined as follows:*

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

13.3 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 translate this to our time series representations, 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* is defined as follows:

Definition 13.3 (Threshold-Distance) *Let X and Y be two time series and S_X and S_Y be the corresponding threshold-crossing time intervals.*

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

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. But this distance measure has further advantages. Though, 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 global similarity measure.

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

13.4 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 $\varepsilon \in \mathrm{R}_0^+$ 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 $k \in \mathrm{N}^+$ the k most closest objects to Q according to the

used similarity distance measure. Applying our similarity model, we can reformulate these two similarity queries as follows:

Definition 13.4 (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 $\tau \in \mathrm{R}$ and an $\varepsilon \in \mathrm{R}_0^+$ parameter. 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 modification of the distance-range similarity query we can modify the nearest-neighbor similarity query as well.

Definition 13.5 (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 $\tau \in \mathrm{R}$ and a parameter $k \in \mathrm{N}^+$. 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 the 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 $k = 1$.

13.5 Summary

In this chapter, we motivated and proposed a novel query type on time series databases called *threshold query* and introduced two versions of this query

type, the *threshold-based ε-range query* and the *threshold-based k-nearest-neighbor query*. First we introduced a new form of time series representation called threshold-crossing time intervals. This representation which consists of a sequence of intervals 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 which threshold-crossing time intervals are most similar to that of the query time series. This type of query is motivated by several practical application ranges. Examples are discussed in Chapter 11.

Chapter 14

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 which 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 exceeds and falls 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. The work presented in this chapter is published in [AKK+06c].

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.

14.1 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 each time series X. Consequently, each time series object is represented by a sequence of intervals. There are several methods to store intervals efficiently (cf. Section 4.4), e.g. the RI-tree [KPS01]. However, like the other interval access methods, the RI-tree well supports intersection queries on interval data 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 will 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 14.1. 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

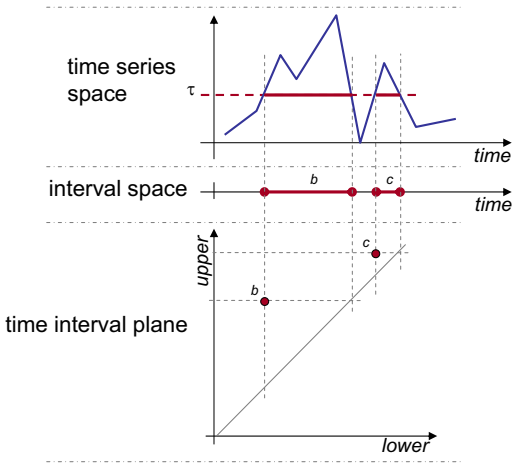


Figure 14.1: Mapping of Time Intervals to the Time Interval Plane.

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 13.2.

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 14.1.

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 will be presented later in Chapter 15.

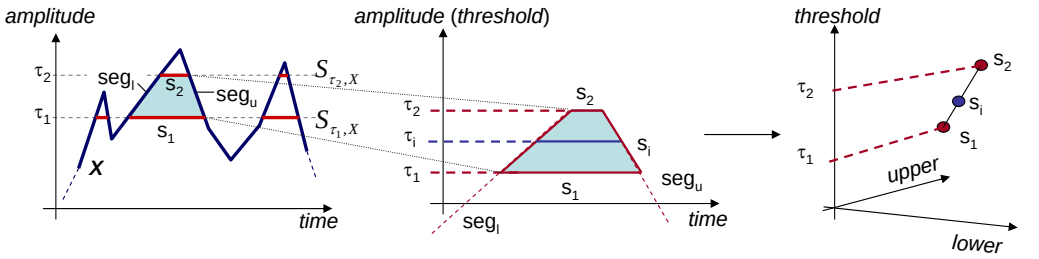


Figure 14.2: Time Intervals in Parameter Space for Arbitrary Threshold.

14.2 Managing Threshold-Crossing Time Intervals for Arbitrary τ

In contrast to the first approach, we will 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 will 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)(cf. Section 3.2.2). 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 14.2a).

Lemma 14.1 *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 from 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 an-*

other 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

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

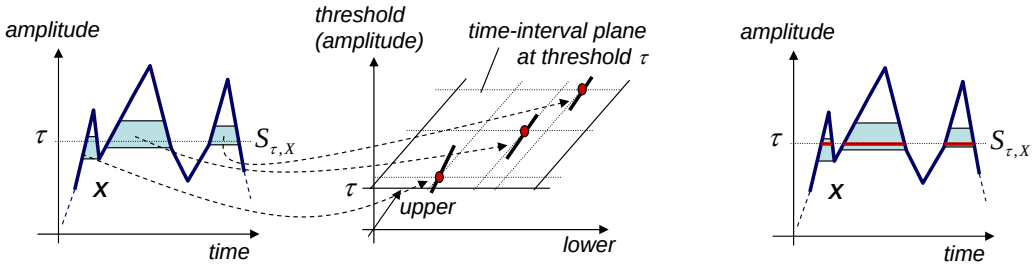
and

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

□

Let us consider the following example shown in Figure 14.2 in order to clarify Lemma 14.1. Figure 14.2(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 14.2(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 14.2c.

Following Lemma 14.1, all time intervals which are bounded by the same time series segments can be transformed into one segment in the parameter space. In order to represent all threshold-crossing time intervals of one time series in the parameter space, we have to



a) building time interval groups (native space)

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

c) threshold-crossing time intervals (native space)

Figure 14.3: Determination of threshold-crossing time intervals from parameter space.

- 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 14.3a).

- Each set is then transformed into a three-dimensional segment in the parameter space (cf. Figure 14.3b).

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 14.3b). The resulting intersection points correspond to the time intervals of $S_{\tau,X}$ as depicted in Figure 14.3c.

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 will introduce a method which enables us to compute efficiently the smallest number of parameter-space segments for a time series.

14.3 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 14.2 *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 13.1, 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 13.1. The validity of Lemma 14.2 w.r.t. the right bounding segment can be shown analogously.* □

Due to Lemma 14.2, 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:

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

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

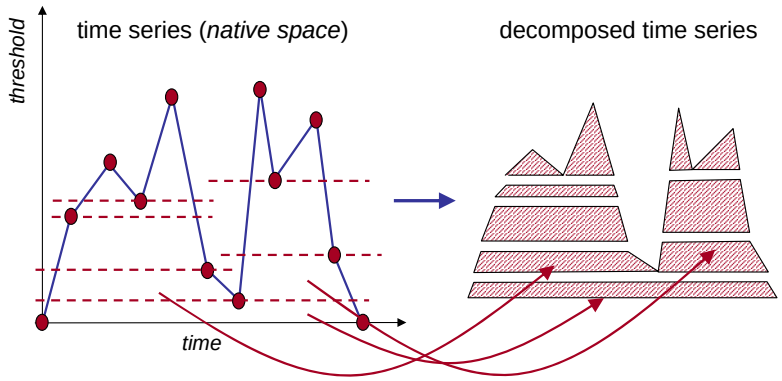


Figure 14.4: Time Series Decomposition.

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 14.4 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 14.6.

Let us illustrate the decomposition algorithm by means of the following example depicted in Figure 14.5. 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. Actually (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 the procedure $compute\_trapezoid(s_2,s_3)$. Then we cut the segment s_2 at the

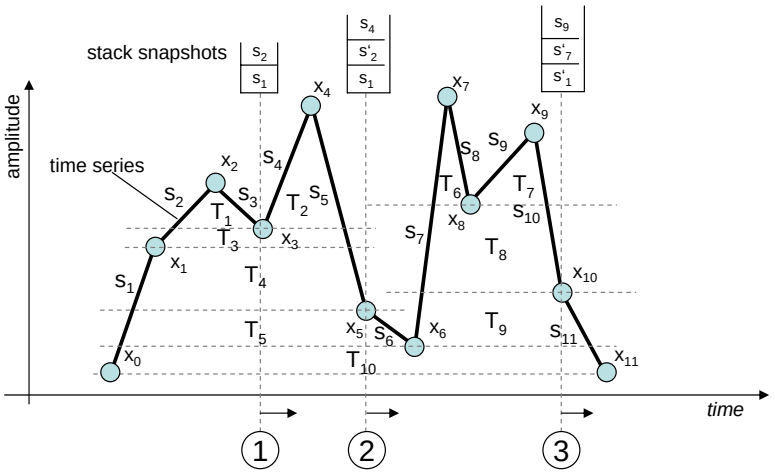


Figure 14.5: Time Series Decomposition Example.

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.

14.4 Parameter Space Indexing

A straight forward approach to manage the time series trapezoids in the native space is to approximate them by a minimal bounding box and insert them into a convenient spatial access method like the R*-tree. The main disadvantage of this naive method is the significant overlap of the trapezoid approximations of the time series objects. Trapezoids that cover similar amplitudes would be grouped together into one page according to their mutual overlap. That means that trapezoids with a small extension along the time axis would be grouped together with trapezoids with a large extension along the time axis. This would be done regardless the fact that two trapezoids which significantly differ in their time extension represent time

intervals that are very dissimilar according to our similarity model. Rather, for our approach we need an indexing method which builds clusters of trapezoids representing similar intervals according to our similarity model. For this reason we propose to index the trapezoids which are transformed into the parameter space.

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

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

14.5 Summary

In this chapter, we presented a novel approach for managing time series data to efficiently support such threshold queries. In particular, we proposed a threshold invariant representation of time series which allows to pre-compute 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.

```
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 14.6: Linear time series decomposition.

Chapter 15

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 13.4 and 13.5). The proposed algorithms have been published in [AKK+06b].

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.

15.1 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 15.1 (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 other time intervals and $\varepsilon \in \mathrm{R}_0^+$ 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 15.2 (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 other time intervals and $k \in \mathrm{N}^+$ be the*

[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.

$\mathcal{D}$	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 15.1).
$NN_S(q)$	The nearest neighbor of q in S (cf. Definition 15.2).
$NN_{k,S}(q)$	The k^{th} nearest neighbor of q in S (cf. Definition 15.2).
$kNN_S(q)$	The k nearest neighbors of q in S (cf. Definition 15.2).

Table 15.1: Notations and operations on time interval sets.

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 15.1 we summarized the most important parameters required throughout the following sections.

15.2 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 15.1 *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:

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

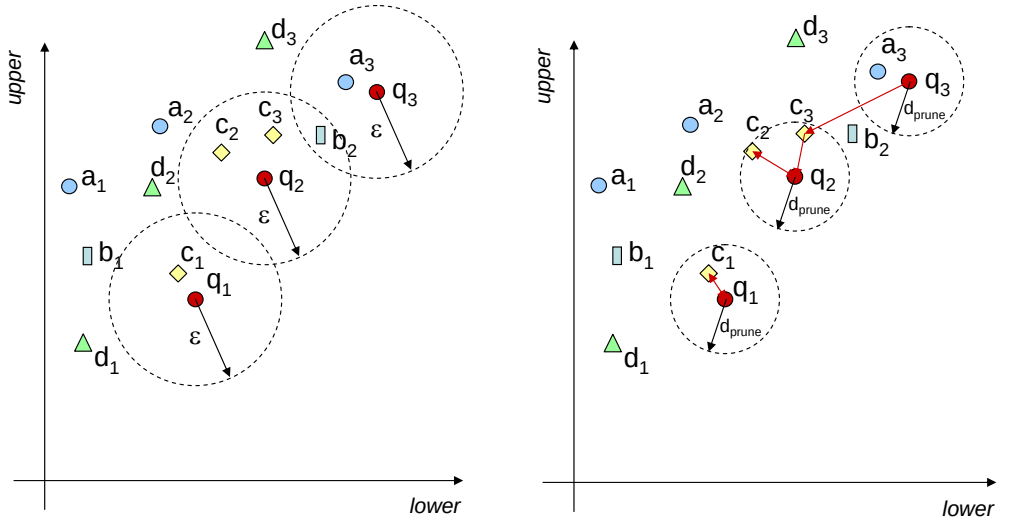
$$> \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot \sum_{q \in S_Q} \varepsilon + \frac{1}{|S_X|} \cdot \sum_{s \in S_X} \varepsilon \right) = \frac{1}{2} \cdot \left(\frac{1}{|S_Q|} \cdot |S_Q| \cdot \varepsilon + \frac{1}{|S_X|} \cdot |S_X| \cdot \varepsilon \right) = \varepsilon.$$

□

An example is depicted in Figure 15.1(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 15.1, 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 15.2 *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 $d_{TS}(S_Q, S_X)$ between Q and any database object X. Then each database*



(a) threshold based ε-range query

(b) threshold based k-nearest-neighbor query

Figure 15.1: Properties of threshold queries w.r.t. object pruning.

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

□

Let us illustrate Lemma 15.2 by means of our example depicted in Figure 15.1(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 both threshold queries aim at keeping the cost required for the expensive 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.

15.3 Threshold-Based ε-Range Query Algorithm

The algorithm for the threshold-based ε-range query is depicted in Figure 15.2. We assume that the threshold-crossing time intervals of the query object Q is 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 refine each candidate X by computing the threshold distance to Q and put

```
TQ^{ε−range}(S_Q, ε, D, S){
    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;
}
```

Figure 15.2: Threshold-based ε-range query algorithm.

them to the result set if $d_{TS}(S_Q, S_X) \leq \varepsilon$.

15.4 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 should be fast, 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.

15.4.1 Lower Bounding Threshold Distance

Now, we will 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 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 15.3 *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 15.2 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 15.3 and defines a lower bound of $D_1(S_Q, S_X)$ for all database objects $X \in \mathcal{D}$ for any $k \in \mathrm{N}^+$.

Lemma 15.4 *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 15.2 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 15.5 *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 15.2 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.

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

□

15.4.2 Pruning Based on Lower Bounding Distance

In this section, we will 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 15.4 and 15.5, 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 15.4 and 15.5:

Lemma 15.6 *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 15.7 *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 $k \in \mathrm{N}^+$. 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 will 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 15.7. 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 15.6). As soon as we have found a pruning distance d_{prune} for which Lemma 15.6 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 15.7. The pruned objects can be omitted from the remaining search steps. The search proceeds by continuing the computation of the next ranking iteration $NN_{k+1,\mathcal{S}}$.

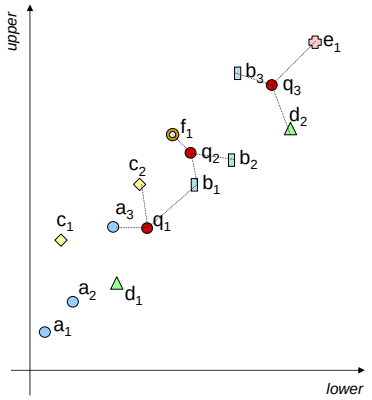
The search algorithm terminates as soon as all object candidates, except for the most similar one (in case of the threshold-based 1^st-nearest-neighbor query), have been pruned from the object list.

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

15.5 Threshold-Based Nearest-Neighbor Query Algorithm

The query algorithm of the TQ_1^{k-NN} query is depicted in Figure 15.5. 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]. Therefore, 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 15.6.

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 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



(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 15.3: Example of the threshold-based nearest-neighbor query.

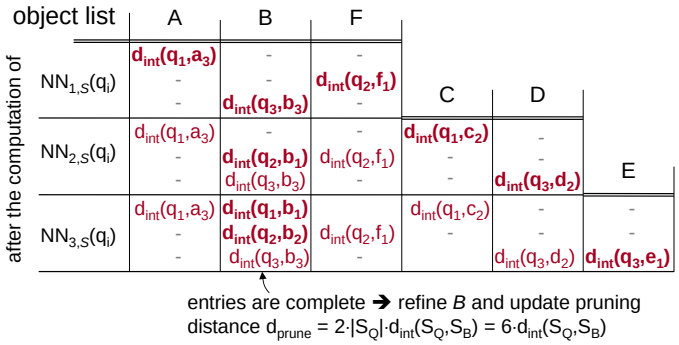
means of the new pruning distance, we decide whether the previous *result* has to keep, and then prune X. If X is closer than the previous *result* according to the threshold distance, we replace the previous *result* with X.

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.

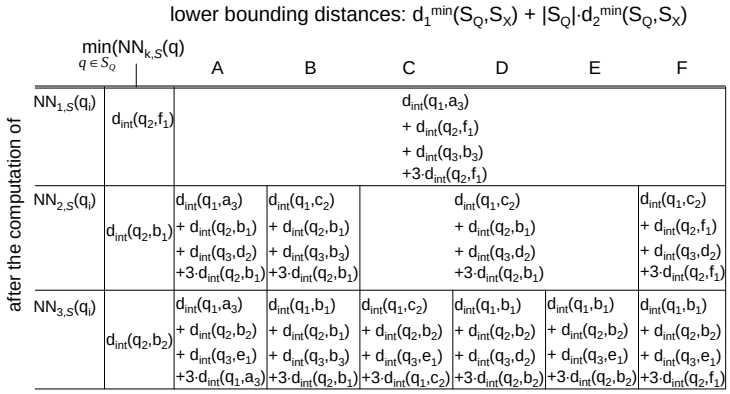
As long as the object list is not empty, we repeat this procedure in the next iterations. Finally, we get the (1^{st})-nearest-neighbor of Q, based on our threshold-based similarity measure.

In order to enable the computation of threshold-based k-nearest-neighbor queries, we have to modify marginally our algorithm. 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.

We will now run through our algorithm with the query example depicted in Figure 15.3. In our example, the query consists of three time-interval plane points $S_Q = \{q_1, q_2, q_3\}$. Figure 15.3(a) shows the time-interval plane $\mathcal{P}$ with



(a) Object list



(b) Lower-bounding distance computation

Figure 15.4: Step-wise lower-bounding distance computation of the threshold-based nearest-neighbor query example.

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 15.3(b) shows the results of the first three query iterations of the incremental k-nearest-neighbor queries $NN_{1,\mathcal{S}}(q_i)$, $NN_{2,\mathcal{S}}(q_i)$ and $NN_{3,\mathcal{S}}(q_i)$. The states of the corresponding object list *object_distList* after each iteration is shown in Figure 15.4(a). Furthermore, we depicted in Figure 15.4(b) 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 15.7. In this case, all database objects have equal lower bounding distances as depicted in Figure 15.4(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 15.7 and 15.6.

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 17.3), we will demonstrate that in average n_q is very small in comparison to the length of the time sequences. Furthermore, we will 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.

15.6 Summary

In this chapter, we developed 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.

```
TYPE Q_ARRAY[N] := ARRAY[N] of DOUBLE;
TQ_1^{k-NN}(S_Q, D, S) {
  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);
  end repeat;
  report result;
}
```

Figure 15.5: Threshold-based nearest-neighbor query algorithm.

Chapter 16

Semi-Supervised Time Series Analysis

The novel concept of threshold based similarity enables us to analyze time series tightly focused to a specific amplitude spectrum, in particular amplitudes which are significant for the analysis. However, the most important issue of threshold similarity is obviously the choice of an appropriate threshold τ. In this chapter, we propose an approach for semi-supervised analysis of time series based on our threshold-similarity measure. The key point of this approach is that a promising query threshold τ is automatically adjusted to the characteristics of a small training dataset. This work has been published in [AKK+06a].

16.1 Introduction

The most important issue of threshold similarity is obviously the choice of the threshold τ. In the example for medical analysis, depicted in Figure 11.1, a suitable threshold τ can be selected by the domain expert which should know about the characteristics of an abnormal ECG curve in case of a cardiac infarct patient. The expert knows best a promising threshold value which can be used to discriminate between patients with a high risk for cardiac infarct

and patients with a low risk. Seldom it is so simple. Often it is difficult to identify the most promising threshold value, even for domain experts. For example, sometimes the "best" threshold value is very close to values which are not very promising. This could make a "blind" justification difficult. Let us again consider the example in Figure 11.1. If the threshold would have been chosen a little bit lower than the depicted threshold τ, all three time series would result in rather similar time intervals and, thus, it might be difficult to discriminate time series $T1$ from the other two time series $T2$ and $T3$.

Generally, in data mining applications an optimal threshold for discriminating a predefined class system by means of threshold-based similarity is not known in advance. As a result, a method for the automatical determination of the optimal threshold using a small number of labeled time series as training set is mandatory. Thus, for cluster analysis, a semi-supervised approach is envisioned. First, the best suitable threshold is determined automatically by means of a small training set. Second, a cluster analysis using the concept of threshold similarity (based on the previously learned, i.e. adapted threshold) is performed in order to detect novel and important patterns. So far, there is no approach to 'optimally' adapt the threshold for a threshold similarity analysis of time series, except the approach which we will present in this chapter.

16.1.1 General Idea of Semi-Supervised Cluster Analysis

In this chapter, we present a novel semi-supervised framework for the cluster analysis of time series using adaptable threshold similarity. This frame work consists of two phases, a training and a clustering phase as depicted in Figure 16.1. In the first phase, the most suitable parameter setting, i.e. the choice of the threshold value, is determined by applying training datasets for the complete clustering process. Our proposed method uses the observation that the classification of some labeled objects leads to different results, depending on the chosen threshold τ, i.e. τ influences the separability of the classes

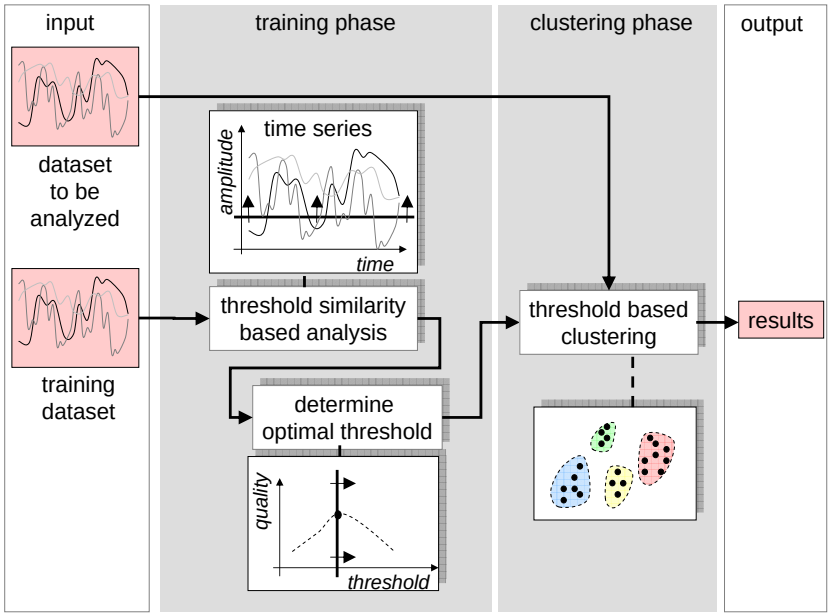


Figure 16.1: Framework for semi-supervised clustering.

which we quantify by a so-called *separation score*. Therefore, we compute the separation score for each threshold of a given training set in a training step first. This results in a quality curve depending on τ. The optima of this curve can give us useful hints how to adapt the threshold for the second phase where the entire dataset is clustered. One might argue that performing a number of clusterings for different thresholds followed by a clustering quality computation can lead to the same results and even eliminates the training phase. However, this ignores the fact that many clustering algorithms have a runtime of $O(n^2)$ or require several iterations until they terminate. Besides, we use only a small training set for calculating the separability score whereas the clusterings would have to be computed on the entire dataset. Furthermore, additional time would be required to analyze whether the clustering results are meaningful or not.

16.2 Related Work

Having defined a distance measure on time series data, we can apply any analysis task. For clustering time series data, most of the various clustering methods proposed in the past decades have been successfully applied. A general overview over clustering methods is given in [HK01]. In addition to the similarity information used by unsupervised clustering, in many cases a small amount of knowledge is available concerning either pairwise (must-link or cannot-link) constraints between data items or class labels for some items. In contrast to clustering which does not use any knowledge except for the similarity information of the data, semi-supervised analysis can profit from this knowledge to guide or adjust the clustering. Obviously, semi-supervised analysis methods achieve better results than their unsupervised counterparts. In recent years, several methods in the area of semi-supervised cluster analysis have been proposed. The main idea of semi-supervised clustering is to determine clusters that are 'immaculate' w.r.t. the class labels of the objects that are analyzed. It can also be considered as using labeled data as feedback in order to help to cluster unlabeled data. Most of the proposed methods for semi-supervised clustering assume that class labels for all objects to be processed are given.

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

Several extensions of existing standard clustering algorithms have been proposed in the literature. A brief survey is given in [EZZ04] describing SPAM, a supervised variant of PAM, SRIDHCR, a greedy algorithm with random restart, SCEC, an evolutionary algorithm, and TDS, a medoid-based top down partitioning algorithm. In [WCRS01], a variant of a k-means based clustering algorithm is proposed. The authors derive constraints from the labeled objects which are used during the clustering. They distinguish be-

tween explicit and cannot-link constraints. In [BBM04a], a k-means based method is introduced which is based on both types of constraints and which exploits the data distribution. The authors of [DBE99] describe an evolutionary method for semi-supervised clustering. This approach has to be initialized with k arbitrary centroids and optimizes a quality measure considering cluster dispersion and impurity. In order to detect a cluster structure that reflects the class distribution of the labeled training data, further methods have been developed which use a standard clustering algorithm by applying an adaptive similarity measure. The authors of [KKM02] propose to apply a complete-link clustering algorithm after replacing the Euclidean distance with the shortest path algorithm. The approach described in [BM03] weights the edit distance using an expectation maximization algorithm to detect approximately duplicate objects in a database. [BBM04b] describes a probabilistic framework for semi-supervised clustering to additionally support several non Euclidian distance measures, e.g. the cosine distance.

All mentioned methods for semi-supervised clustering do not take the threshold-based similarity of time series data into account. In our approach, we use the density-based hierarchical clustering algorithm OPTICS [ABKS99]. However, any clustering algorithm is applicable in our framework.

16.3 Framework for Semi-Supervised Time Series Analysis

As stated above, our main goal is to yield an accurate clustering of the database $\mathcal{D}$ of time series using threshold similarity. In order to choose the optimal threshold value τ, we want to apply a semi-supervised clustering procedure, where we learn the optimal threshold from a small training set $TS \subseteq \mathcal{D}$ of already labeled time series before clustering (cf. Figure 16.2). This learning phase, preceding the clustering phase, is the key step in our framework.

Let TS be the training set containing time series objects that are labeled

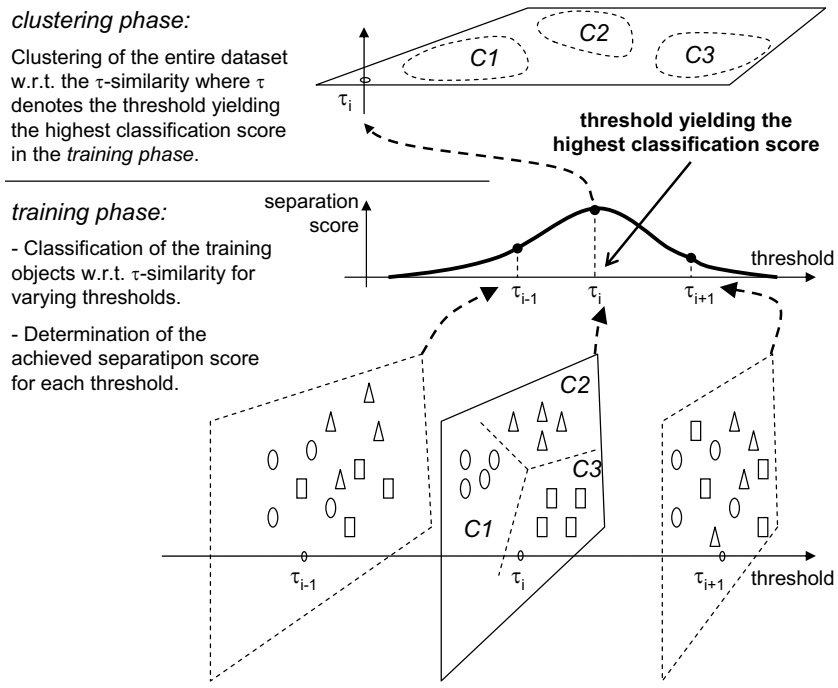


Figure 16.2: General approach.

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

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

Obviously, these classes are excluded from the learning phase, i.e. the learned threshold need not be optimal for these classes. However, threshold values that exhibit a high separability for only a few classes in the training

data are quite often also a good choice for the detection of unknown or missing classes during the clustering phase, as we show in the experimental section (cf. Chapter 17). Thus, by providing only partial information during the training phase, our approach is able to retrieve novel information in the clustering phase. This is contrary to a fully supervised approach, where novel classes cannot be detected.

As already mentioned, the optimal threshold τ_{opt} separates the classes $C_i \in CS$, $1 \leq i \leq k$ in our training set TS in the best possible way. We formalize the separability of a threshold τ by means of a *separation score*. In fact, we measure the separation score of a broad range of possible thresholds. Note that our semi-supervised approach allows different methods for the computation of this separation score. We will present one possibility for semi-supervised cluster analysis on the basis of our framework (cf. Figure 16.1). The separability is measured by using the minimum of all pairwise silhouette widths of each pair of classes C_i and C_j in TS. Though, this is a quite simple heuristic, we achieved promising results in our experiments.

In the following, we first introduce a quality measure for a threshold-based time series clustering (cf. Section 16.4). Afterwards we explain how the optimal threshold is determined (cf. Section 16.5) in order to yield a good clustering of the entire database $\mathcal{D}$.

16.4 Threshold Similarity Based Analysis

In the following, we assume that the user's expectation is modeled by means of a small training dataset. Results for a given query are considered as good, if a lot of results are marked with the same class label as the query time series. The general idea of our technique is to search for threshold values that promise high similarity scores between objects with similar behavior and low scores between objects with different behavior. In the next sections, we will explain how we derive quality values for different thresholds and how we use this information to obtain global high quality values.

16.4.1 Threshold Value Quality for One Class

In this section, we will outline how to detect suitable threshold values for a fixed class. Let us assume that a training data set C_M is given which consists of k classes, $C_M = \{C_1, ..., C_k\}$. For a fixed class C_i we need to determine these threshold values which yield a good separability of C_i from the remaining classes. To evaluate the score of a query for a fixed class C_i and a fixed threshold τ, we pairwise compute the silhouette width [KR90] of C_i and all the other classes C_j, $j \neq i$, in C_M. The silhouette width value is a measure of cluster validity. Assume that we have a clustering of N time series objects into k clusters such that an object X belongs to cluster C of size r. The average dissimilarity between X and all the other entries in cluster C is

$$c(X) = \frac{1}{r-1} \cdot \sum_{Y \in C, Y \neq X} d_{int}(S_X, S_Y).$$

The average dissimilarity of X to all objects Y that belong to another cluster $U \neq C$ of size t is

$$g(X, U) = \frac{1}{t} \sum_{Y \in U} d_{int}(S_X, S_Y).$$

The dissimilarity between X and the closest cluster that is different from C can be defined as

$$v(X) = \min_{U \neq C} g(X, U).$$

The silhouette width $s(X)$ for object X and the average silhouette width $\bar{s}$ for the set in C are defined as

$$s(X) = \begin{cases} \frac{v(X)-c(X)}{\max\{c(X),v(X)\}}, & r \neq 1 \text{ and } r \neq N \\ 0 & r = 1 \text{ or } r = N \end{cases}$$

$$\bar{s} = \frac{1}{r} \cdot \sum_{X \in C} s(X).$$

Silhouette values lie in the range of [-1, 1]. Entries with a silhouette value $s(X)$ close to 1.0 are well clustered because in the sense that the average distance to entries in the same cluster is small compared to the average distance to the closest other cluster. If the silhouette value is smaller than 0, the object is misclassified.

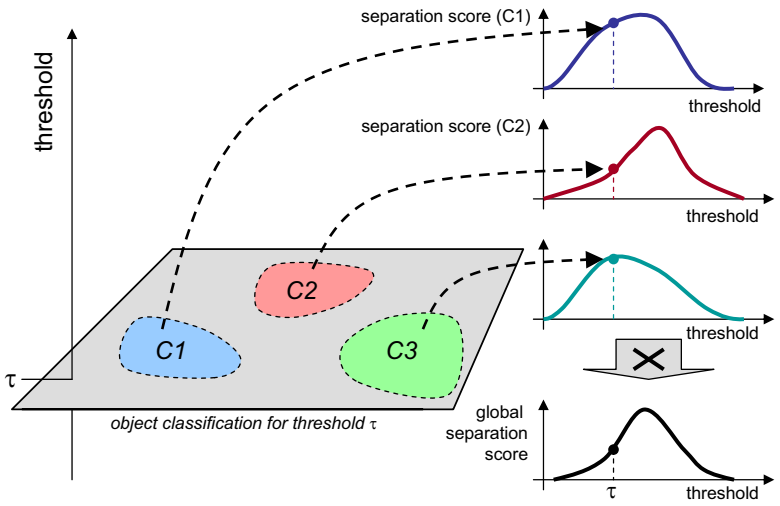


Figure 16.3: Determination of the threshold dependent class separation score.

The minimal average silhouette width of all clusters is used as separation score of a threshold τ. Calculating the separation score for any existing threshold results in a quality measure which estimates the separability score for a training dataset C_M, a class $C_i \in C_M$ and a given threshold τ.

16.4.2 Derivation of a Global Suitable Threshold Value

In the last section, we have developed a quality measure which computes the separation score for each class C_i of our training dataset C_M. Now, we need a suitable combination of all k separation score functions. For our approach, we choose the sum of all score functions, i.e. compute the silhouette coefficient [KR90] for C_M. The global separation score function now reflects the overall separability score of our training dataset for an arbitrary threshold τ. Based on the idea of semi-supervised learning, the global score function gives the user hints to choose the most promising threshold values. The example depicted in Figure 16.3 points up one step of this procedure for a certain threshold.

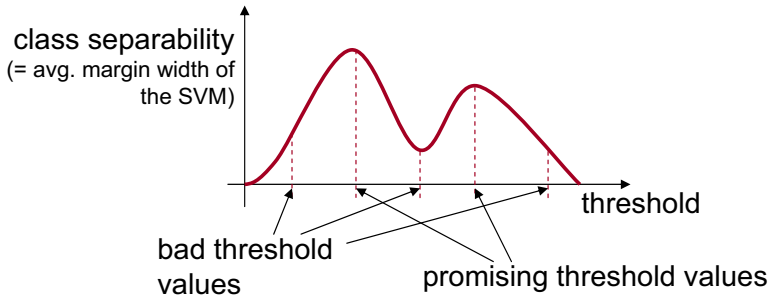


Figure 16.4: Determination of the most promising threshold values.

16.5 Determination of the Optimal Threshold

Having determined the separation score for a range of interesting values, we can consider a quality curve over all these threshold values (cf. Figure 16.4). In such a separability diagram, we plot the examined threshold values along the x-axis and the corresponding separation scores along the y-axis. From this diagram, we can easily determine those threshold values which yield high curve values which are most promising for the clustering step.

In the last step, we picked out one of the most promising threshold values and perform a clustering on the entire dataset based on the threshold similarity. Let us note that the separability diagram not only helps to detect a certain threshold for cluster analysis, but also helps to readjust the parameter setting if the first cluster analysis returns dissatisfying results or the user wants to confirm the validity of the results by alternative threshold parameterizations.

16.6 Summary

In this chapter, we proposed a framework for semi-supervised cluster analysis using adaptable threshold similarity. In particular, we proposed a method to adapt the threshold by learning the optimal threshold from a small training

set in order to yield an accurate clustering of the entire time series.

Chapter 17

Experimental Evaluation and Discussion

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 will demonstrate in Section 17.3 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 14) and our query processing strategy (cf. Chapter 15) by measuring the selectivity, execution time and pruning power of similarity queries. Furthermore, we will experimentally show in Section 17.4 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. Though, our approach can be processed significantly faster than the DTW approach; for some datasets our approach even outperforms the dynamic time warping w.r.t. classification accuracy.

Second, we evaluate our semi-supervised analysis approaches in Section 17.4.

17.1 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/s, a 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.

17.2 Datasets

We applied several real-world and synthetic datasets for our evaluation, one audio dataset (*AUDIO*), two scientific datasets (*SCIENTIFIC*) and a set of well-established test datasets often used as Benchmark (*STANDARD*).

17.2.1 AUDIO

The audio dataset contains time sequences, expressing the temporal behavior of the energy and frequency in music sequences. It contains up to 700,000 time series objects with a length of up to 300 values per sequence. If not otherwise stated, the database size was set to 50,000 objects and the length of the objects was set to 50. This dataset is used to evaluate the performance of our approach (cf. Section 17.3).

17.2.2 SCIENTIFIC

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 [1] 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, containing 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.

17.2.3 STANDARD

The standard datasets are derived from diverse fields and cover the complete spectrum of stationary/non-stationary, noisy/smooth, cyclical/non-cyclical, symmetric/asymmetric etc. data characteristics. They are available from the UCR Time Series Data Mining Archive [KF02]. Due to their variety, they are often used as benchmark for novel approaches in the field of similarity search in time series databases. We used the following four datasets: GUN/POINT (*GunX*), TRACE (*Trace*), CYLINDER-BELL-FUNNEL (*CBF*) and CONTROL CHART (SynCtrl).

The ***GunX*** dataset is a two-class dataset which comes from the video surveillance domain. The dataset 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 replicate 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

[1] www.bayern.de/lfu

return their hands to their sides.

For both classes, we tracked the centroid of the right hand in X-axes. Each instance has the same length of 150 data points (plus the class label), and is z-normalized (mean = 0, std = 1).

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

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

The ***SynCtrl*** data set[2] contains 600 examples of control charts synthetically generated by a process in Alcock and Manolopoulos [AM99]. This dataset consists of the Cyclic pattern subset of the control chart data from the UCI KDD archive (kdd.ics.uci.edu). The data is effectively a sine wave with noise consisting of 6,000 data points. There are six different classes (100 instances per class) of control charts: *normal cyclic*, *increasing trend*, *decreasing trend*, *upward shift* and *downward shift*.

We applied the two scientific datasets and the four standard datasets in order to show the effectiveness of threshold queries and to evaluate our semi-supervised analysis approach. All datasets are summarized with their attributes in Table 17.1.

[2]This dataset was donated by Dr. Robert Alcock.

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

Table 17.1: Summary of Test Datasets.

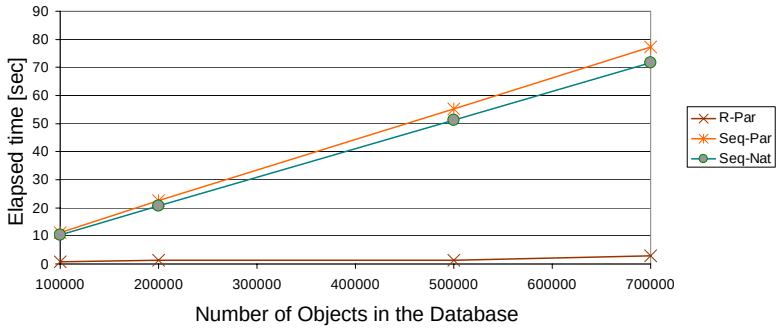
17.3 Performance Results

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.

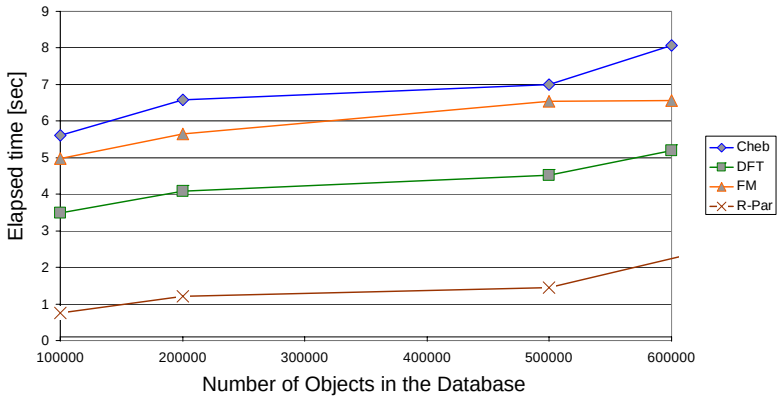
The first competing approach, denoted by 'Seq_{Nat}', works on the native time series data. It corresponds to a sequential processing of the native data. The threshold-crossing time intervals of each time series was computed at query time.

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 [SK98] which adapts the GEMINI framework (cf. Section 3.6) 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



(a)



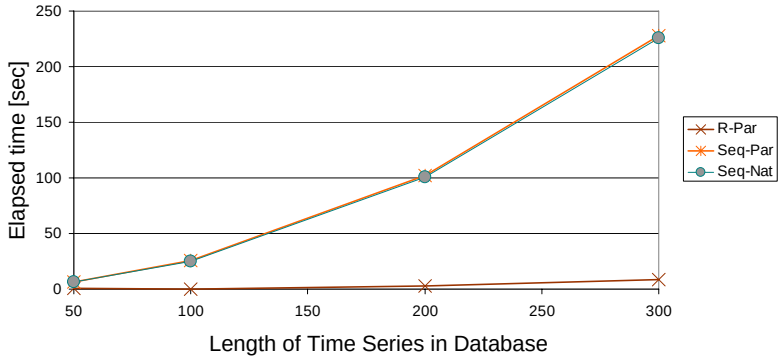
(b)

Figure 17.1: Scalability of the *threshold-query* algorithm against database size.

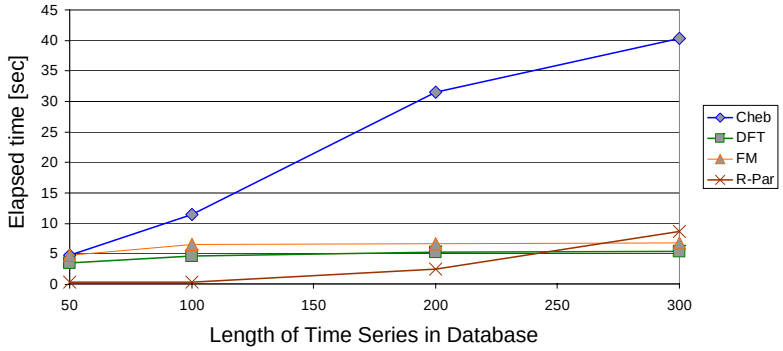
Euclidean distance. They cannot be applied to threshold-based similarity search applications.

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 17.1 exhibits the performance results for each database. In Figure



(a)

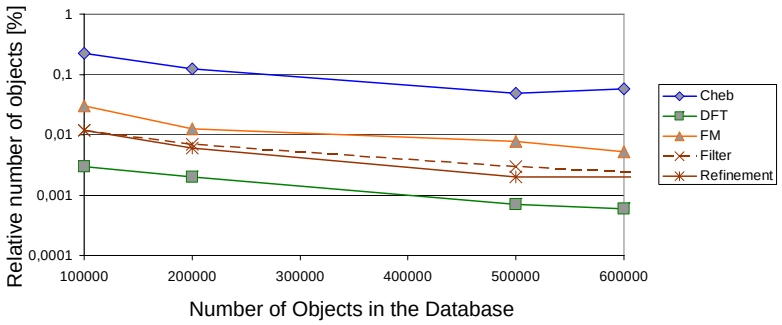


(b)

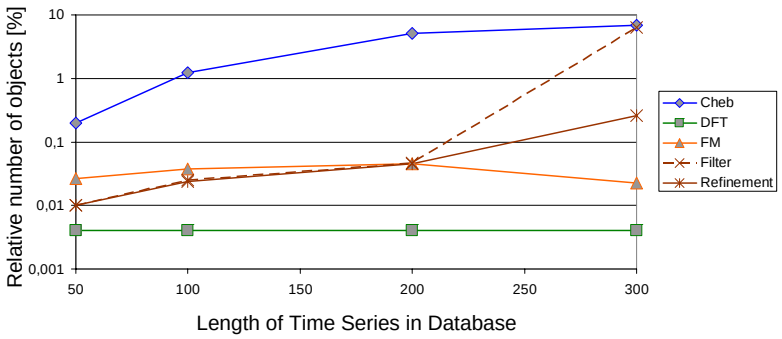
Figure 17.2: Scalability of the *threshold-query* algorithm against time series length.

17.1(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 17.1(b). Yet, our approach even outperforms them by a factor of 4 to 5.

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 17.2. Again, our technique outperforms the competing approaches Seq_{Nat} and Seq_{Par} whose cost increase very fast due to the expensive distance computations (cf. Figure 17.2(a)). In contrast, our approach, like *DFT* and *FM*, scales well for larger



(a) Pruning power for varying database size.



(b) Pruning power for varying time series length.

Figure 17.3: Pruning Power of the threshold-based nearest-neighbor algorithm.

time series objects. For small time series it even outperforms by far the three dimensionality reduction approaches as shown in Figure 17.2(b). If the length of the time series objects exceeds 200, then the both approaches *DFT* and *FM* scales better then our approach. In contrast, *Cheb* scales relatively bad for larger time series. The reason is that the number of required Chebyshev coefficients has to be increased with the time series length for constant approximation quality. 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 dimen-

sionality reduction methods *Cheb*, *DFT* and *FM*. Figure 17.3(a) and Figure 17.3(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 methods, our approach scales well for large databases. For small time series, our approach has a lightly better pruning power then *Cheb* and *FM*. We can observe that the pruning power of our approach decreases with increasing time series length. An interesting point is that the number of candidates to be accessed in the filter step increases faster with larger time series than the number of finally refined candidates. Yet, for the AUDIO dataset the *DFT* method shows the best results 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, 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}. But 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 will demonstrate that, for some applications, our approach is also more effective for data mining tasks then Euclidean distance based similarity measures.

17.4 Evaluation of the Threshold Based Similarity Measure

In this section, we will experimentally evaluate the effectiveness of threshold queries. In particular, we will proof the suitability of our similarity model

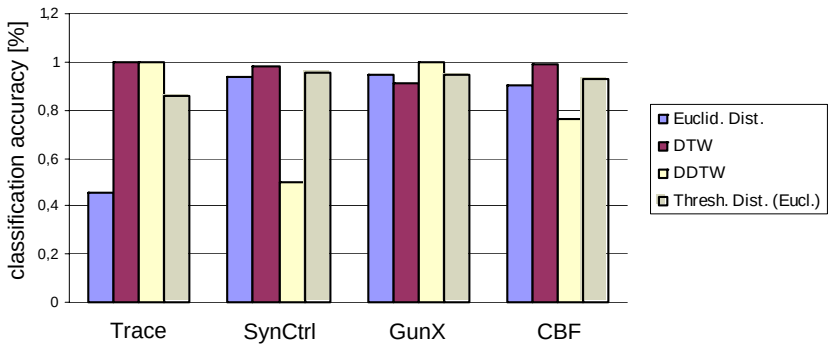


Figure 17.4: Comparison to Traditional Distance Measures.

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.

17.4.1 Comparison to Traditional Distance Measures

In the first experiment (cf. Figure 17.4), we compare our approach to competing similarity measures traditionally used for time series data, the Euclidean distance (*Eucld. Dist.*), Dynamic Time Warping (DTW) and Derivative Dynamic Time Warping (DDTW) [KP01]. Our approach achieves good classification qualities 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.

17.4.2 Comparison of Different Similarity Distances for Time Intervals

First, we will examine different L_p-norms ($p = 1, 2, \infty$) applied to the interval-similarity distance measure d_{int}. Figure 17.5(a) shows the results of the classification accuracy achieved, respectively. As we have expected in Section 12.8, all three L_p-norms show similar behavior w.r.t. the classification accuracy.

For comparison, we also applied other similarity measures including *mid point measure* (Mid Point) (cf. Chapter 12.1), *ratio gap measure* (Ratio Gap) (cf. Chapter 12.3) and our similarity distance measure $\hat{d}_{overlap}$ (Overl.) which takes the overlap between two time intervals into account (cf. Chapter 12.7). However, as depicted in Figure 17.5(b), none of them achieves the quality of the Euclidean distance for all datasets. Except for the *Trace* dataset, they rather show poor effectiveness against the Euclidean distance.

17.4.3 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 17.6 shows the results of the achieved classification accuracy for all four datasets. Additionally, we depicted again 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

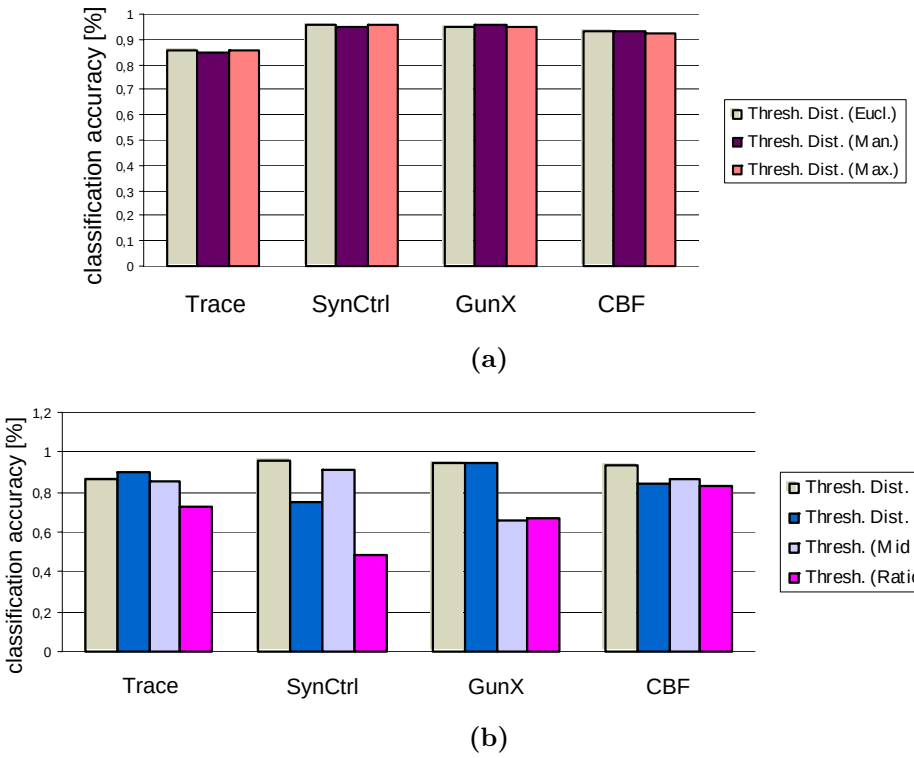


Figure 17.5: Comparison of Different Interval Similarity Distances.

set-kernel distance measure is that one time interval of one time series is matched to all time intervals of the other time series.

17.4.4 Results on Scientific Datasets

Now we will 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 measured also 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.

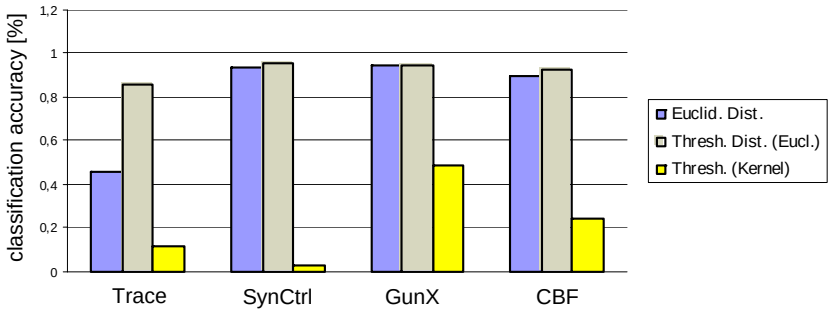


Figure 17.6: Comparison of Different Set Similarity Distances.

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 intuition behind that 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 interestedness, using the SGD database[3]. Indeed, we retrieved functionally related genes for most of the query genes. For example, for query gene CDC25 we obtained the gene CIK3. Both genes play an important role during the mitotic cell cycle. For the query gene DOM34 and MRPL17 we obtained two genes that are not yet labeled (ORF-names: YOR182C and YGR220C, respectively). However, all four genes are participating in the protein biosynthesis. In particular, threshold queries can be used to predict the function of genes whose biological role is not resolved yet.

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

17.5 Evaluation of the Semi-Supervised Time Series Analysis

In this section, we investigated the effectiveness of semi-supervised threshold queries which are used to find the optimal threshold value by means of a

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

training dataset. We applied the density-based clustering method OPTICS [ABKS99] for the cluster analysis step. We used OPTICS due to its robustness w.r.t. data distribution and parameter setting. Again, let us note that any other clustering method is also applicable.

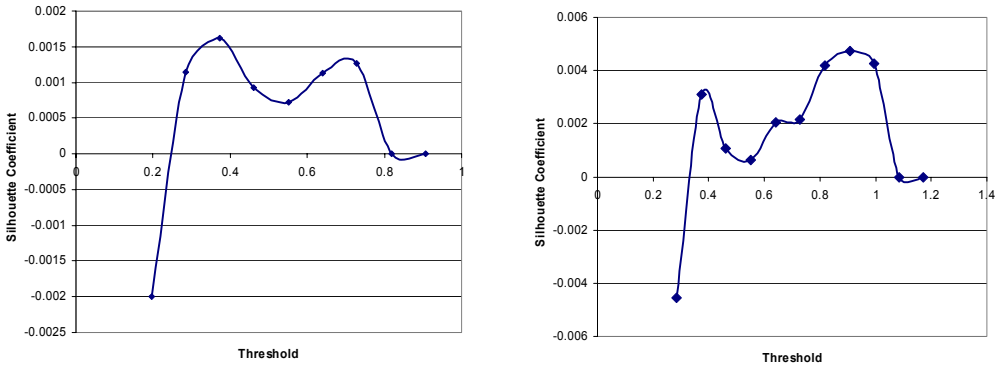
17.5.1 Evaluation of Threshold-Based Separation Score

At first, we are interested in how the optimal threshold values change when the expected results differ, i.e. when the focus of the query changes. The following experiments were performed on the datasets *GDS30* and *GDS38* from Gene Expression Omnibus (GO)[4]. For the first experiment, we used the GO functional classes on level 4. Afterwards, we changed the focus of our queries to the GO level 5. As expected, we obtained different separation score curves as depicted in Figure 17.7. These results raise the question whether the computed optimal threshold values indeed yield good results on the whole dataset or not. To evaluate this, we clustered the time series for varying threshold values and determined the rand index [HBV01]. For example, the threshold value 0.73 which corresponds to a high separation score on the *GDS30* dataset for GO level 4 resulted in a rand index equal to 0.94. Contrary, when using a threshold value of 0.2, the rand index decreased to 0.86. Similar results were observed for other levels, for other threshold values, and on other datasets.

17.5.2 Evaluation of the Cluster Quality

Last but not least, we evaluated the cluster quality for several datasets (from SCIEN_GEX and STANDARD) using our semi-supervised analysis approach. We compared our method to another similarity measure, in particular the Euclidean distance, denoted as(*Eucl. dist.*). The results of the comparison are depicted in Figure 17.8. Figure 17.8(a) depicts the *rand index* [HBV01] and Figure 17.8(b) depicts the average *entropy* [HBV01]. The higher the rand index the higher the clustering quality, whereas high average entropy

[4]http://www.ncbi.nlm.nih.gov/geo/



(a) GO level 4 **(b)** GO level 5

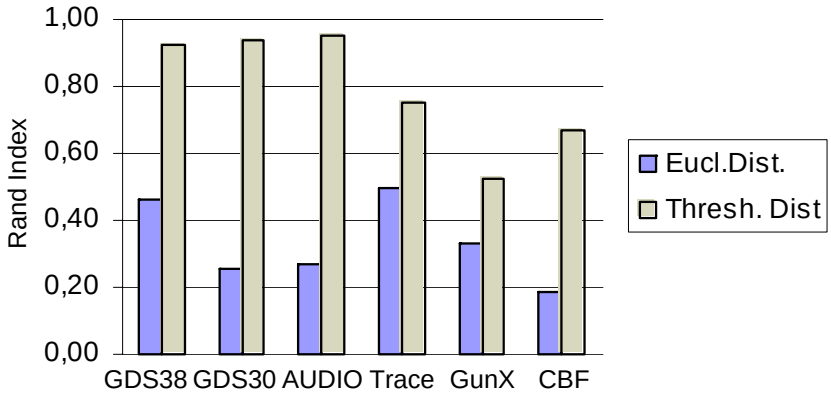
Figure 17.7: Separation score curves for different GO levels (GDS.

values indicate low clustering qualities. The rand index results show that the threshold-based clustering analysis always outperforms the analysis based on the Euclidean distance in terms of effectiveness. For the gene datasets *GDS38* and *GDS30* the Euclidean distance achieves slightly better results w.r.t. the entropy but fails drastically for the STANDARD datasets *AUDIO*, *Trace*, *GunX* and *CBF*, while our approach still yields good results. This underlines the capability of our semi-supervised analysis approach.

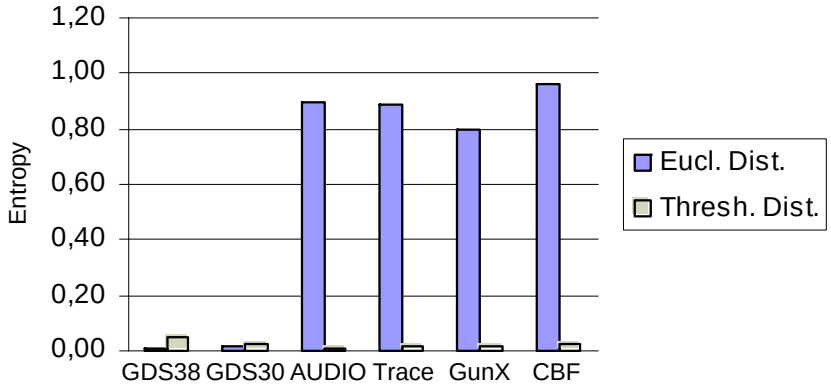
17.6 Summary

In a broad experimental evaluation, presented in this chapter, we demonstrated the importance of the new concepts proposed in the previous chapters. In particular, we demonstrated that threshold queries can be successfully applied to several applications. Furthermore, we examined the scalability of our proposed algorithms and compared it to straightforward approaches. The evaluation of the query algorithm proposed in Chapter 15 clearly demonstrated the power of our pruning strategy which is mainly responsible for the achieved query performance.

In the experimental evaluation of the semi-supervised time series analysis, we have shown that our proposed approach yields valuable clustering results,



(a) Rand Index



(b) Entropy

Figure 17.8: Effectiveness comparison to the Euclidean Distance.

even if only partial information is available for adapting the threshold to an optimal value. Beside the analysis of a dataset, according to specific class labels our approach can help to find unknown but potentially useful knowledge.

Part IV

Conclusions

Chapter 18

Summary and Future Directions

The increasing advancement in sensor technology and the growing progress of recording more and more complex objects and processes of our real life requires computational assistance on the preservation, recovery, analysis, and reporting of a very large quantity of complex data. Modern information systems which can cope with complex data open up new perspectives for several scientific disciplines and engineering facilities. Present data retrieval methods have to be adapted to this emerging evolution. The methods and concepts presented in this thesis contribute to the solution of novel challenges for retrieval algorithms, in particular for complex spatial and temporal data. This chapter summarizes the main contributions of this thesis (Section 18.1) and shows potentials for future research directions (Section 18.2).

18.1 Summary of Contributions

The rapidly increasing amount of complex structured objects stored and organized in databases is challenging for query procedures retrieving relevant information out of the collected data. The effective and efficient processing of complex objects are indispensable for many modern application areas,

such as geographical information systems, digital-mock-up, computer-aided design, medical imaging, molecular biology or real-time virtual reality applications, e.g. haptic rendering. This thesis contributes in the field of query processing on complex structured objects. New and original solutions for collision queries on three-dimensional CAD-parts and GIS data as well as similarity queries on time series are proposed. In the following, we give a detailed summary of these contributions.

18.1.1 Complex Spatial and Temporal Data (Part I)

Part I of this thesis provides a brief and rather general overview over existing models of data representation for spatially extended objects and time series objects. We give a short overview of modeling and managing spatial and temporal data. In particular, we motivate the interval based representation and show that interval sequences are a suitable basic data type for both, spatial and temporal data. The advantage of intervals is that they are very easy to handle and can be easily integrated into commercial database management systems. In addition, efficient access methods which are applicable for fully-fledged object-relational database systems exist for interval data.

18.1.2 Spatial Query Processing for Complex Structured Objects (Part II)

In this part, we showed how to accelerate spatial intersection queries performed on highly resolved spatial data and represented by means of interval sequences. The basic concept of the presented techniques was to take statistics about the data distribution into account.

Using Statistics to Accelerate Intersection Queries on Complex Spatial Objects. Since the interval based representation of spatial objects can be suitably organized within object-relational database management systems we started with the introduction of statistics which help us to minimize the overall navigational cost of space partitioning relational access

methods. First of all, we demonstrated that using our proposed statistic based decomposition algorithm *Decompose()*, grouping the replicating representations of the query object, achieved an acceleration of the query process by 30% to 300% compared to queries performed on the traditional relational access methods. In addition, we showed that our efficient access methods applied to commercial database systems even suffices to provide haptic rendering systems with real-time contact-force computation in very large and complex structured virtual 3D-worlds. Thereby, we obtained a frame rate of about 1,000 Hz for the haptic rendering loop which is by far sufficient for many haptic rendering applications.

Applying Statistics for Decomposing Highly Resolved Spatial Objects. Next, we introduced a cost-based decomposition algorithm for linearized high-resolution spatial objects which helps to range between the two extremes of one-value approximations and the use of unreasonably many approximations. We presented interval containers as a new concept and showed how we can efficiently store them by means of data compression techniques. In particular, we introduced a quick spatial data compressor $QSDC$ in order to emphasize those packer characteristics which are important for efficient spatial query processing, namely good compression ratio and high unpack speed. Furthermore, we presented a cost-based decomposition algorithm for complex spatial objects, called $CoDec$. $CoDec$ takes the decompression cost of the interval containers and their access probabilities into account. This decomposition algorithm is applicable for different spatial index structures, data space resolutions and compression algorithms. We showed in a broad experimental evaluation that our new approach, i.e. the combination of $CoDec$ and $QSDC$, accelerates the Relational Interval-tree (RI-tree) and the Relational Quad-tree (RQ-tree) by up to two orders of magnitude.

We introduce efficient intersection joins for complex rasterized objects based on a cost-based decomposition algorithm $CoDecJ$, an adaption of the decomposition algorithm $CoDec$. The cost model again takes the actual data distribution reflected by statistical information and the used packer characteristics into account. In a broad experimental evaluation on real-world

geographical and 3D CAD datasets, we demonstrated the efficiency of our new spatial join algorithm for complex rasterized objects. It is experimentally shown that *CoDecJ* builds interval-sequence approximations having the best trade-off between redundancy and accuracy and good adapts to the characteristics of the datasets and the used compression method. As a result, *CoDecJ* gains a speed-up of the join queries of about one to two orders of magnitude.

Applying Statistics for Joining Complex Spatial Objects Distributed Over Several Clients. In addition, an intersection join for distributed complex spatial objects represented by interval sequences is presented. The objects were assumed to be distributed on clients located at different sites. The intersection join was executed at a central server which was connected to all clients via local or wide area networks. Our proposed solution is based on generating approximations of the interval sequence data at client site which were transmitted from the clients to the server site for a filter step. The main goal was to minimize the client-server-communication cost of the server site join process. The main contributions of this approach are

- an intersection probability model which is used to build suitable approximations at client side and
- a cost-based refinement strategy taking the intersection probability and the transmission cost into account.

In contrast to existing solutions, e.g. error-bound approaches, our approximation technique *CoDecDJ* achieves a good trade-off between the communication cost of the filter and the refinement step. It adapts automatically to different client-server characteristics, e.g. different datasets, varying number of clients, or the used compression technique. Furthermore, the experiments show that the cost-based refinement achieves the lowest transmission cost as well as the lowest number of transmission requests.

18.1.3 Enhanced Similarity Search on Time Series (Part III)

In the third part of this thesis we motivated and proposed a novel similarity model for time series called *threshold similarity*. In contrast to traditional similarity models including the Euclidean distance or dynamic time warping, the proposed similarity measure allows us to focus the observation of time-series characteristics at a certain amplitude value. The advantage is that we can take relevant amplitudes into account while suppressing less relevant amplitude values. This kind of similarity search on time series has been insufficiently addressed so far by other approaches or not yet addressed at all.

Based on the new similarity measure we defined two similarity queries called threshold queries, the *threshold-based ε-range query* and the *threshold-based k-nearest-neighbor 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 intervals reflecting the behavior at a certain time-series amplitude τ. More exactly threshold-crossing time intervals of a time series represents the interval sequence of elements that have a value above the threshold τ. We mentioned several practical application ranges 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 of our proposed algorithms in comparison to competing approaches. Furthermore, we experimentally evaluated the effectiveness of our novel similarity model by means of classification accuracy. It can be shown that in fact the new similarity measure can be successfully applied to several datasets and outperforms the traditional similarity measures. In addition, we discussed several other variants of amplitude, focused on similarity search and experimentally showed the superiority of our proposed model.

Based on the threshold-similarity model, we presented a framework for

semi-supervised cluster analysis using adaptable threshold similarity. In particular, we proposed a method to adapt the threshold by learning the optimal threshold from a small training set in order to yield an accurate clustering of the entire time series. In our experimental evaluation, we showed that our proposed approach yields valuable clustering results, even if only partial information is available for adapting the threshold to an optimal value. Beside the analysis of a dataset according to specific class labels, our approach can help to find unknown but potentially useful knowledge.

18.2 Future Directions

At the end of this thesis, let us emphasize the potentials of the proposed methods for query processing on complex objects.

For spatial query processing based on interval representations, we see the following opportunities for future research:

- Currently, the intervals approximated by interval containers are assumed to be equally distributed within the bounds of the interval containers. However, the interval patterns within the approximations depends on the used dataset, e.g. the dimensionality of the used dataset, and the used linearization method, i.e. the used space filling curve. Incorporating more pattern characteristics within the interval containers would be an enhancement for the computation of the intersection probability, and thus, maybe would improve the object decomposition and the corresponding query performance. However, more parameters yield more expensive object approximations. The number and the type of parameters which should additionally be taken to optimize the approximations is still an open question.

- Another interesting idea would be to extend the interval based representation of spatially extended objects with additional information concerning the object topology, e.g. the depth of penetration. This additional information could be used to determine the intensity of the

intersection between two objects. Instead of simply distinguishing only the object voxel from the free-space voxel, we can store the depth of object penetration for each object voxel. Consequently, each two- or three-dimensional object can be represented as a sequence of intersection depths. The main advantage of this kind of object representation would be that the complete spectrum of query and indexing techniques developed for time series, like dimensionality reduction and threshold based representation, can be used. This kind of spatial object representation especially seems promising for haptic query processing, since distance information achieves a better enhancement of the haptic rendering methods than simple collision information.

For the similarity search on time series data, future research could be guided in the following directions:

- Analogue to the object approximation approaches proposed for complex spatial data, the threshold-crossing time intervals of a time series can also be approximated by means of interval containers. That means, that close time intervals can be suitably grouped into larger approximations in a compressed form in order to apply efficient filter steps during the query process. Following this idea, we have to develop adequate approximations for the trapezoid segments in the parameter space.

- Currently, we assume that for some datasets and some applications there is exactly one amplitude and one threshold value respectively which has to be taken into consideration for similarity search. However, investigating threshold similarity analysis with multiple thresholds at the same time could yield an potential improve of data mining tasks.

- Last but not least, it is interesting to generalize the concept of threshold queries to threshold-range queries. Instead of taking one certain threshold value into account, threshold-range queries allows us to focus the similarity search on one or more amplitude ranges. An open question is how to define similarity measures for threshold ranges and how to process threshold-queries efficiently.

List of Figures

List of Tables

References

[ABB03] O. Alter, P. Brown, and D. Botstein. "Generalized Singular Value Decomposition for Comparative Analysis of Genome-Scale Expression Data Sets of two Different Organisms". In *Proc. Natl. Aca. Sci. USA*, volume 100, pages 3351–3356, 2003.

[ABKS99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. "OPTICS: Ordering Points to Identify the Clustering Structure". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'99), Philadelphia, PA*, pages 49–60, 1999.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. "Efficient Similarity Search in Sequence Databases". In *Proc. 4th Int. Conf. on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[AKK+06a] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. "Semi-Supervised Threshold Queries on Pharmacogenomics Time Sequences". In *Proc. 4th Asia Pacific Bioinformatics Conference (APBC 2006), Taipei, Taiwan*, pages 307–316, 2006.

[AKK+06b] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. "Similarity Search on Time Series based on Threshold Queries". In *Proc. 10th Int. Conf. on Extending Database Technology (EDBT 2006), Munich, Germany*, pages 276–294, 2006.

[AKK+06c] J. Aßfalg, H.-P. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. "Threshold Similarity Queries in Large Time Series Databases". In *Proc. 22nd Int. Conf. on Data Engineering (ICDE 2006), Atlanta (GA), U.S.A.*, page 149, 2006.

[All83] J. Allen. "Maintaining knowledge about temporal intervals". In *Commun. ACM 26, 11*, pages 832–843, 1983.

[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.

[AMW97] H. K. Ahn, N. Mamoulis, and H. M. Wong. "A survey on multidimensional access methods". In *Lecture COMP630c,"Spatial, Image and Multimedia Databases", University of Science and Technology, Clearwater Bay, Hong Kong*, 1997.

[APR+98] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. "Scalable Sweeping-Based Spatial Join". In *Proc. 24th Int. Conf. on Very Large Databases (VLDB'98), New York, NY*, pages 570–581, 1998.

[AT95] C.-H. Ang and K.-P. Tan. "The Interval B-Tree". In *Information Processing Letters 53(2)*, pages 85–89, 1995.

[AV96] L. Arge and J. S. Vitter. "Optimal Dynamic Interval Management in External Memory". In *Proc. 37th Annual Symp. on Foundations of Computer Science*, pages 560–569, 1996.

[BBM04a] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'04), Seattle, WA*, pages 59–68, 2004.

[BBM04b] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proc. 21st Int. Conf. on Machine Learning (ICML 2004), Banff, Alberta, Canada*, 2004.

[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)*, pages 229–248, 1994.

[Bel61] R. Bellman. "Adaptive Control Processes: A Guided Tour". In *Princeton University Press*, 1961.

[BG94] G. Blankenagel and R. H. Güting. "External Segment Trees". In *Algorithmica, 12(6)*, pages 498–532, 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 *3rd Int. Symp. on Large Spatial Databases (SSD'93)*, pages 357–376, 1993.

[BKK99] C. Böhm, G. Klump, and H.-P. Kriegel. "XZ-Ordering: A Space-Filling Curve for Objects with Spatial Extension". In *Proc. 6th Int. Symp. on Large Spatial Databases, LNCS 1651*, pages 75–90, 1999.

[BKP98] S. Berchtold, H.-P. Kriegel, and M. Pötke. "Database Support for Concurrent Digital Mock-Up". In *Proc. IFIP Int. Conf. PROLAMAT, Globalization of Manufacturing in the Digital Communications Era of the 21st Century*, pages 499–509, 1998.

[BKS93a] 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. 9th Int. Conf. on Data Engineering (ICDE'93), Vienna, Austria*, pages 40–49, 1993.

[BKS93b] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. "Efficient Processing of Spatial Joins Using R-trees". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'93), Washington, D.C.*, pages 237–246, 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. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'90), Atlantic City, NJ*, pages 322–331, 1990.

[BKSS94] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger. "Multi-Step Processing of Spatial Joins". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94), Minneapolis, MN*, pages 197–208, 1994.

[BLMB02] R. Bellazzi, C. Larizza, P. Magni, and R. Bellazzi. "Quality assessment of dialysis services through intelligent data analysis and temporal data mining". In *Proc. ECAI'02 Workshop on Knowledge Discovery from (Spatio-) Temporal Data, Lyon, France*, pages 3–9, 2002.

[BM03] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'03), Washington, D.C.*, pages 39–48, 2003.

[BW94] M. Burrows and D. J. Wheeler. "A Block-sorting Lossless Data Compression Algorithm". In *Digital Systems Research Center Research Report 124*, 1994.

[BÖ98] T. Bozkaya and Z. M. Özsoyoglu. "Indexing Valid Time Intervals". In *Proc. 9th Int. Conf. on Database and Expert Systems Applications, LNCS 1460*, pages 541–550, 1998.

[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. 15th Int. Conf. on Data Engineering (ICDE'99), Sydney, Australia*, 1999.

[Cha03] C. Chatfield. *"The Analysis of Time Series"*. Chapman and Hall, 6^th edition, 2003.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *"Introduction to Algorithms"*. MIT-Press, Cambridge, MA, 1990.

[CN04] Y. Cai and R. Ng. "Index Spatio-Temporal Trajectories with Chebyshev Polynomials". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'04), Paris, France*, pages 599–610, 2004.

[Coh88] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Ass., New Jersey, 1988.

[Dat99] C. J. Date. *"An Introduction to Database Systems"*. Addison Wesley Longman, Boston, MA, 1999.

[DBE99] A. Demiriz, K. P. Bennett, and M. J. Embrechts. Semi-supervised clustering using genetic algorithms. In *Intelligent Engineering Systems Through Artificial Neural Networks 9*, pages 809–814, 1999.

[DeM97] M. DeMers. *"Fundamentals of Geographic Information Systems"*. J. Wiley & Sons, New York, 1997.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In

Journal of the Royal Statistical Society, Series B, volume 39(1), pages 1–38, 1977.

[Ede80] H. Edelsbrunner. *"Dynamic Rectangle Intersection Searching"*. Institute for Information Processing Report 47, Technical University of Graz, Austria, 1980.

[EHS04] J. Enderle, M. Hampel, and T. Seidl. "Joining Interval Data in Relational Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'04), Paris, France*, pages 683–694, 2004.

[EJS98] O. Etzion, S. Jajodia, and S. Sripada. *"Temporal Databases: Research and Practice"*. Lecture Notes in Computer Science, Vol. 1399, Springer-Verlag, Berlin, 1998.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96), Portland, OR*, pages 291–316, 1996.

[EM97] T. Eiter and H. Mannila. "Distance Measure for Point Sets and Their Computation". In *Acta Informatica, 34*, pages 103–133, 1997.

[ES93] M. Egenhofer and J. Sharma. "Topological Relations Between Regions in R^2 and Z^2". In *Proc. 3rd Int. Symp. on Large Spatial Databases (SSD'93), Singapore, LNCS 692*, pages 316–336, 1993.

[EWK90] R. Elmasri, G. T. J. Wuu, and Y.-J. Kim. "The Time Index: An Access Structure for Temporal Data". In *Proc. 16th Int. Conf. on Very Large Databases (VLDB'90), Brisbane, Australia*, pages 1–12, 1990.

[EZZ04] C. F. Eick, N. M. Zeidat, and Z. Zhao. Supervised clustering - algorithms and benefits. In *16th IEEE Int. Conf. on Tools with Artificial Intelligence (ICTAI 2004), Boca Raton, FL*, pages 774–776, 2004.

[FFS00] J.-C. Freytag, M. Flasza, and M. Stillger. "Implementing Geospatial Operations in an Object-Relational Database System". In *Proc. 12th Int. Conf. on Scientific and Statistical*

Database Management (SSDBM 2000), Berlin, Germany, pages 209–219, 2000.

[FJM97] C. Faloutsos, H. V. Jagadish, and Y. Manolopoulos. "Analysis of the n-Dimensional Quadtree Decomposition for Arbitrary Hyperrectangles". In *IEEE Trans. on Knowledge and Data Engineering 9(3)*, pages 373–383, 1997.

[FL95] C. Faloutsos and K.-I. Lin. "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'95), San Jose, CA*, pages 163–174, 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 *Proc. ACM Symp. on Principles of Database Systems (PODS'89), Philadephia, PA*, pages 247–252, 1989.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. "Fast Subsequence Matching in Time-series Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94), Minneapolis, MN*, pages 419–429, 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 *Proc. 4th Int. Symp. on Large Spatial Databases (SSD'95), LNCS 951, Portland, ME*, pages 96–116, 1995.

[GFKS02] T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola. "Multi-Instance Kernels". In *Proc. 19th Int. Conf. on Machine Learning (ICML 2004), Sydney, Australia*, pages 179–186, 2002.

[GG97] V. Gaede and O. Günther. *"Survey on Multidimensional Access Method".* Technical Report ISS-16, Department of Economics and Business Administration, Humboldt University Berlin, revised version, 1997.

[GG98] V. Gaede and O. Günther. "Multidimensional Access Methods". In *ACM Computing Surveys*, volume 30(2), pages 170–231, 1998.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. "A Hierarchical Structure for Rapid Interference Detection". In *Proc. ACM SIGGRAPH Int. Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH'96), New Orleans, LA*, pages 171–180, 1996.

[GLOT96] C. H. Goh, H. Lu, B. C. Ooi, and K.-L. Tan. "Indexing Temporal Data Using Existing B+-Trees". In *Data & Knowledge Engineering, Elsevier, 18(2)*, pages 147–165, 1996.

[GU99] G. Guimarães and A. Ultsch. "A method for temporal knowledge conversion". In *Proc. 3rd Int. Symp. on Intelligent Data Analysis, Amsterdam, The Netherlands*, pages 369–380, 1999.

[Gut84] A. Guttman. "R-Trees: A Dynamic Index Structure for Spatial Searching". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'84), Boston, MA*, pages 47–57, 1984.

[Güt94] R. H. Güting. "An introduction to spatial database systems". In *VLDB Journal*, volume 3(4), pages 357–400, 1994.

[HBV01] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. "On Clustering Validation Techniques". In *Journal of Intelligent Information Systems*, volume 17(2-3), pages 107–145, 2001.

[HJ96] E. Hanson and T. Johnson. "Selection Predicate Indexing for Active Databases Using Interval Skip Lists". In *Information Systems, 21(3)*, pages 269–298, 1996.

[HJR97] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. "A Cost Model for Estimating the Performance of Spatial Joins Using R-trees". In *Proc. 9th Int. Conf. on Scientific and Statistical Database Management (SSDBM'97), Olympia, WA*, pages 30–38, 1997.

[HK01] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Academic Press, 2001.

[HS95] G. Hjaltason and H. Samet. "Ranking in Spatial Databases". In *Proc. Int. Symp. on Large Spatial Databases (SSD'95), Portland, OR*, pages 83–95, 1995.

[Höp01] F. Höppner. "Discovery of temporal patterns – learning rules about the qualitative behavior of time series". In *Proc. of the 5th European Conf. on Principles and Practice of Knowledge Discovery in Databases, Freiburg, Germany*, pages 192–203, 2001.

[Jag90] H. V. Jagadish. "Linear Clustering of Objects with Multiple Attributes". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'90), Atlantic City, NJ*, pages 332–342, 1990.

[JD88] A. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flyn. "Data Clustering: A Review. ACM Computing Surveys". In *ACM Computing Surveys, 31(3)*, pages 264–323, 1999.

[Joh06] T. K. Johnson. *"A reformulation of Coombs' Theory of Unidimensional Unfolding by representing attitudes as intervals"*. Doctoral thesis, University of Sydney, Psychology, 2006.

[JS99] C. S. Jensen and R. T. Snodgrass. "Temporal Data Management". In *Proc. IEEE Trans. on Knowledge and Data Engineering (TKDE'99)*, pages 36–44, 1999.

[Kau87] A. Kaufman. "An Algorithm for 3D Scan-Conversion of Polygons". In *Proc. Eurographics, Portland, OR*, 1987.

[KBS93] H.-P. Kriegel, T. Brinkhoff, and R. Schneider. "Efficient Spatial Query Processing in Geographic Database Systems". In *IEEE Data Engineering Bulletin*, volume 16(3), pages 10–15, 1993.

[KC00] H. Kargupta and P. Chan. "Advances in Distributed and Parallel Knowledge Discovery". In *AAAI/MIT Press*, 2000.

[KCMP01] E. Keogh, K. Chakrabati, S. Mehrotra, and M. Pazzani. "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'01), Santa Barbara, CA*, pages 151–162, 2001.

[KCPM01] E. Keogh, K. Chakrabati, M. Pazzani, and S. Mehrotra. "Dimensionality reduction for fast similarity search in large time series databases". In *Knowledge and Information Systems 3(3)*, pages 263–286, 2001.

[KF00] P.-S. Kam and A. W.-C. Fu. "Discovering temporal patterns for interval-based events". In *Proc. of Int. Conf. on Data Warehousing and Knowl. Discovery, LNCS 1874*, pages 317–326, 2000.

[KF02] E. Keogh and T. Folias. "The UCR Time Series Data Mining Archive". In *http://www.cs.ucr.edu/∼eamonn/TSDMA/index.html*, 2002.

[KHM+98] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs". In *IEEE Trans. on Visualization and Computer Graphics, 4(1)*, pages 21–36, 1998.

[KJF97] F. Korn, H. Jagadish, and C. Faloutsos. "Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'97), Tucson, AZ*, pages 289–300, 1997.

[KKM02] D. Klein, S. D. Kamvar, and C. Manning. "From Instance-level Constraints to Space-Level Constraints: Making the Most of Prior Knowledge in Data Clustering". In *Proc. 19th Int. Conf. on Machine Learning (ICML 2000), Sydney, Australia*, pages 307–314, 2002.

[KKPR04a] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. "Effective Decompositioning of Complex Spatial Objects into Intervals". In *Proc. IASTED Int. Conf. on Databases and Applications (DBA 2004), Innsbruck, Austria*, 2004.

[KKPR04b] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. "Spatial Join for High-Resolution Objects". In *Proc. 16th Int. Conf. on Scientific and Statistical Database Management (SSDBM 2004), Santorini Island, Greece*, pages 151–160, 2004.

[KKPR04c] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. "Statistic Driven Acceleration of Object-Relational Spatial Index Structures". In *Proc. 9th Int. Conf. on Database Systems for Advanced Applications (DASFAA 2004), Jeju Island, Korea*, pages 169–183, 2004.

[KKPR05a] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. "Database Support for Haptic Exploration in Very Large Virtual Environments". In *Proc. 11th Int. MultiMedia Modelling Conference (MMM 2005), Melbourne, Australia*, pages 352–357, 2005.

[KKPR05b] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. "Distributed Intersection Join of Complex Interval Sequences". In *Proc. 10th Int. Conf. on Database Systems for Advanced Applications (DASFAA 2005), Beijing, China*, pages 748–760, 2005.

[KKPR05c] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz. "Efficient Join Processing for Complex Rasterized Objects". In *Proc. 7th Int. Conf. on Enterprise Information Systems (ICEIS 2005), Miami, FL*, pages 20–30, 2005.

[KP99a] E. Keogh and M. Pazzani. "Relevance Feedback Retrieval of Time Series Data". In *Research and Development in Information Retrieval*, pages 183–190, 1999.

[KP99b] E. Keogh and M. Pazzani. "Scaling up Dynamic Time Warping to Massive Datasets". In *3rd European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'99), Prague, Czech Republic*, pages 1–11, 1999.

[KP01] E. J. Keogh and M. J. Pazzani. "Derivative Dynamic Time Warping". In *Proc. 1st SIAM Int. Conf. on Data Mining (SDM'01), Chicago, IL*, 2001.

[KPPS02] H.-P. Kriegel, M. Pfeifle, M. Pötke, and T. Seidl. "A Cost Model for Interval Intersection Queries on RI-Trees". In *Proc. 14th Int. Conf. on Scientific and Statistical Database Management (SSDBM 2002), Edinburgh, Scotland*, pages 131–141, 2002.

[KPPS03a] H.-P. Kriegel, M. Pfeifle, M. Pötke, and T. Seidl. "Spatial Query Processing for High Resolutions". In *Proc. 8th Int. Conf. on Database Systems for Advanced Applications (DASFAA'03), Kyoto, Japan*, pages 17–26, 2003.

[KPPS03b] H.-P. Kriegel, M. Pfeifle, M. Pötke, and T. Seidl. "The Paradigm of Relational Indexing: A Survey". In *10th GI-Conf. on Database Systems for Business, Technology, and the Web (BTW 2003), Leipzig, Germany. GI-Edition Lecture Notes in Informatics, P-26*, pages 285–304, 2003.

[KPS00a] H.-P. Kriegel, M. Pötke, and T. Seidl. "Managing Intervals Efficiently in Object-Relational Databases". In *Proc. 26th Int. Conf. on Very Large Databases (VLDB'00), Cairo, Egypt*, pages 407–418, 2000.

[KPS00b] H.-P. Kriegel, M. Pötke, and T. Seidl. *"Relational Interval Tree"*. European Patent Office (EPO), Patent Application No. 00112031.0, 2000.

[KPS01] H.-P. Kriegel, M. Pötke, and T. Seidl. "Interval Sequences: An Object-Relational Approach to Manage Spatial Data". In *Proc. 7th Int. Symp. on Spatial and Temporal Databases (SSTD 2001), Los Angeles, CA, LNCS 2121*, pages 481–501, 2001.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[KS91] C. P. Kolovson and M. Stonebraker. "Segment Indexes: Dynamic Indexing Techniques for Multi-Dimensional Interval Data". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'91), Denver, CO*, pages 138–147, 1991.

[KSF+96] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. "Fast Nearest Neighbor Search in Medical Image Databases". In *Proc. 22nd Int. Conf. on Very Large Databases (VLDB'96), Mumbai (Bombay), India*, pages 215–226, 1996.

[Kun02] P. Kunath. "Compression of CAD data". In *Diploma Thesis, University of Munich*, 2002.

[LG98] M. C. Lin and S. Gottschalk. "Collision detection between geometric models: a Survey". In *Proc. IMA Conf. on Mathematics of Surfaces*, page 20, 1998.

[LM86] P. C. Lockemann and H. C. Mayr. *"Information System Design: Techniques and Software Support"*. Springer, Amsterdam, The Netherlands, 1986.

[LR96] M.-L. Lo and C. V. Ravishankar. "Spatial Hash-Joins". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'96), Montreal, Canada*, pages 247–258, 1996.

[LSW99] C. Lennerz, E. Schömer, and T. Warken. "A Framework for Collision Detection and Response". In *Proc. 11th European Simulation Symposium (ESS)*, pages 309–314, 1999.

[LT98] C. Lee and T.-M. Tseng. "Temporal Grid File: A File Structure for Interval Data". In *Data & Knowledge Engineering, Elsevier, 26(1)*, pages 71–97, 1998.

[LZ77] A. Lempel and J. Ziv. "A Universal Algorithm for Sequential Data Compression". In *IEEE Transactions on Information Theory, Vol. IT-23, No. 3*, pages 337–343, 1977.

[McQ67] J. McQueen. "Some Methods for Classification and Analysis of Multivariate Observations". In *5th Berkeley Symp. Math. Statist. Prob.*, volume 1, pages 281–297, 1967.

[MH99] T. Möller and E. Haines. "Real-Time Rendering". In *A K Peters, 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". In *Tech. Rep. CS-TR-3611, University of Maryland*, 1996.

[MP94] C. B. Medeiros and F. Pires. "Databases for GIS". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94), Minneapolis, MN*, pages 107–115, 1994.

[MPT99] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling". In *Proc. ACM SIGGRAPH Int. Conf. on Computer Graphics and Interactive Techniques*, pages 401–408, 1999.

[MPT06] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. "Voxel-Based 6-DOF Haptic Rendering Improvements". In *Haptics-e (http://www.haptics-e.org)*, volume 3(7), 2006.

[MTT00] Y. Manolopoulos, Y. Theodoridis, and V. J. Tsotras. *"Advanced Database Indexing"*. Kluwer, Boston, MA, 2000.

[NH94] R. Ng and J. Han. "Efficient and Effective Clustering Methods for Spatial Data Mining". In *Proc. 20th Int. Conf. on Very Large Databases (VLDB'94), Santiago, Chile*, pages 144–155, 1994.

[Ore89] J. A. Orenstein. "Redundancy in Spatial Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'89), Portland, OR*, pages 294–305, 1989.

[PD96] J. M. Patel and D. J. DeWitt. "Partition Based Spatial-Merge Join". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'96), Montreal, Canada*, pages 259–270, 1996.

[PRS99] A.N. Papadopoulos, P. Rigaux, and M. Scholl. "A Performance Evaluation of Spatial Join Processing Strategies". In *Proc. Symp. on Large Spatial Databases (SSD'99), Hong Kong, China*, pages 286–307, 1999.

[PS93] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction. 5th ed.* Springer, 1993.

[Pöt01] M. Pötke. *"Spatial Indexing for Object-Relational Databases"*. Doctoral thesis, University of Munich, 2001.

[Ram97] S. Ramaswamy. "Efficient Indexing for Constraint and Temporal Databases". In *Proc. 6th Int. Conf. on Database Theory (ICDT'97), Delphi, Greece*, pages 419–431, 1997.

[Ren00] M. Renz. "Dynamic Collision Detection in Virtual Environments". In *Advanced Term Project, University of Munich, German Aerospace Center (DLR), Oberpfaffenhofen*, 2000.

[Ren02] M. Renz. "Real-Time Support of Haptic Simulations Using Very Large Data Bases". In *Diploma Thesis, University of Munich*, 2002.

[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*, pages 771–777, 2005.

[Rov02] D. Roverso. "Plant diagnostics by transient classification: The aladdin approach". In *IJIS,Special Issue on Intelligent Systems for Plant Surveillance and Diagnostics*, volume 17, pages 767–790, 2002.

[RPP+01] M. Renz, C. Preusche, M. Pötke, H.-P. Kriegel, and G. Hirzinger. "Stable Haptic Interaction with Virtual Environments Using an Adapted Voxmap-PointShell^{TM} Algorithm.". In *Proc. Int. Conf. Eurohaptics 2001, Birmingham, UK*, pages 149–154, 2001.

[Sai94] N. Saito. "Local feature extraction and its application using a library of bases". In *PhD thesis, Yale University, December*, 1994.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison Wesley Longman, Boston, 1990.

[SCSS05] A. Schliep, I. G. Costa, C. Steinhoff, and A. Schonhuth. "Analyzing Gene Expression Time-Courses". In *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, volume 2(3), pages 179–193, 2005.

[Sew06] J. Seward. "The bzip2 and libbzip2 official home page". In *http://sources.redhat.com/bzip2*, 2006.

[SFGM93] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. "The SEQUOIA 2000 Sorage Benchmark". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'93), Washington, D.C.*, pages 2–11, 1993.

[Sib73] R. Sibson. "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method". In *The Computer Journal*, volume 16(1), pages 30–34, 1973.

[SK93] M. Schiwietz and H.-P. Kriegel. "Query Processing of Spatial Objects: Complexity versus Redundancy". In *Proc. 3rd Int. Symp. on Large Spatial Databases (SSD'93), Singapore, LNCS 692*, pages 377–396, 1993.

[SK98] T. Seidl and H.-P. Kriegel. "Optimal Multi-Step k-Nearest Neighbor Search". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'98), Seattle, WA*, pages 154–165, 1998.

[SLM93] B. Seeger, P. Larson, and R. McFadyen. "Reading a Set of Disk Pages". In *Proc. 19th Int. Conf. on Very Large Databases (VLDB'93), Dublin, Ireland*, pages 592–603, 1993.

[SOL94] H. Shen, B. C. Ooi, and H. Lu. "The TP-Index: A Dynamic and Efficient Indexing Mechanism for Temporal Databases". In *Proc. 10th Int. Conf. on Data Engineering (ICDE'94), Houston, TX*, pages 274–281, 1994.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects". In *Proc. 13th Int. Conf. on Very Large Databases (VLDB'87), Brighton, England*, pages 507–518, 1987.

[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.". In *Molecular Biolology of the Cell*, volume 9, pages 3273–3297, 1998.

[SW88] H. Six and P. Widmayer. "Spatial searching in geometric databases". In *Proc. 4th IEEE Int. Conf. On Data Engineering (ICDE'88), Los Angeles, CA*, pages 496–503, 1988.

[TCG+93] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass. *"Temporal Databases: Theory, Design and Implementation"*. Benjamin-Cummings, 1993.

[VHTM99] R. Villafane, K. A. Hua, D. Tran, and B. Maulik. "Mining Interval Time Series". In *Proc. on Data Warehousing and Knowledge Discovery*, pages 318–330, 1999.

[VLB05] J. R. Viqueria, N. Lorentzos, and N. Brisaboa. *"Survey on Spatial Data Modelling Approaches"*. Idea Group 2005, 2005.

[WCRS01] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. "Constrained K-means Clustering with Background Knowledge". In *Proc. Int. Conf. on Machine Learning (ICML 2001), Williamstown, MA*, pages 577–584, 2001.

[WFS04] S. Wichert, K. Fokianos, and K. Strimmer. "Identifying Periodically Expressed Transcripts in Microarray Time Series Data". In *Bioinformatics*, volume 20(1), pages 5–20, 2004.

[XEKS98] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander. "A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases". In *Proc. 14th Int. Conf. on Data Engineering (ICDE'98), Orlando, FL*, pages 324–331, 1998.

[YF00] B. K. Yi and C. Faloutsos. "Fast Time Sequence Indexing for Arbitrary Lp Norms". In *Proc. 26th Int. Conf. on Very Large Databases (VLDB'00), Cairo, Egypt*, pages 385–394, 2000.

[Zho05] S. Zhong. "Semi-Supervised Sequence Classification With Hmms". In *IJPRAI*, volume 19(2), pages 165–182, 2005.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'96), Montreal, Canada*, pages 103–114, 1996.

[ÖV99] T. Özsu and P. Valduriez. "Principles of Distributed Database Systems". In *Prentice Hall, ISBN 0-13-659707-6*, 1999.

Curriculum Vitae

Matthias Alexander Renz was born on September 6, 1971 in Munich, Germany. He attended primary school and high-school until 1992.

From 1992 to 1997, he studied Electrical Engineering at the University of Applied Sciences in Munich. During this time he worked as a student assistant at the Maurer Electronics GmbH, Munich, a company which is specialized in laser technology systems, culminating in his diploma thesis on "Development of a computer-controlled illumination-system" (written in German).

In 1997 he entered the University of Munich (LMU), studying Computer Science with a minor in Computer linguistics. His diploma thesis was on "Real-Time Support of Haptic Simulations Using Very Large Databases", supervised by Prof. Dr. Hans-Peter Kriegel, Dr. Marco Pötke and Prof. Dr. Thomas Seidl from the University of Munich and co-supervised by Dr. Gerhard Grunwald and Dr. Max Fischer from the German Aerospace Center. During this time he worked as a student assistant for the German Aerospace Center, Institute of Robotics and Mechatronics, in Oberpfaffenhofen, Germany for the MORPHA project and also for the University of Munich.

In 2002, he started working at the University of Munich as a research and teaching assistant in the group of Prof. Dr. Hans-Peter Kriegel, the chair of the teaching and research unit for database and information systems at the Institute of Computer Science. The research interests of Matthias Renz include spatial data management and knowledge discovery in large spatial, temporal and multimedia databases. Recently he received the "Best Paper Award" at the International Conference on Database Systems for Advanced Applications (DASFAA 2006) in Singapore.