

Spatial Database Support for Virtual Engineering

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

von
Martin Pfeifle

Tag der Einreichung: 21. April 2004
Tag der mündlichen Prüfung: 15. Oktober 2004

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

Acknowledgments

Many people supported and encouraged me in the past three years while I was working on this dissertation. I would like to thank them here, even if I cannot mention them all by name.

I would like to express my deepest thanks to my supervisor Prof. Dr. Hans-Peter Kriegel. Without his confidence in me and my ideas, and without the productive and inspiring working atmosphere he created, this work could never have come into existence. I am also very grateful to Prof. Dr. Bernhard Seeger for his interest in my work and his immediate willingness to act as a second referee.

This work would not have been possible without the cooperation of my colleagues in the database group. In particular, I would like to thank Dr. Marco Pötke and Prof. Dr. Thomas Seidl who introduced me into the area of spatial databases and guided my first research efforts. Their suggestions and judgments have always been a rich source of inspiration. Furthermore, I would like to thank Matthias Renz, Peter Kunath, Stefan Brecheisen, Eshref Januzaj, Peer Kröger, Matthias Schubert, Karin Kailing and Stefan Schönauer for constructive and productive team-work.

I also appreciate the substantial help of the students whose study thesis or diploma thesis I supervised, including Stefan Brecheisen, Peter Kunath, Felix Leis, Olaf Schmitt, Maximillian Viermetz, Petra-Maria Strauß, Ralf Hofmann, Hans Maier, Wolfgang Mühlberger, Markus Veith, Marc Hiller and Michael Passer.

I want to express special thanks to Franz Krojer, for taking care of our technical environment, and to Susanne Grienberger, for bearing much of the administrative burdens.

Furthermore, I would like to thank Stefan Brecheisen and Peter Kunath for carefully reading significant portions of this work and for providing valuable hints on improving the presentation.

In particular, I would like to express my deepest thanks to my parents, who constantly supported me. They brought me up and taught me the readiness to work hard for my goals. I know that I would be nothing without them. Last but not least, my special thanks go to my wife Valerie for her encouragement and sacrificial love. Without her considerateness, this work could never have been accomplished. She took care of our two lovely little sons, Samuel and Benito, and answered their probing questions about the whereabouts of their father with a patient smile: “He is writing his dissertation in Munich”. Thank you!

Martin Pfeifle

Munich, April 2004.

Abstract

The development, design, manufacturing and maintenance of modern engineering products is a very expensive and complex task. Shorter product cycles and a greater diversity of models are becoming decisive competitive factors in the hard-fought automobile and plane market. In order to support engineers to create complex products when being pressed for time, systems are required which answer collision and similarity queries effectively and efficiently. In order to achieve industrial strength, the required specialized functionality has to be integrated into fully-fledged database systems, so that fundamental services of these systems can be fully reused, including transactions, concurrency control and recovery.

This thesis aims at the development of theoretical sound and practical realizable algorithms which effectively and efficiently detect colliding and similar complex spatial objects.

After a short introductory Part I, we look in Part II at different spatial index structures and discuss their integrability into object-relational database systems. Based on this discussion, we present two generic approaches for accelerating collision queries. The first approach exploits available statistical information in order to accelerate the query process. The second approach is based on a cost-based decomposition of complex spatial objects. In a broad experimental evaluation based on real-world test data sets, we demonstrate the usefulness of the presented techniques which allow interactive query response times even for large data sets of complex objects.

In Part III of the thesis, we discuss several similarity models for spatial objects. We show by means of a new evaluation method that data-partitioning similarity models yield more meaningful results than space-partitioning similarity models. We introduce a very effective similarity model which is based on a new paradigm in similarity

search, namely the use of vector set represented objects. In order to guarantee efficient query processing, suitable filters are introduced for accelerating similarity queries on complex spatial objects. Based on clustering and the introduced similarity models we present an industrial prototype which helps the user to navigate through massive data sets.

Abstract (in German)

Ein schneller und reibungsloser Entwicklungsprozess neuer Produkte ist ein wichtiger Faktor für den wirtschaftlichen Erfolg vieler Unternehmen insbesondere aus der Luft- und Raumfahrttechnik und der Automobilindustrie. Damit Ingenieure in immer kürzerer Zeit immer anspruchsvollere Produkte entwickeln können, werden effektive und effiziente Kollisions- und Ähnlichkeitsanfragen auf komplexen räumlichen Objekten benötigt. Um den hohen Anforderungen eines produktiven Einsatzes zu genügen, müssen entsprechend spezialisierte Zugriffsmethoden in vollwertige Datenbanksysteme integriert werden, so dass zentrale Datenbankdienste wie Transaktionen, kontrollierte Nebenläufigkeit und Wiederanlauf sichergestellt sind.

Ziel dieser Doktorarbeit ist es deshalb, effektive und effiziente Algorithmen für Kollisions- und Ähnlichkeitsanfragen auf komplexen räumlichen Objekten zu entwickeln und diese in kommerzielle Objekt-Relationale Datenbanksysteme zu integrieren.

Im ersten Teil der Arbeit werden verschiedene räumliche Indexstrukturen zur effizienten Bearbeitung von Kollisionsanfragen diskutiert und auf ihre Integrationsfähigkeit in Objekt-Relationale Datenbanksysteme hin untersucht. Daran anknüpfend werden zwei generische Verfahren zur Beschleunigung von Kollisionsanfragen vorgestellt. Das erste Verfahren benutzt statistische Informationen räumlicher Indexstrukturen, um eine gegebene Anfrage zu beschleunigen. Das zweite Verfahren beruht auf einer kostenbasierten Zerlegung komplexer räumlicher Datenbank-Objekte. Diese beiden Verfahren ergänzen sich gegenseitig und können unabhängig voneinander oder zusammen eingesetzt werden. In einer ausführlichen experimentellen Evaluation wird gezeigt, dass die beiden vorgestellten Verfahren interaktive Kollisionsanfragen auf umfangreichen Datenmengen und komplexen Objekten ermöglichen.

Im zweiten Teil der Arbeit werden verschiedene Ähnlichkeitsmodelle für räumliche Objekte vorgestellt. Es wird experimentell aufgezeigt, dass datenpartitionierende Modelle effektiver sind als raumpartitionierende Verfahren. Weiterhin werden geeignete Filtertechniken zur Beschleunigung des Anfrageprozesses entwickelt und experimentell untersucht. Basierend auf Clustering und den entwickelten Ähnlichkeitsmodellen wird ein industrietauglicher Prototyp vorgestellt, der Benutzern hilft, durch große Datenmengen zu navigieren.

Survey of Chapters

PART I. INTRODUCTION TO VIRTUAL ENGINEERING

1	Introduction	3
2	Spatial Engineering Databases	15

PART II. DATABASE SUPPORT FOR DIGITAL MOCKUP

3	Object Relational Indexing	29
4	A Cost Model for Spatial Intersection Queries	67
5	Statistic-Driven Acceleration of Relational Index Structures	95
6	Cost-based Decompositioning of Complex Spatial Objects	121

PART III. DATABASE SUPPORT FOR SIMILARITY SEARCH

7	Foundations of Similarity Search	171
8	Similarity Models for Voxelized CAD Data	199
9	Effectiveness of Similarity Models	223
10	Efficiency of Similarity Models	243
11	BOSS: Browsing Optics-Plots for Similarity Search	289
12	Conclusions	313

Table of Contents

Acknowledgments	i
Abstract	iii
Abstract (in German)	v
Survey of Chapters	vii
Table of Contents	ix

PART I. INTRODUCTION TO VIRTUAL ENGINEERING

1 Introduction	3
<i>1.1 Industrial Applications</i>	4
1.1.1 Digital Mockup	5
1.1.2 Similarity Search	6
1.1.3 Further Applications	7
<i>1.2 Outline of the Thesis</i>	10
1.2.1 Virtual Engineering (Part I)	11
1.2.2 Database Support for Digital Mockup (Part II)	11
1.2.3 Database Support for Similarity Search (Part III)	12
2 Spatial Engineering Databases	15
<i>2.1 Spatial Engineering Data</i>	16
2.1.1 Triangle Meshes	17
2.1.2 Voxelized Data	18
2.1.3 Feature Vectors	21

2.2	<i>Spatial Engineering Query Processing</i>	22
2.2.1	Effective Query Processing	22
2.2.2	Efficient Query Processing	23

PART II. DATABASE SUPPORT FOR DIGITAL MOCKUP

3	Object Relational Indexing	29
3.1	<i>Object Relational Databases</i>	30
3.1.1	Classification of Data Models	30
3.1.2	Abstract Data Types	32
3.2	<i>Extensible Query Language</i>	32
3.2.1	Extensible Indexing	32
3.2.2	Extensible Optimizing	34
3.3	<i>Implementation of Access Methods</i>	36
3.3.1	Integrating Approach	37
3.3.2	Generic Approach	41
3.3.3	Relational Approach	43
3.4	<i>Basics of Relational Access Methods</i>	44
3.4.1	Paradigms of Access Methods	44
3.4.2	Relational Storage of Index Data	46
3.5	<i>Operations on Relational Access Methods</i>	47
3.5.1	Cursor-Bound Operations	48
3.5.2	Cursor-Driven Operations	49
3.6	<i>Navigational Scheme of Index Tables</i>	51
3.6.1	RR-tree – An Example for the Navigational Scheme	52
3.7	<i>Direct Scheme of Index Tables</i>	55
3.7.1	RQ-tree – An Example for the Direct Scheme	55
3.7.2	RI-tree - Another Example for the Direct Scheme	57
3.8	<i>Industrial Application</i>	61
3.8.1	Spatial Data Management	62
3.8.2	Query Processing	62
3.8.3	Efficient Execution Plans	63

3.8.4	Experimental Evaluation of Query Processing	64
3.8.5	System Architecture	65
4	A Cost Model for Spatial Intersection Queries	67
4.1	<i>Introduction</i>	68
4.2	<i>Related Work</i>	70
4.2.1	Selectivity Estimation	70
4.2.2	Cost Estimation	71
4.3	<i>Selectivity Estimation</i>	71
4.3.1	Histogram-Based Selectivity Estimation	72
4.3.2	Quantile-Based Selectivity Estimation	73
4.4	<i>Model for I/O Cost</i>	76
4.5	<i>Extension to Spatial Interval Sequences</i>	80
4.5.1	Aggregates on Interval Sequences	81
4.5.2	Extended Selectivity Estimation	83
4.5.3	Extended I/O Cost Model	84
4.6	<i>Empirical Evaluation</i>	87
4.6.1	Experimental Setup	87
4.6.2	Computation of Statistics	88
4.6.3	Selectivity Estimation	89
4.6.4	Cost Estimation	91
4.7	<i>Summary</i>	93
5	Statistic-Driven Acceleration of Relational Index Structures	95
5.1	<i>Introduction</i>	96
5.2	<i>Acceleration of Relational Space-Partitioning Access Methods</i>	96
5.2.1	Statistics Related to the Relational Access Method	97
5.2.2	Statistics Related to the Built-in Index Structure	98
5.2.3	Statistics Related to the Object Decompositioning	104
5.3	<i>Acceleration of Relational Data-Partitioning Access Methods</i>	105
5.3.1	Relational Mapping of a Hierarchical Index Structure	106
5.3.2	General Idea	107
5.3.3	Adaption to the RR-tree	111

5.4	<i>Experimental Evaluation</i>	112
5.4.1	Space-Partitioning Index Structures	112
5.4.2	Data-Partitioning Index Structures	117
5.5	<i>Summary</i>	118
6	Cost-based Decompositioning of Complex Spatial Objects	121
6.1	<i>Introduction</i>	122
6.2	<i>Related Work</i>	123
6.3	<i>Decompositioning of High-Resolution Spatial Objects</i>	124
6.3.1	Gray Containers	125
6.3.2	Storing Gray Containers in an ORDBMS	128
6.3.3	Compression of Gray Containers	129
6.3.4	Grouping into Gray Containers	134
6.4	<i>Object Relational Query Processing</i>	138
6.4.1	The intersect SQL Statements	138
6.4.2	Optimizations	140
6.5	<i>Spatial Join</i>	143
6.5.1	Related Work	145
6.5.2	Cost Model	146
6.5.3	Decompositioning Algorithm	147
6.5.4	Join Algorithms	149
6.6	<i>Experimental Evaluation</i>	154
6.6.1	Storage Requirements	155
6.6.2	Update Operations	156
6.6.3	Collision Queries based on the MaxGap-Grouping	157
6.6.4	Collision Queries based on the GroupCon-Grouping	160
6.6.5	Spatial Join Processing	162

6.7 Conclusion.....	166
---------------------	-----

PART III. DATABASE SUPPORT FOR SIMILARITY SEARCH

7 Foundations of Similarity Search	171
7.1 <i>Similarity Query Types</i>	172
7.1.1 Similarity Range Queries.....	172
7.1.2 Similarity k-nn Queries	173
7.1.3 Similarity Ranking Queries	174
7.1.4 Further Similarity Queries	175
7.2 <i>Similarity Queries within Object-Relational Database Systems</i>	176
7.2.1 Integration of Range Queries.....	176
7.2.2 Integration of k-nn Queries	176
7.2.3 Integration of Ranking Queries	178
7.3 <i>Access Methods for Similarity Search</i>	178
7.3.1 Multi-Dimensional Access Methods	178
7.3.2 One-Dimensional Access Methods	180
7.3.3 Scan-Based Access Methods	182
7.3.4 Metric Access Methods	183
7.3.5 Miscellaneous Access Methods.....	184
7.4 <i>One-Step Similarity Query Processing</i>	185
7.4.1 Index based Range Queries	186
7.4.2 Index based k-nn Queries.....	186
7.4.3 Index based Ranking Query.....	187
7.5 <i>Multi-Step Similarity Query Processing</i>	188
7.5.1 The Lower-Bounding Property	189
7.5.2 Multi-Step Range Queries	189
7.5.3 Multi-Step k-nn Queries	190
7.5.4 Multi-Step Ranking Queries	191
7.6 <i>Similarity Models</i>	192
7.6.1 Feature-Based Similarity Search	193
7.6.2 Geometry-Based Similarity Search	196
7.6.3 Summary	198

8	Similarity Models for Voxelized CAD Data	199
8.1	<i>Object Similarity</i>	200
8.1.1	Normalization of CAD Data	201
8.2	<i>Space Partitioning Similarity Models</i>	204
8.2.1	Shape Histograms for Voxelized CAD Data	204
8.2.2	The Volume Model.	205
8.2.3	The Solid-Angle Model	206
8.2.4	The Eigen-Value Model	207
8.3	<i>Data Partitioning Similarity Models</i>	209
8.3.1	The Cover Sequence Model.	209
8.3.2	The Vector Set Model	213
9	Effectiveness of Similarity Models	223
9.1	<i>A New Approach for Evaluating Similarity Models</i>	224
9.1.1	k-nn Queries.	224
9.1.2	OPTICS: A Density-Based Hierarchical Clustering Algorithm	225
9.2	<i>Experimental Evaluation</i>	233
9.2.1	Experimental Setup	233
9.2.2	Evaluation of the Space Partitioning Similarity Models	234
9.2.3	Evaluation of the Data Partitioning Similarity Models	237
9.2.4	Summary	240
10	Efficiency of Similarity Models	243
10.1	<i>The NB-Tree: A Filter for the Single Vector Models</i>	244
10.2	<i>Filters for the Minimal Matching Distance</i>	246
10.2.1	The Centroid Approach	247
10.2.2	The Euclidean Norm Approach	249
10.2.3	The Closest Pair Approach	252
10.2.4	Combined Filters	254
10.2.5	Conclusion	254
10.3	<i>The Optimized Relational M-Tree</i>	255
10.3.1	The M-tree	256
10.3.2	The Relational M-tree	259

10.3.3 The Scanning M-Tree	260
10.3.4 The Filtering M-tree.	263
10.3.5 The Caching M-tree	268
<i>10.4 Experimental Evaluation.</i>	<i>269</i>
10.4.1 Single Vector Models: The NB-Tree	270
10.4.2 Vector Set Model: Filters for the Minimal Matching Distance . .	276
10.4.3 Vector Set Model: M-tree.	282
<i>10.5 Summary</i>	<i>287</i>
11 BOSS: Browsing Optics-Plots for Similarity Search	289
<i>11.1 Introduction.</i>	<i>290</i>
<i>11.2 Hierarchical Clustering.</i>	<i>291</i>
11.2.1 Major Advantages of OPTICS	291
11.2.2 Application Ranges	292
<i>11.3 Cluster Recognition.</i>	<i>295</i>
11.3.1 Recent Work.	296
11.3.2 Gradient Clustering	298
<i>11.4 Cluster Representatives.</i>	<i>301</i>
11.4.1 The Extended Medoid Approach	303
11.4.2 The Minimum Core-Distance Approach	303
11.4.3 The Maximum Successor Approach	304
<i>11.5 System Architecture</i>	<i>306</i>
<i>11.6 Evaluation</i>	<i>307</i>
11.6.1 Cluster Recognition	308
11.6.2 Cluster Representation	310
11.6.3 Summary	310
<i>11.7 Summary</i>	<i>311</i>
12 Conclusions	313
<i>12.1 Summary and Contributions</i>	<i>314</i>
12.1.1 Virtual Engineering (Part I)	314
12.1.2 Database Support for Digital Mockup (Part II)	314
12.1.3 Database Support for Similarity Search (Part III)	315

<i>12.2 Potentials for Future Work</i>	317
12.2.1 Efficient Haptic Simulation	317
12.2.2 A Similarity Model based on Patterns	318
12.2.3 Efficient Density-Based Clustering	318
12.2.4 Navigating through Massive Multimedia Data Sets	319
12.2.5 Browsing Distributed Data Sets	320
List of Figures	321
List of Definitions	327
References	329
Curriculum Vitae	345

Part I

Introduction to Virtual Engineering



Chapter 1

Introduction

In the automotive and aerospace industry, millions of technical documents are generated during the development of complex engineering products. Particularly, the universal application of Computer Aided Design (CAD) from the very first design to the final product created the need for transactional, concurrent, reliable, and secure data management. The huge underlying CAD databases, occupying terabytes of distributed secondary and tertiary storage, are typically stored and referenced in Engineering Data Management (EDM) systems and organized by means of hierarchical product structures. Existing EDM systems are based on fully-fledged object-relational database servers. They organize and synchronize concurrent access to the CAD data of an engineering product. The distributed CAD files are linked to global product structures which allow a hierarchical view on the many possible product configurations emerging from the various versions and variants created for each component.

Although most CAD files represent spatial objects or contain spatially related data, existing EDM systems do not efficiently support the evaluation of spatial predicates, e.g. collision queries and similarity queries. 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 1). 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 in order to “find all parts

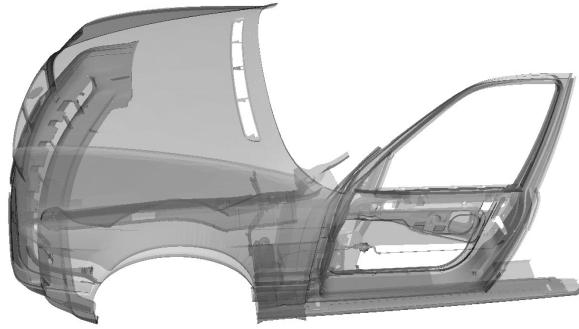


Figure 1: Partial virtual prototype of a car.

intersecting a specific query volume”. Furthermore, the system should support queries as for instance “find the five parts which are most similar to a specific hydraulic hose”. Unfortunately, the inclusion of these spatial predicates is not supported efficiently by common, structure-related EDM systems.

Although the EDM system maintains a consistent knowledge about the storage location of each native CAD file, only a few spatial properties, including the position and bounding box of the respective part in the product space, are immediately accessible. But many applications of virtual engineering require a more fine-grained spatial selection. Thus the EDM system has to be extended by suitable representations of the product geometry.

In this thesis, we introduce spatial database technology into the file-based world of CAD. As we integrate 3D spatial data management into standard object-relational database systems, the required support for data independence, transactions, recovery, and interoperability can be achieved.

1.1 Industrial Applications

In this section, we present four different industrial applications of virtual engineering which immediately benefit from an object-relational database integration. We have analyzed and evaluated them in cooperation with partners in the automotive and aerospace industry, including the DaimlerChrysler AG, Stuttgart, the Volkswagen AG, Wolfsburg, the German Aerospace Center DLR e.V., Oberpfaffenhofen, and the Boeing Company, Seattle.

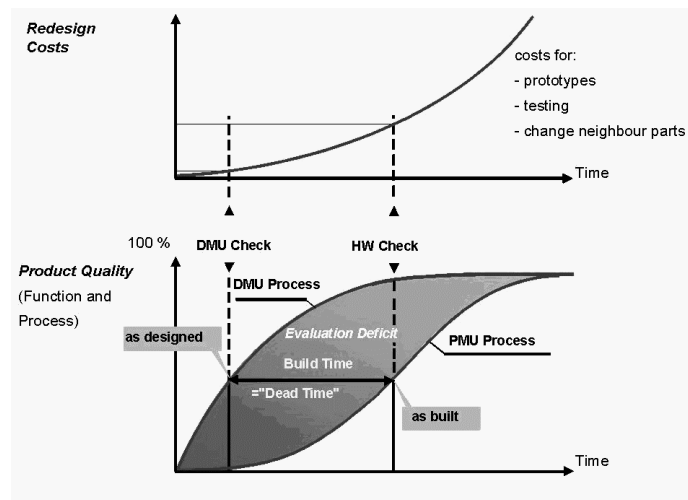


Figure 2: Digital mock-up (DMU) [IWB 01].

1.1.1 Digital Mockup

For a long time industrial practice in automotive, aerospace, and other manufacturing industries involves the creation of a number of physical product models. During the product development phase, both the product and the process are verified on the basis of *physical mock-up* (PMU). However, hardware checks cause tremendous time delays. Moreover, a late hardware verification very often leads to respectively late design changes, which are cost intensive (cf. Figure 2). 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 [CF 91].

Nowadays, the different industries do not have to build real test models anymore. The real models currently required for automobile development, for instance, are being dispensed with new *digital mockup* (DMU) methods, which combine all digital data from Computer Aided Design (CAD), Computer Aided Engineering (CAE) and Computer Aided Manufacturing (CAM), including the results from simulations and animations. The tools for the digital mockup of engineering products allow 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,

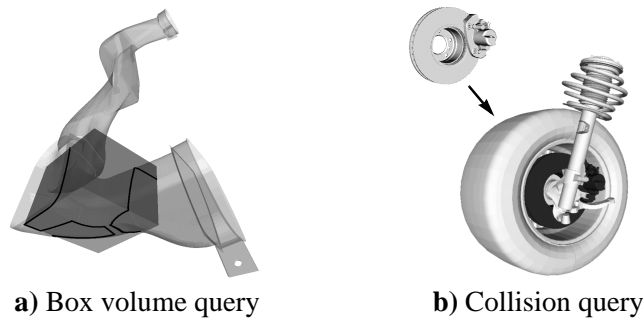


Figure 3: Spatial queries on CAD data.

spread over many file servers and are managed by a variety of users [BKP 98]. In a concurrent engineering process, several cross-functional project teams may be recruited from different departments, including engineering, production, and quality assurance to develop their own parts as a contribution to the whole product. However, the 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 in importance for industrial file exchange. Engineers, working in the United States, 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 EDM system. Figure 3 depicts two typical spatial queries on a three-dimensional product space, retrieving the parts intersecting a given box volume (*box volume query*), and detecting the parts colliding with the geometry of a query part (*collision query*). A spatial filter for DMU-related queries on huge CAD databases is easily implemented by a spatial access method which determines a tight superset of the parts qualifying for the query condition. Then, the computationally intensive query refinement on the resulting candidates, including the accurate evaluation of intersection regions (cf. Figure 3a), can be delegated to an appropriate main memory-based CAD tool.

1.1.2 Similarity Search

In the last ten years, an increasing number of database applications has emerged for which efficient and effective support for similarity search is substantial. The importance of similarity search grows in application areas such as multimedia, medical

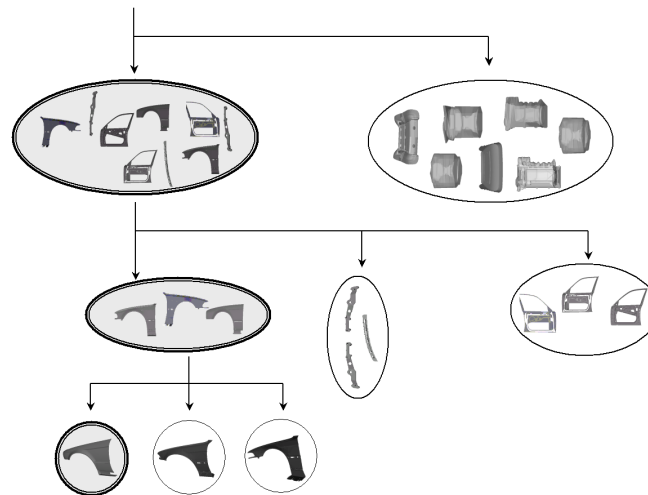


Figure 4: Browsing through a hierarchy of CAD objects.

imaging, molecular biology, computer aided engineering, marketing and purchasing assistance. Particularly, the task of finding similar shapes in 2D and 3D becomes more and more important.

The development, design, manufacturing and maintenance of modern engineering products is a very expensive and complex task. Shorter product cycles and a greater diversity of models are becoming decisive competitive factors in the hard-fought automobile and plane market. These demands can only be met if the engineers have an overview of already existing CAD parts. An engineer should be able to find those parts which are most similar to the one he has in mind. Even if he has no concrete part in mind, it would be very helpful for the user, if he could browse through the already existing parts in order to get an overview (cf. Figure 4). Such a system would help the companies to save time and money by avoiding unnecessary redesigns.

To sum up, with an increasing number of already existing CAD files we have to face new challenges. The question at issue is no longer how we can create new CAD files, but how we can find already designed and approved parts.

1.1.3 Further Applications

In this section, we shortly sketch two further application ranges of virtual engineering, while concentrating on *digital mockup* and *similarity search* throughout the remainder of this thesis.

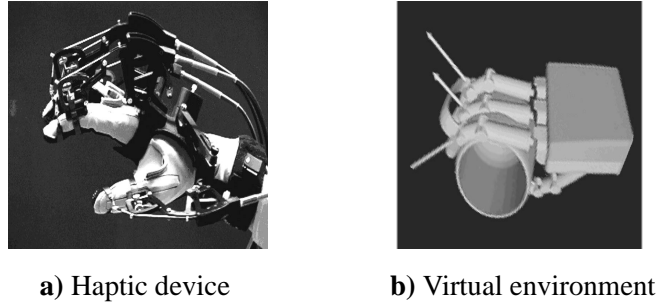


Figure 5: Sample scenario for haptic rendering.

Haptic Rendering. The modern transition from the physical to the digital mockup has stimulated the well-known problem of simulating real-world engineering and maintenance tasks. Therefore, many approaches have been developed to 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 [GLM 96] [LSW 99] [MPT 99]. Figure 5a 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 5b. 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 6 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 ro-

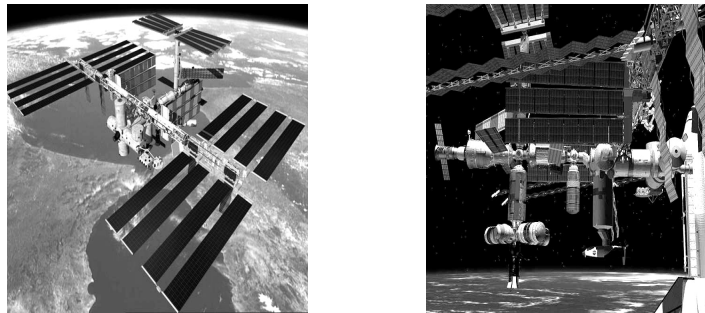


Figure 6: Virtual environment of the International Space Station.

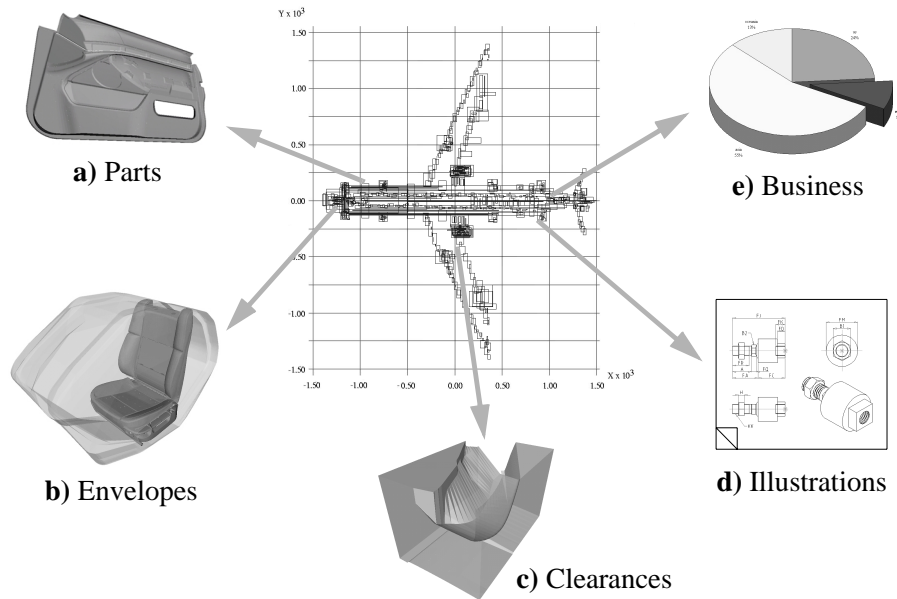


Figure 7: Spatial referencing of engineering documents.

bots, an index-based prefetching of persistent spatial objects can be coupled with real-time haptic rendering.

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 7), including kinematic envelopes which represent moving parts in 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 meetings concerning the design of a specific detail of a product may affect many different components. Spatial referencing provides a solution by simply attaching the meetings to a spatial key created for the region of interest. A very intuitive query could be: “retrieve all meetings of the previous month concerning the spatial region between parts A and B”. Such queries can be efficiently supported by spatial indexes.

1.2 Outline of the Thesis

This thesis discusses problems and solutions related to spatial database support for virtual engineering. It is divided into a short introductory part, called *virtual engineering*, and two major parts, called *database support for digital mockup* and *database support for similarity search*. The algorithms and ideas discussed in the different chapters of this thesis have already been published in parts. For clearness and convenience, we list these own publications in this section, while refining from citing them repeatedly throughout the remainder of this thesis:

Part I	<i>Virtual Engineering</i>
Chapter 1 Chapter 2	<i>Spatial Data Management for Virtual Product Development</i> [KPP+ 03]
Part II	<i>Database Support for Digital Mockup</i>
Chapter 3	<i>The Paradigm of Relational Indexing: A Survey</i> [KPPS 03b] <i>Object-Relational Spatial Indexing</i> [KPP+ 04] <i>Stochastic Driven Relational R-Tree</i> [KKP+ 03]
Chapter 4	<i>A Cost Model for Interval Intersection Queries on RI-Trees</i> [KPPS 02] <i>A Cost Model for Spatial Intersection Queries</i> [KPPS 04]
Chapter 5	<i>Acceleration of Relational Index Structures based on Statistics</i> [KKPR 03a] <i>Efficient Query Processing on the Relational Quadtree</i> [KKPR 03b] <i>Statistic Driven Acceleration of Object-Relational Space-Partitioning Index Structures</i> [KKPR 04b] <i>Efficient Query Processing on Relational Data-Partitioning Index Structures</i> [KKPR 04d]
Chapter 6	<i>Spatial Query Processing for High Resolutions</i> [KPPS 03a] <i>Cost-based Decompositioning of Complex Spatial Objects for Efficient Relational Indexing</i> [KKPR 04a] <i>Effective Decompositioning of Complex Spatial Objects into Intervals</i> [KKPR 04c] <i>Spatial Join Processing for High-Resolution Objects</i> [KKPR 04e]
Part III	<i>Database Support for Similarity Search</i>
Chapter 8 Chapter 9	<i>Effective Similarity Search on Voxelized CAD Objects</i> [KKM+ 03] <i>Using Sets of Feature Vectors for Similarity Search on Voxelized CAD Objects</i> [KBK+ 03]
Chapter 10	<i>Efficient Query Processing on Sets of Feature Vectors</i> [BKP 04a] <i>Indexing of Complex Objects for Efficient Density-Based Clustering</i> [KKPS 04]
Chapter 11	<i>Representatives for Visually Analyzing Cluster Hierarchies</i> [BKK+ 03] <i>BOSS: Browsing OPTICS-Plots for Similarity Search</i> [BKK+ 04] <i>Visually Mining Through Cluster Hierarchies</i> [BKPP 04]

1.2.1 Virtual Engineering (Part I)

In this first introductory part, we shortly discuss various aspects related to virtual engineering. After having mentioned some industrial application ranges of virtual engineering, in the first chapter, we look at spatial engineering databases in Chapter 2. We list various engineering data types and query types. Furthermore, we shortly sketch a generic approach for efficient spatial query processing.

1.2.2 Database Support for Digital Mockup (Part II)

The process of digital mockup is based on an effective and efficient detection of colliding spatial objects.

Chapter 3 motivates the use of an object-relational database management system (ORDBMS) for virtual engineering. We discuss the possibilities and limitations of extensibility in an ORDBMS. For the seamless integration of spatial index structures, two basic interfaces have to be addressed. First, a high-level interface is required to embed index operations into the declarative query language, i.e. into SQL. Second, a low-level interface of the index structure to the database kernel has to be implemented. We will show that the high-level interface is well supported by common ORDBMSs. After discussing the difficulties of implementing the low-level interface, we will present the paradigm of relational indexing which solves the low-level interface problem. We will develop the main properties of relational access methods, and illustrate the presented concepts by discussing the Relational R-tree (RR-tree), the Relational Quadtree (RQ-tree) and the Relational Interval Tree (RI-tree).

Chapter 4 completes the object-relational integration of spatial indexing by introducing a cost model to estimate the selectivity and I/O cost of spatial queries on the RI-tree. Similar to the integration of the index structure, the integration of the cost model into the ORDBMS can be achieved by implementing a high- and low-level interface. By design, the models immediately fit to common extensible indexing/optimization frameworks, and their implementation exploit the built-in statistics facilities of the database server. In a broad empirical evaluation, we show that the presented techniques yield accurate results which are indispensable for determining the best possible query execution plan.

Chapter 5 explains how we can accelerate the query process based on relational index structures. For space-partitioning index-structures, e.g. the RI-tree and the RQ-tree, we propose to use statistics which are required and maintained by the corresponding cost models to accelerate the query process. The main idea is to reduce the number of generated join partners which results in less logical reads. We cut down on the number of join partners by grouping different join partners together according to a statistic driven grouping algorithm. For hierarchical data-partitioning index structures, e.g. the RR-tree, we reduce the navigational index traversal cost by using “extended index range scans”. If a directory node is “largely” covered by the actual query, the recursive tree traversal for this node can beneficially be replaced by a scan on the leaf level of the index instead of navigating through the directory any longer. In our experiments, we show that the presented techniques considerably accelerate the RR-tree, the RQ-tree and the RI-tree.

Chapter 6 presents a cost based decomposition approach for complex spatial objects. In contrast to common black-and white decomposition algorithms which suffer from the lack of intermediate solutions, we introduce gray containers as a new and general concept which are stored in a compressed way within an ORDBMS. The gray containers are created by using a cost-based decomposition algorithm which takes the access probability and the decompression cost of the gray containers into account. The experimental evaluation on real-world test data points out that our new concept accelerates collision queries on the RR-tree, the RQ-tree, and the RI-tree as well as spatial join processing by up to two orders of magnitude.

1.2.3 Database Support for Similarity Search (Part III)

The effective and efficient retrieval of similar objects is a crucial factor for cutting down the time spent for developing and designing modern engineering products. In this part of the thesis, we show how effective similarity models supported by efficient access methods can turn themselves useful to an industrial user.

Chapter 7 is dedicated to the related work in the area of effective and efficient similarity search. After introducing the basic similarity query types, we explain how they can be integrated into an off-the-shelf ORDBMS. Thereafter, we present different similarity models and access methods from the literature which are used for effective and efficient similarity search.

Chapter 8 discusses invariance properties for effective similarity search on voxelized CAD data together with different similarity models. First, we introduce three different space partitioning similarity models, namely the *volume model*, the *solid-angle model* and the *eigen-value model*. Then, we turn our attention to data partitioning similarity models, starting with the *cover-sequence model*. This model serves as a starting point for the *vector set model* which is based on a new paradigm in similarity search. In contrast to the other four models, the vector set model uses sets of feature vectors for representing an object instead of single feature vectors.

Chapter 9 introduces density-based hierarchical clustering as a new and effective way to analyse and compare similarity models. We motivate this new evaluation approach by demonstrating its superiority compared to the commonly used k -nn queries which are subjective and error-prone. Based on this new evaluation method, we compare the quality of the different similarity models. We show that among the space partitioning models, the eigen-value model is the most suitable model. The quality of the space partitioning eigen-value model is comparable to the quality of the data-partitioning cover sequence model. Nevertheless, both of these models are outperformed by our data partitioning vector set model. To sum up, we show in this chapter that the vector set model yields by far the highest quality of all investigated similarity models.

Chapter 10 presents the efficiency evaluation of the five introduced similarity models. In order to accelerate the query processing for the feature vector based similarity models, i.e. the volume model, the solid-angle model, the eigen-value model and the cover sequence model, we use a common access method which can easily be integrated into an ORDBMS. For improving the query response time of the vector set model, we introduce three different filter steps, the *centroid approach*, the *Euclidean norm approach* and the *closest pair approach*. We present a broad experimental evaluation showing that the introduced filter steps considerably accelerate similarity queries based on the vector set model. Especially the combination of the centroid approach and the Euclidean norm approach yields very good results. Furthermore, we present the Relational M-tree (RM-tree) along with suitable optimizations. Nevertheless, similarity queries based on the high quality vector set model are still slower than similarity queries on the single feature vector models. The efficiency evaluation of this chapter together with the effectiveness evaluation of the last chapter help the user to find an individual trade-off between quality and query response time.

Chapter 11 illustrates how an appropriate visualization of the hierarchical clustering structure can aid the user in his time consuming task to find similar objects. We introduce approaches which automatically extract the significant clusters in a hierarchical cluster representation along with suitable cluster representatives. These techniques can be used as a basis for visual data mining. We implemented our algorithms resulting in an industrial prototype which was also used for the experimental evaluation presented in Chapter 9.

Chapter 12 recapitulates the main contributions of this thesis and suggests some directions for future work.

Chapter 2

Spatial Engineering Databases

A database system (DBS) is designed to manage and analyze huge amounts of persistent data, offering important advantages compared to a file-based organization. DBSs provide logical and physical data independence, transactions, concurrency control, integrity checking, recovery, security, standardization, and distribution [Dat 99].

One of the most promising data models for DBSs is the *object-relational* model. It provides two substantial advantages. First, the practical impact of object-relational database management systems is very strong, as object-relational functionality has been added to most commercially available relational database servers, including Oracle [Doh 98], IBM DB2 [CCN+ 99], and Informix IDS/UDO [Bro 01]. Secondly, its extensibility is a necessary prerequisite for the seamless embedding of user-defined data types and predicates, which is vital for virtual engineering. Defining spatial data types and spatial predicates on top of any off-the-shelf ORDBMS enables us to ask all kinds of similarity and intersection queries. Furthermore, integrating these spatial features into an ORDBMS allows us to combine structural queries as, for instance, “retrieve all documents, that refer to the current version of the jet engine” with the evaluation of geometric predicates. To put it another way, ORDBMSs allow us to easily combine EDM systems with spatial database systems.

According to Güting [Güt 94], spatial database systems could be defined in the following way:

- A spatial database system is a database system.
- It offers spatial data types in its data model and query language.
- It supports spatial data types in its implementation, providing at least spatial indexing.

This definition points up, that a spatial database system is a fully-fledged database system, with additional modules for handling spatial data. The extensibility interfaces of most ORDBMSs, including Oracle [Ora 99a][SMS+ 00], IBM DB2 [IBM 99] [CCF+ 99], or Informix IDS/UDO [Inf 98][BSSJ 99], enable us to integrate spatial requirements into off-the-shelf object-relational database systems.

In Section 2.1, we introduce different data representation types for managing spatial objects within an ORDBMS. In Section 2.2, we look at effective and efficient spatial query processing. First, we introduce the main spatial *query predicates* discussed in this thesis. Second, we sketch the general paradigm of *multi-step query processing*.

2.1 Spatial Engineering Data

An engineering product can be regarded as a collection of individual, three-dimensional parts, while each part potentially represents a complex and intricate geometric shape. The original surfaces and solids are designed at a very high precision. In order to cope with the demands of accurate geometric modeling, highly specialized CAD applications are employed, using different data primitives and native encodings for spatial data. To homogenize these different encodings, several neutral file formats have been defined, including the popular standards VDAFS [VDA 87], IGES [IGES 96], STEP [STEP 94+], and VRML [CBM 97]. An enterprise-wide spatial CAD database should rely on one or more of these data exchange formats. In order to integrate the different geometric semantics, we propose a set of universal representations which can be derived from any native geometric surface and solid. The supported geometric data models include triangle meshes for visualization and interference detection (cf. Section 2.1.1), as well as voxel sets as conservative approximations for spatial keys (cf. Section 2.1.2). In the area of similarity search, feature vectors are

used to represent spatial objects (cf. Section 2.1.3). Dependent on the chosen similarity model, the feature vectors contain compact information which is derived from another appropriate data representation of the spatial object, e.g. triangle meshes or voxel sets.

As already mentioned, we will concentrate on database support for digital mock-up (cf. Section 1.1.1) and similarity search (cf. Section 1.1.2) in this thesis. For these two application ranges, voxel sets form an adequate representation of CAD objects. Although we confine ourselves to this specific data model throughout the remainder of this thesis, we place voxel sets in a broader context in this section.

2.1.1 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 [MH 99]. As a solution, approximative polygon (e.g. triangle) meshes can be derived from the accurate surface representation. These triangle meshes allow for 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. We distinguish three actions for interference detection [MH 99]: *collision detection*, *collision determination*, and *collision response*:

- **Collision detection:** This basic interference check simply detects if the query part q and a database object o 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:** 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 or visual feedback on the interfering parts and, if computed, the intersection lines seems to be appropriate.

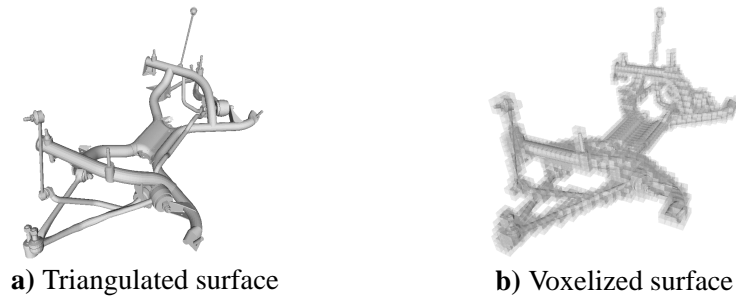


Figure 8: Scan conversion on a triangulated surface.

2.1.2 Voxelized Data

In order to employ spatial index structures for our CAD database, we propose a conversion pipeline to transform the geometry of each single CAD part to an interval sequence by means of voxelization. In the following, we assume a uniform three-dimensional voxel grid covering the global product space.

The voxelization of polygon meshes is a major research topic in the field of computer graphics and CAD. Voxelization techniques and applications have been proposed for instance for interactive volume visualization [HYFK 98] and haptic rendering [MPT 99]. A basic algorithm for the 3D scan-conversion of polygons into a voxel-based occupancy map has been proposed by Kaufmann [Kau 87]. If we apply this conversion to the given triangle mesh of a CAD object (cf. Figure 8a), a conservative approximation of the part surface is produced (cf. Figure 8b).

Solid Object. 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. Thus, 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 9a), which yields a consistent representation of the object surface. Next, we apply a 3D flood-fill algorithm [FDFH 00] to compute the exterior voxels of the object (cf. Figure 9b), and thus, determine the outermost boundary voxels of the solid. 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

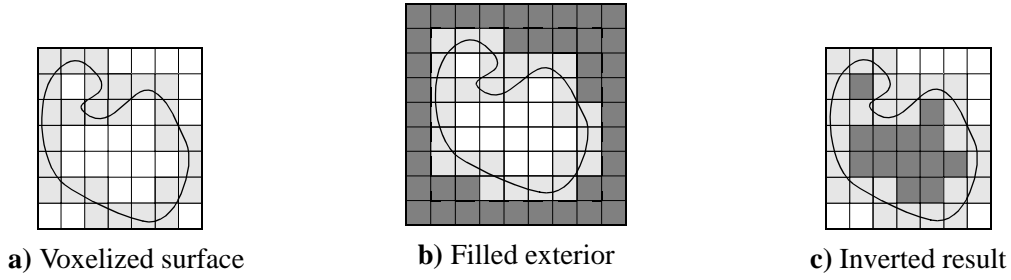


Figure 9: Filling a closed voxelized surface.

voxels as interior which are neither boundary nor exterior voxels (cf. Figure 9c). In consequence, we obtain a volumetric reconstruction of the original solid, marking any voxels behind the outermost surface as interior [Pöt 01].

Space Filling Curves. 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 10.

Spatial Index Entries. Voxels can be grouped together to *tiles*, *intervals*, or *boxes* which can efficiently be managed by spatial index structures. A basic parameter for the mapping of extended objects to these basic spatial data types is the granularity, i.e. the resolution of the underlying grid. When refining the resolution, the approximations become more accurate but redundancy increases. Figure 11a illustrates the granularity-bound decomposition into variable-sized Z-tiles (top row) and into

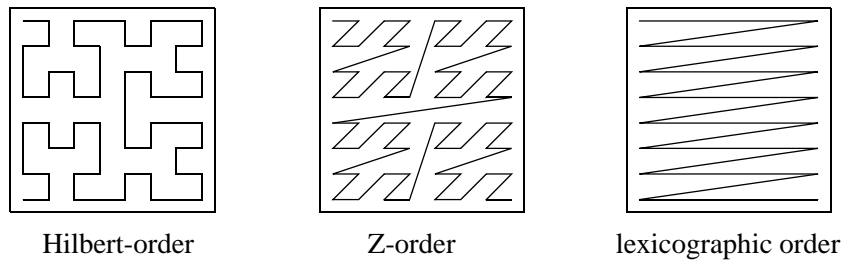


Figure 10: Examples of space-filling curves in the two-dimensional case.

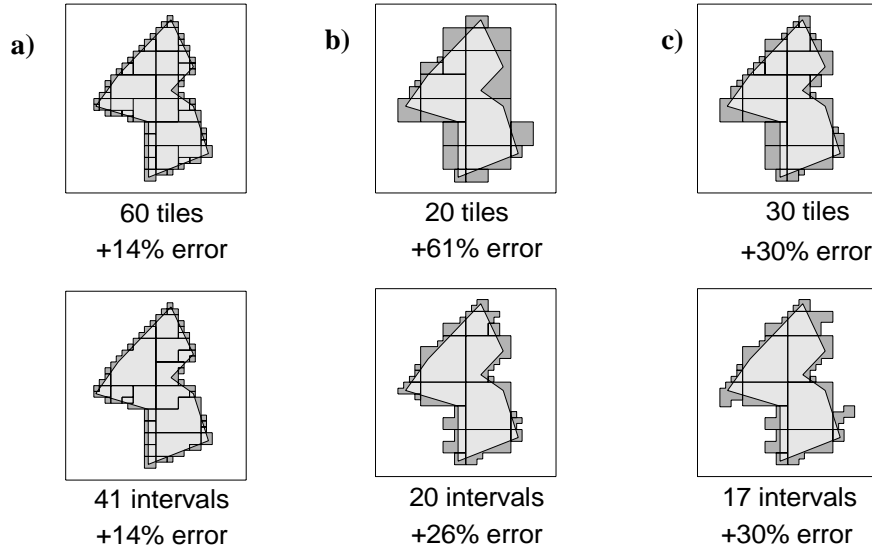


Figure 11: Object decompositions.

a) Granularity-bound, **b)** Size-bound, **c)** Error-bound decomposition into Z-tiles (top row) and Z-ordered interval sequences (bottom row) [KPS 01].

Z-ordered interval sequences (bottom row). The approximation error is the ratio of the dead space to the object area. According to the extensive analysis given in [FJM 97] and [MJFS 96], the asymptotic redundancy of a tile- or interval-based decomposition is in both cases proportional to the surface of the approximated object. Nevertheless, as intervals on a Z- or Hilbert-curve may span many tiles, their average number is lower than the number of tiles [Vei 03] (cf. Figure 11). The Hilbert-order generates the minimum number of intervals and tiles per object [Jag 90] [FR 89] but unfortunately, it is the most complex linear order. Taking redundancy and complexity into consideration, the Z-order seems to be a good solution.

Decompositioning. On top of the resolution of the data space and the clustering properties of the space-filling curve, a more fine-grained control of the trade-off between redundancy and accuracy is desired for many applications. First, the granularity may have to differ for each individual object rather than to apply the same resolution to all objects. Second, the resolution is fixed at database creation time whereas an object may have to be approximated differently at insertion time and at query time. An approach to control this trade-off is the concept of size-bound and error-bound approximation [Ore 89] beyond the mentioned granularity-bound approximation [Gae 95]. The decompositioning is based on a recursive subdivision

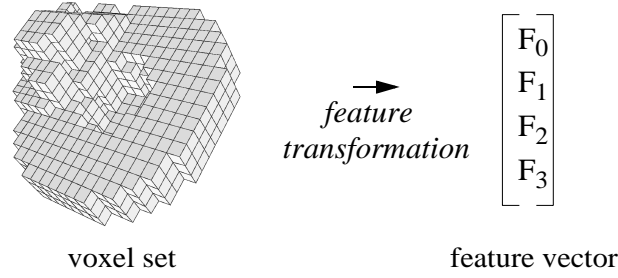


Figure 12: Feature transformation.

procedure which stops if the desired redundancy (size-bound) or the desired maximum approximation error (error-bound) is reached. In [KPS 01], it was demonstrated how to adapt the algorithms of [Ore 89] by integrating the management of generated intervals into the recursive spatial decomposition. Figure 11b illustrates the size-bound approximation of a polygon into variable-sized tiles (top row) and into Z-ordered interval sequences (bottom row). Examples for an error-bound approximation are depicted in Figure 11c.

2.1.3 Feature Vectors

Triangle meshes as well as voxel sets are representations which are both unsuitable for similarity search as there exist no appropriate distance functions for sets of voxels and for sets of triangle meshes. As distance functions form the foundation of similarity search, we need an object representation which allows efficient and meaningful distance computations. As we can easily compute the distance between two numerical vectors, a common approach is to represent a spatially extended object by a numerical vector. In this case a feature transformation extracts distinguishable spatial characteristics which are represented by numerical values grouped together in a feature vector (cf. Figure 12).

On the basis of such a feature transformation and under the assumption that similarity corresponds to feature distance, it is possible to define a similarity measure (function) for two data objects as the distance between the corresponding feature vectors. Thus, searching for similar data objects to a given query data object is transformed into *proximity search* in the feature space. Most applications use the Euclidean metric (L_2) to evaluate the feature distance, but there are several other metrics commonly used, e.g. the Manhattan metric (L_1), and the Maximum metric (L_∞).

2.2 Spatial Engineering Query Processing

Based on the different data representations discussed in the last section, we can effectively and efficiently process engineering queries. In this section, we first introduce those query types which we will focus on in this thesis. The first two query types are related to collision queries (cf. Part II), and the remaining ones to similarity search (cf. Part III). Secondly, we will shortly outline the foundation of efficient query processing, namely *spatial index structures* and the paradigm of *multi-step query processing*.

2.2.1 Effective Query Processing

A CAD database management system should enable engineers to find colliding and similar parts. In this section, we will informally introduce the necessary query types, without specifying the actual data representation of the CAD objects. The queries can be based on triangle meshes, or on some derived information, as for instance voxelized data. More detailed definitions are presented in Part II and Part III of this thesis when required.

In the following, let O be the domain of all objects that may occur as database objects or query objects. Furthermore, let $DB \subseteq O$ denote a database.

- **Boolean Intersection Query:** A boolean intersection query yields true, if two objects $o_1, o_2 \in O$ intersect and false if they do not intersect, i.e. $intersect_{boolean}: O \times O \rightarrow \{true, false\}$. Note, that they might intersect based on their voxelization, but do not intersect if we look at the finer triangle meshes (cf. collision detection in Section 2.1.1). If it is clear from the context, we simply use the term *intersect* instead of $intersect_{boolean}$.
- **Ranked Intersection Query:** In contrast to a boolean intersection query a ranked intersection query $intersect_{ranked}$ yields not a boolean return value, but a numerical one, i.e. $intersect_{ranked}: O \times O \rightarrow IR_0^+$. For voxelized data, the return value of $intersect_{ranked}$ indicates the intersection volume, whereas for triangle meshes it is equal to the intersection length. If two objects $o_1, o_2 \in O$ do not intersect, i.e. $intersect_{boolean}(o_1, o_2) = false$, then $intersect_{ranked}(o_1, o_2) = 0$ holds (cf. collision determination in Section 2.1.1).

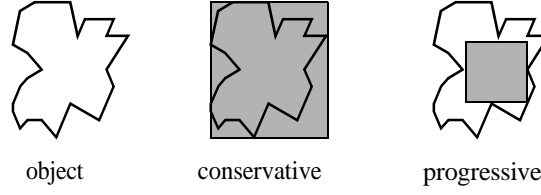


Figure 13: Conservative and progressive approximations.

- **Similarity Range Query:** In a similarity range query $sim_{range}: O \times IR_0^+ \rightarrow 2^{DB}$, the user specifies a query object $q \in O$ and a query radius $\epsilon \in IR_0^+$. The system retrieves all objects from the database that have a distance, e.g. a feature distance, from q which does not exceed ϵ .
- **Similarity k -nn query:** In a similarity k -nn query $sim_{knn}: O \times IN \rightarrow 2^{DB}$, the user specifies a query object $q \in O$ and the cardinality $k \in IN$ of the result set. The system retrieves those k objects from the database which are closest to q .
- **Similarity Ranking Query:** In a similarity ranking query the user specifies a query object $q \in O$ and the system retrieves all objects from the database ordered by their distance to q , i.e. $sim_{rank}: O \rightarrow (\{1..|DB|\} \rightarrow DB)$. For reasons of efficiency, the ranking procedure should not be performed and completed in advance at query initialization time. Instead, the ranking procedure should incrementally proceed, especially, because users are often satisfied with the first few elements.

2.2.2 Efficient Query Processing

A CAD database system has to provide an effective and efficient query processor in order to render itself useful for the user. *Spatial indexing* and the *multi-step query processing strategy* are the basic concepts of this fundamental system component [BKSS 94]. In this section, we will briefly look at spatial indexing and multi-step query processing which form the foundation for efficient query processing.

Spatial Index Structures. *Spatial index structures*, or synonymously, *spatial access methods*, partition the multidimensional search space for spatial queries. Particularly for spatial selections and spatial similarity search, spatial indexing allows the query processor to quickly exclude many irrelevant objects. Thereby, only a subset of the database has to be considered to detect the actual query results. As the accurate representations of spatial objects can have arbitrary complexity, spatial index structures typically use *conservative approximations* (cf. Figure 13) to maintain their knowledge about the spatial shape and location of each object.

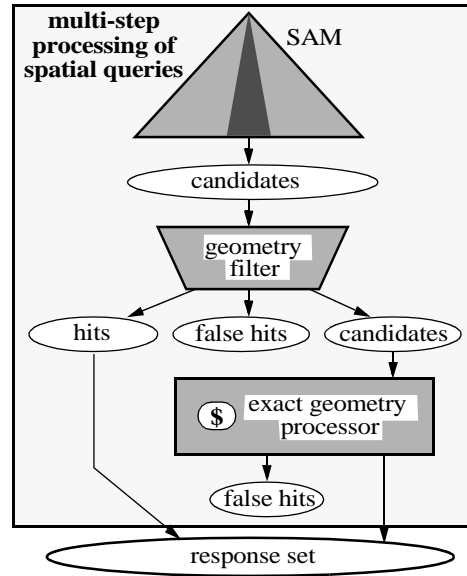


Figure 14: Multi-step query processing.

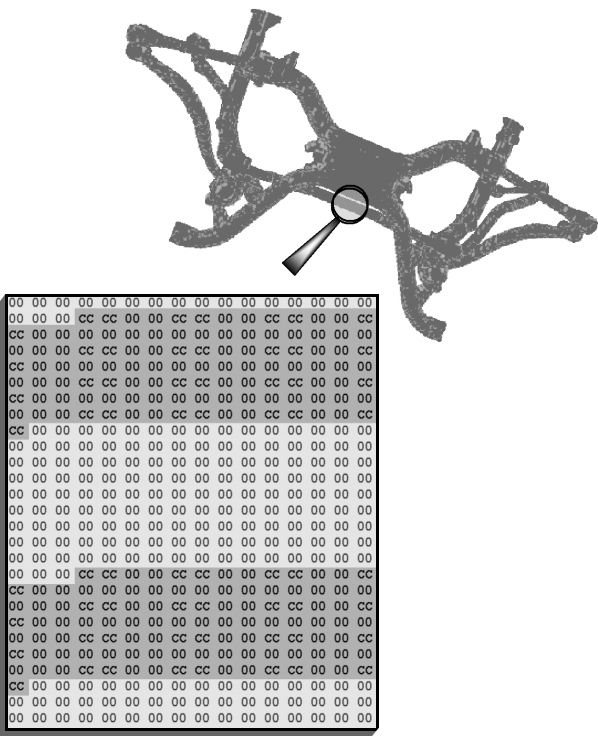
Multi-Step Strategy. The use of approximations results in a multi-step strategy to process spatial queries (cf. Figure 14). First, a *filter step* is executed returning a superset of the objects qualifying for the spatial predicate. As the filter step is based on conservative approximations of the objects, no false drops are possible, i.e. all objects satisfying the predicate are included in the resulting *set of candidates*. A cascade of subsequent filter steps may further reduce the number of candidates, e.g. by the usage of more accurate representations or *progressive approximations* (cf. Figure 13) [BKS 93a] [BKSS 94]. These progressive approximations are also adequate for identifying objects which already belong to the result set. For highly selective predicates, the first filter step should be processed on a suitable spatial access method (SAM). The multi-step query process is finished by the *refinement step* (cf. Figure 14), which checks the exact geometry.

- **Intersection Queries:** In the case of intersection queries, the filter step is based on the voxelized data, whereas the refinement step is carried out on the triangle meshes. For many applications, the voxelized data are accurate enough so that we can abstain from an additional refinement step. In this thesis, we solely concentrate on efficiently detecting intersecting voxelized objects.

- **Similarity Queries:** In the case of similarity queries the filter step might be carried out on feature vectors or single numerical values approximating the objects coarsely. Sometimes there is a more accurate distance function, which might be applied in the refinement step. In order to omit false hits, the distance function applied in the filter step has to lower bound the object distance function (cf. Definition 22).

Part II

Database Support for Digital Mockup



Chapter 3

Object Relational Indexing

The design of extensible architectures represents an important area in database research. The object-relational data model marked an evolutionary milestone by introducing abstract data types into relational database servers. Thereby, object-relational database systems may be used as a natural basis to design an integrated user-defined database solution. The ORDBMSs already support major aspects of the declarative embedding of user-defined data types and predicates. In order to achieve a seamless integration of custom object types and predicates within the declarative data definition language (DDL) and data manipulation language (DML), ORDBMSs provide the database developer with extensibility interfaces. They enable the declarative embedding of abstract data types within the built-in optimizer and query processor. Corresponding frameworks are available for most object-relational database systems. Custom server components using these built-in services are called *data cartridges*, *database extenders*, and *data blades*, in Oracle, DB2 and Informix, respectively.

In this section, we categorize possible approaches to incorporate third-party indexing structures into an object-relational database system by what we call *Relational Indexing*. After shortly discussing the main properties of ORDBMSs in Section 3.1, we will look at the extensible indexing facilities of modern database systems in Section 3.2. In Section 3.3, we discuss three different implementations of user-defined access methods, including the relational approach. In Section 3.4, basic concepts of relational access methods are introduced, and in Section 3.5, the design of

the corresponding update and query operations are investigated. In Section 3.6 and Section 3.7, we identify two generic schemes for modeling relational access methods which are discussed with respect to their support of concurrent transactions and recovery. Finally, in Section 3.8, we introduce an industrial prototype which allows interactive spatial data management on top of object-relational database systems.

3.1 Object Relational Databases

3.1.1 Classification of Data Models

Users of database systems want to manage data of very different types, depending on the particular application area. While office applications, for example, mainly perform simple access and update operations on records of simple data types, spatial data usually have a complex structure and demand specialized operations. It is not a choice for vendors of database management systems to provide data types and management functions for each conceivable domain. So the design of extensible architectures allowing users to adapt systems to their special needs represents an important area in database research.

Stonebraker and Brown [SB 98] considered the complexity of the stored data and submitted queries to classify some common data models. Four major groups were identified (cf. Figure 15):

File system. File-based applications use the file system of the underlying operating system to access data located on secondary storage. When a user opens a file, the content of the file is being read into main memory on behalf of the user. Then the user can make modifications that update the main memory object. Finally, when the user is finished he can close the file, thereby causing the main memory copy to be stored back to the file system. The only query made by a file-based application is *getfile*, and the only update is *putfile*. The application is satisfied with the data model presented by the file system, namely as an arbitrary length sequence of characters. Therefore, file-based applications are “simple query - simple data” applications.

Relational DBMS (RDBMS). In an RDBMS, all data have to be mapped on rows of flat tables consisting of attributes with simple types like numbers, character strings or dates. For the retrieval and manipulation of data, there exist only generic operations for selecting, inserting, updating and deleting (parts of) rows within tables. Data

	Simple data	Complex data
Simple queries	File system	Object-oriented DBMS
Complex queries	Relational DBMS	Object-relational DBMS

Figure 15: Classification of data models.

of more complex types cannot be stored directly as a unit in the database but have to be split across several tables. To restore the data from the system, complex queries with many joins have to be performed. Alternatively, the data can be coded within a large object which prevents direct access to single components of the data using the database language. Operations on complex types have to be implemented within the application and cannot be used within the database language directly.

Object-oriented DBMS (OODBMS). OODBMSs seem to provide solutions for most of the cited problems of relational databases. An OODBMS has an extensible type system which allows the user to define new data types (by the nested application of type constructors) together with corresponding operations. The resulting object types then describe the structure as well as the behavior of the objects based on this type. Furthermore, subtypes (inheriting the properties of their supertypes) can be derived from existing object types.

Object-relational DBMS (ORDBMS). In order to provide object-oriented and extensibility features also in relational systems, database researchers and manufacturers proposed and implemented corresponding enhancements for the relational model during the last years. The resulting ORDBMSs retain all features of the relational model, especially the storage of data within tables and the powerful declarative query processing with the relational database language SQL. Beyond that, the object-relational data model introduces abstract data types into relational database servers. Thereby, object-relational database systems may be used as a natural basis to design an integrated user-defined database solution. The ORDBMSs already support major aspects of the declarative embedding of user-defined data types and predicates. In order to achieve a seamless integration of custom object types and predicates within the declarative DDL and DML, ORDBMSs provide the database developer with extensibility interfaces. They enable the declarative embedding of abstract data types within the built-in optimizer and query processor.

In this thesis, we focus on object-relational database management systems, as they combine the advantages of both, the object-oriented and the relational data model. Their extensible design enables us to integrate new access methods which are necessary for efficiently carrying out engineering queries. In addition, the practical impact of ORDBMSs is very strong as object-relational functionality has been added to most commercially available relational database servers, including Oracle [Doh 98], IBM DB2 [CCN+ 99], and Informix IDS/UDO [Bro 01].

3.1.2 Abstract Data Types

DDL statements like *CREATE*, *ALTER* and *DROP* have been extended in SQL:1999 [SQL 99] to support the declaration and implementation of *abstract data types* [Bro 01][CZ 01], often also referred to as *object types*. According to [Ora 99c], an object type is a schema object with three kinds of components:

- *A name*, which identifies the object type uniquely within that schema.
- *Attributes*, which model the structure and state of the real-world entity. Attributes can be built-in types or object types.
- *Methods* of an object type are functions or procedures that are called by the application to model the behavior of the objects. Methods can be stored in the database, which is preferable for data-intensive procedures and short procedures that are called frequently.

3.2 Extensible Query Language

Most ORDBMSs, including Oracle [Ora 99a] [SMS+ 00], IBM DB2 [IBM 99] [CCF+ 99], or Informix IDS/UDO [Inf 98] [BSSJ 99], provide extensibility interfaces in order to enable database developers to seamlessly integrate custom object types and predicates within the declarative DDL and DML. In this section, we shortly outline the extensible indexing and optimizing frameworks of modern ORDBMSs.

3.2.1 Extensible Indexing

An important requirement for applications is the availability of user-defined access methods. Extensible indexing frameworks proposed by Stonebraker [Sto 86] enable developers to register custom secondary access methods at the database server in addition to the built-in index structures. An object-relational *indextype* encapsu-

Function	Task
<code>index_create()</code> , <code>index_drop()</code>	Create and drop a custom index.
<code>index_open()</code> , <code>index_close()</code>	Open and close a custom index.
<code>index_fetch()</code>	Fetch the next record from the index that meets the query predicate.
<code>index_insert()</code> , <code>index_delete()</code> , <code>index_update()</code>	Add, delete and update a record of the index.

Figure 16: Methods for extensible index definition and manipulation.

lates stored functions for creating and dropping a custom index and for opening and closing index scans. The row-based processing of selections and update operations follows the iterator pattern [GHJV 95]. Thereby, the `indextype` complements the functional implementation of user-defined predicates. Figure 16 shows some basic `indextype` methods invoked by extensible indexing frameworks. Additional functions exist to support query optimization, and user-defined aggregates.

If the optimizer decides to include a custom index into the execution plan for a declarative DML statement, the appropriate `indextype` functions are called by the built-in query processor of the database server. Thereby, the maintenance and access of a custom index structure is completely hidden from the user, and the desired data independence is achieved. Furthermore, the framework guarantees any redundant index data to remain consistent with the user data.

Exemplarily, we create an object type *CADOBJECT* to encapsulate the data and semantics of three-dimensional CAD objects. Instances of this custom object type are stored as elements of relational tuples. Figure 17 depicts some of the required object-relational DDL statements in pseudo SQL thus abstracting from technical details which depend on the chosen product.

After having created a custom `indextype` *CADINDEX* for the *intersect* predicates on the type *CADOBJECT*, we can create an index *CADIDX* on the *geom* attribute of the *CADOBJECTS* table by submitting the usual DDL statement (cf. Figure 18).

By using either the functional or the index-based binding of the user-defined predicate *intersect_boolean*, object-relational queries can be expressed in the usual declarative fashion (cf. Figure 19a).

```

// Type declaration
CREATE TYPE VOXEL AS OBJECT (x NUMBER, y NUMBER, z NUMBER);
CREATE TYPE VOXELSET AS TABLE OF VOXEL;
CREATE TYPE CADOBJECT AS OBJECT (
    voxels VOXELSET,
    MEMBER FUNCTION intersect_boolean (o CADOBJECT)
        RETURN BOOLEAN,
    MEMBER FUNCTION intersect_ranked (o CADOBJECT)
        RETURN NUMBER
);
// Type implementation
// ...

// Functional predicate binding
CREATE OPERATOR intersect_boolean (a CADOBJECT, b CADOBJECT)
RETURN BOOLEAN BEGIN RETURN a.intersect_boolean(b); END;
CREATE OPERATOR intersect_ranked (a CADOBJECT, b CADOBJECT)
RETURN NUMBER BEGIN RETURN a.intersect_ranked (b); END;

// Table definition
CREATE TABLE CADOBJECTS
(id NUMBER PRIMARY KEY, geom CADOBJECT);

```

Figure 17: Object-relational DDL statements for CAD data.

The integration of ranked intersection queries is a little bit more complex, but can be achieved by means of *ancillary operators* [Ora 99b]. An ancillary operator, e.g. *Rank*, has a functional implementation that has access to state generated by the index based implementation of the primary operator, e.g. *intersect_ranked*, occurring in the WHERE clause. By means of a common tag, e.g. 1, the ancillary operator and the primary operator are connected to each other (cf. Figure 19b). Note, the execution of this statement requires an index scan for the *intersect_ranked* operator [Ora 99b].

3.2.2 Extensible Optimizing

Query optimization is the process of choosing the most efficient way to execute a declarative DML statement. Object-relational database systems typically support rule-based and cost-based query optimization, whereby the cost-based approach is preferable to the rule-based approach when referencing user-defined methods as

- a)

```
CREATE TYPE CADINDEX_IM AS OBJECT (  
    // Attributes  
    // ODCI-Functions  
    STATIC FUNCTION ODCIIndexCreate    ...,  
    STATIC FUNCTION ODCIIndexStart    ...,  
    MEMBER FUNCTION ODCIIndexFetch    ...,  
    MEMBER FUNCTION ODCIIndexClose    ...,  
    ....  
    // Additional Functions  
    .... );
```
- b)

```
CREATE INDEXTYPE CADINDEX  
FOR intersect_boolean(CADOBJECT, CADOBJECT),  
    intersect_ranked (CADOBJECT, CADOBJECT)  
USING CADINDEX_IM;
```
- c)

```
CREATE INDEX CADIDX ON CADOBJECTS (geom)  
INDEXTYPE IS CADINDEX;
```

Figure 18: A custom index *CADINDEX* for CAD objects.

a) Implementation of indextype, b) Creation of indextype, c) Creation of an instance

predicates [BO 99][HS 93]. The extensible indexing framework comprises interfaces to tell the built-in optimizer about the characteristics of a custom indextype. Figure 20 shows some cost-based functions, which can be implemented to provide the optimizer with feedback on the expected index behavior. The computation of custom statistics is triggered by the usual administrative SQL statements. With a cost model registered at the built-in optimizer framework, the cost-based optimizer is able

- a)

```
// Collision query  
SELECT id FROM CADOBJECTS  
WHERE intersect_boolean (geom, :query_obj);
```
- b)

```
// Ranked collision query  
SELECT id, Rank (1)  
FROM CADOBJECTS db  
WHERE intersect_ranked (geom, :query_obj, 1)
```

Figure 19: SQL-statements for intersection queries.

a) Boolean intersection queries, b) Ranked intersection queries

Function	Task
stats_collect(), stats_delete()	Collect and delete persistent statistics on the custom index.
predicate_sel()	Estimate the selectivity of a user-defined predicate by using the persistent statistics.
index_cpu_cost(), index_io_cost()	Estimate the CPU and I/O cost required to evaluate a user-defined predicate on the custom index.

Figure 20: Methods for extensible query optimization.

to rank the potential usage of a custom access method among alternative access paths. Thus, the system supports the generation of efficient execution plans for queries containing user-defined predicates. This approach preserves the declarative paradigm of SQL, as it requires no manual query rewriting.

To sum up, the main advantages of extensible indexing and optimizing frameworks are:

- The maintenance and access of a custom index structure is completely hidden from the user, achieving thereby data independence.
- Any redundant index data remains consistent with the user data. The declarative paradigm of SQL is preserved.
- The index structure can be integrated into the cost-based query optimization process.

3.3 Implementation of Access Methods

In the previous section, we have outlined how object-relational database systems support the logical embedding of custom indextypes into the declarative query language and into the optimizer framework. The required high-level interfaces can be found in any commercial ORDBMS and are continuously improved and extended by the database vendors. Whereas the embedding of a custom indextype is therefore well supported, its actual implementation within a fully-fledged database kernel remains an open problem. In the following, we discuss three basic approaches to implement the low-level functionality of a user-defined access method: the *integrating*, the *generic*, and the *relational approach* (cf. Figure 21).

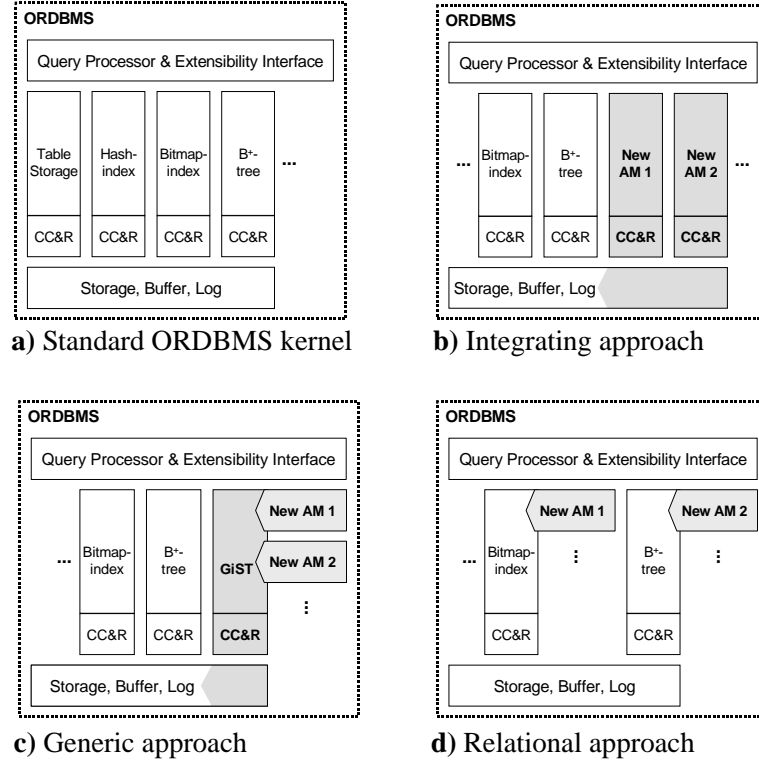


Figure 21: Approaches to implement custom access methods.

3.3.1 Integrating Approach

By following the integrating approach, a new spatial access method (AM) is hard-wired into the kernel of an existing database system (cf. Figure 21b). In consequence, the required support of ACID properties, including concurrency control and recovery services (CC&R) has to be implemented from scratch and linked to the corresponding built-in components. Furthermore, a custom gateway to the built-in storage, buffer, and log managers has to be provided by the developer of the new AM. Most standard storage structures are hard-wired within the database kernel, including plain table storage, hash indexes, bitmap indexes, and B^+ -trees. Only a few non-standard access methods have been implemented into commercial systems in the same way, including the *R-Link-tree* in Informix IDS/UDO for spatially extended objects [Inf 99] and the *UB-tree* in TransBase/HC for multidimensional point databases [RMF+ 00]. The integrating approach comprises the *extending approach* and the *enhancing approach*.

The Extending Approach. Adding a completely new access method to a database system is quite expensive because in addition to the actual index algorithms all the concurrency, recovery and page management has to be implemented for the new access method. Carey et al. [CDG+ 90] guess that the actual index algorithms only comprise about 30 percent of the overall code for the access method while the other 70 percent are needed to integrate the access method properly into the database system.

Several approaches to facilitate the implementation of access methods and other components for special-purpose database systems have been proposed in database research. Under the assumption that it is practically impossible to implement a database management system capable to fulfill the demands of arbitrary application domains, tools and generic database components have been developed that should enable a domain specialist to implement his or her required database system with minimum effort, may it be a document management system or a CAD database. The resulting systems might have completely different characteristics, e. g. different query languages, different access methods, different storage management, and different transaction mechanisms.

The database system toolkit EXODUS [CDG+ 90][CDF+ 91] provides a storage manager for objects, a library of access methods, a library of operator methods for the data model to generate, a rule-based generator for query optimizers, tools for constructing query languages and a persistent programming language for the definition of new access methods and query operators. Using these “tools”, the domain specialist can build an application-specific database system with suitable access methods. Another system of this category is the database generator GENESIS [BBG+ 90] which provides a set of composable storage and indexing primitives and a “database system compiler” for assembling an appropriate storage manager from a specification. Unfortunately, these universally extensible database systems have essentially proven to be hard to use in practice. Though these systems support the user with the implementation of single database components, still a lot of expertise is required to use them. In some ways, they are also a bit too inflexible and incomplete to implement a fully-fledged database management system. So in practice, only few databases have been implemented using such toolkits or generators.

In [BBD+ 01][BDS 00], XXL (eXtensible and fleXible Library) was introduced which is a high-level, easy-to-use, platform independent Java library. It supports the

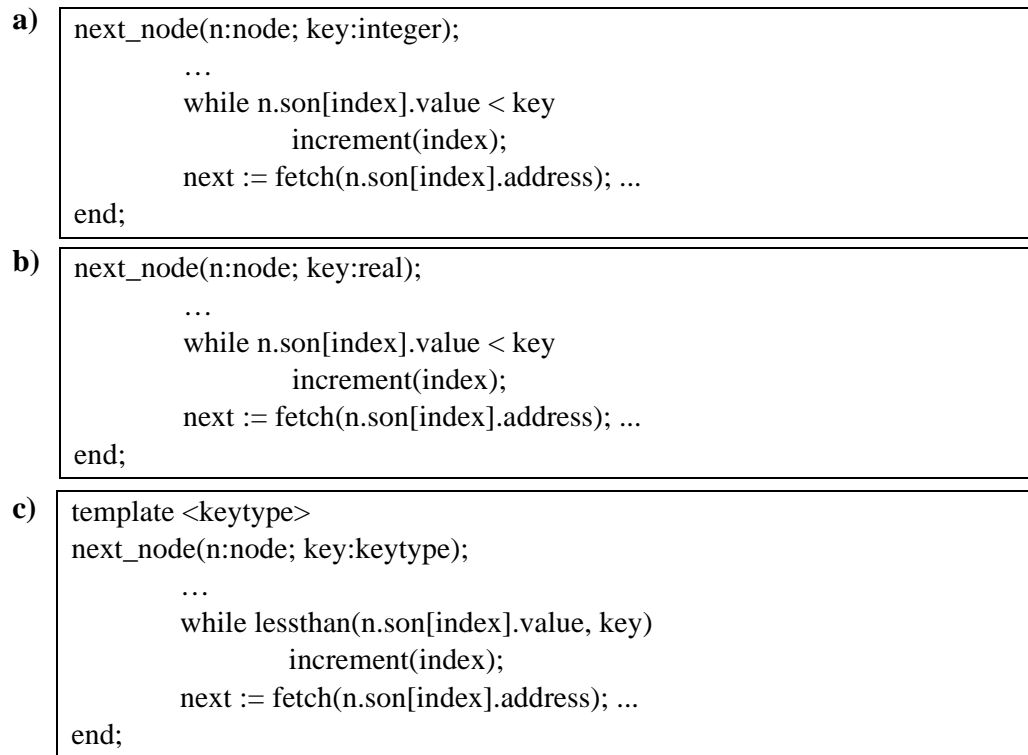


Figure 22: B-tree routine next_node for different data types.

a) Integer, b) Real, c) Arbitrary data type

implementation of new access methods by providing a powerful collection of generic index structures along with some concrete implementations. XXL could be used as a starting point for creating a new domain specific database system. In this case, the user has to augment the library with indispensable database relevant functions, e.g. ACID support.

The Enhancing Approach. In contrast to the extending approach the enhancing approach is much cheaper, since already existing access methods are augmented to support a broader range of data. The code of the access methods to be enhanced has to be adapted in such a way that it gets independent of the indexed data type. As an example, Figure 22a depicts the pseudocode of a B-tree index routine next_node that determines (for a given node and a key value) the node that has to be visited next within a B-tree traversal. This routine only works on key values of type integer, i.e. an index using this routine can only be created on columns of type integer. In order to create B-tree indexes also on columns of type real, the type of the key parameter has to be changed accordingly (Figure 22b).

- a) `CREATE TYPE FracNum (num INTEGER; denom INTEGER)`
- b) `CREATE FUNCTION lessthan (f1 FracNum, f2 FracNum)
RETURNS BOOLEAN
LANGUAGE SQL DETERMINISTIC
BEGIN
 RETURN (f1.num/f1.denom) < (f2.num/f2.denom);
END`

Figure 23: User-defined data type FracNum.

a) Data type for fraction numbers, b) Function lessthan for comparing fraction numbers

In general, to support arbitrary (ordered) types, the B-tree code has to be modified in such a way that it can handle key parameters of any type. Figure 22c depicts a type-independent version of the `next_node` routine. Here, the key type is not determined but has to be instantiated when applying the index. The function `lessthan` has the same functionality as the operator `'<'` for built-in types. If the user defines a new type and wants to use the enhanced B-tree index for columns of this type, he or she has to provide a corresponding `lessthan` function that can handle values of the new type. Alternatively, the built-in operator `'<'` could be overloaded, if the database system used supports this. As a further example, if the user defines a new type `FracNum` for the storage of fraction numbers (consisting of numerator and denominator) in the database system (Figure 23a), he or she has to implement a special version of the function `lessthan` that takes two fraction numbers as parameters (Figure 23b). Whenever the routine `next_node` is called with a key parameter of type `FracNum`, the newly defined version of `lessthan` is used.

In general, to enhance (generalize) an access method in this way, all type-specific operations within the code of the access method have to be identified and isolated so that the user can provide overloaded versions of these operations for his user-defined types. It is necessary to note that not every access method is appropriate for every data type. B-trees e. g. can only be used for types with a linear ordering. In contrast, R-trees are designed to support access to spatially extended and to multi-dimensional data. Depending on the access method and the predicates to be supported by the index, the user has to implement corresponding operations for new data types. To use an enhanced B-tree index, the user must provide implementations of the usual com-

parison operators ' $<$ ', ' \leq ', ' $>$ ', ' \geq ', and ' $=$ ' for a new data type whereas an R-tree index requires spatial operations like “overlaps”, “contains”, “within”, or “equals”.

A further possibility to enhance existing access methods is to implement functional indexes that give quick access to the results of a function defined on the attributes of a table. The type of the function value has to be supported by the enhanced index.

In conclusion, we identify the following properties of the integrating approach:

- **Implementation:** The implementation of a new AM becomes very sophisticated and tedious if writing transactions have to be supported [Bro 01]. In addition, the code maintenance is a very complex task, as new kernel functionality has to be implemented for any built-in access method. Moreover, the tight integration within the existing kernel source produces a highly platform-dependent solution tailor-made for a specific ORDBMS. The enhancement of pre-existing access methods to support user-defined data types and functional indexes is a straightforward task but does not really augment the functionality of the database server (in the sense of having new ways for query processing).
- **Performance:** The integrating approach potentially delivers the maximal possible performance, if the access method is implemented in a closed environment, and the number of context switches to other components of the database kernel is minimized.
- **Availability:** The implementation requires low-level access to most kernel components. If the target ORDBMS is not distributed as open-source, the affected code and documentation will not be accessible to external database developers.

To sum up, the integrating approach is the method of choice only for a few, well-selected access methods serving the requirements of general database applications. It is not feasible for the implementation of rather specialized access methods.

3.3.2 Generic Approach

To overcome the restrictions of the integrating method, Hellerstein, Naughton and Pfeffer [HNP 95] proposed a generic approach to implement new access methods in an ORDBMS. Their Generalized Search Tree (GiST) has to be built only once into an existing database kernel. The GiST serves as a high-level framework to plug in block-based tree structures with full ACID support (cf. Figure 21c).

As in the previous approaches, the database implementor has to integrate the extensibility framework into the database server regarding all tedious tasks like concur-

rency and recovery. Once implemented, a domain specialist can use the GiST framework to derive new index types for particular applications. In contrast to database toolkits or generators, the index implemented does not have to stop the database server and recompile it every time an index type is added. It is just necessary to implement (overload) a number of predefined functions which define the behavior of keys in the tree. This is quite similar as in the enhanced index approach at first glance. However, while enhanced indexes only support new data types for already existing index structures, it is possible to support completely new query predicates with the GiST framework. B-trees and R-trees are both derivable from the GiST framework, for example. Such derived index types may not be as performant as directly integrated ones but require much less effort to realize.

Many extensions to the GiST framework have been presented, including generic support for concurrency and recovery [KMH 97], and additional interfaces for nearest-neighbor search, ranking, aggregation, and selectivity estimation [Aok 98]. In detail, the GiST approach has the following characteristics:

- **Implementation:** Whereas the implementation of block-based spatial access methods on top of the GiST framework can be done rather easily, the intruding integration of the framework itself remains a very complex task. As an advantage, an access method developed for GiST can basically be employed on any ORDBMS that supports this framework. In contrast to the generic GiST implementation, the specialized functionality of a new access method is therefore platform independent.
- **Performance:** Although the framework induces some overhead, we can still achieve high performance for GiST-based access methods. Kornacker [Kor 99] has shown that they may even outperform built-in index structures by minimizing calls to user-defined functions.
- **Availability:** Due to its complex implementation, the GiST framework is not generally available in present-day systems. To our best knowledge, it has only been implemented in the open-source system PostgreSQL but without concurrent access and write-ahead logging of updates for derived indexes. It is an open question, whether and when a comparable functionality with industrial-strength implementation will be a standard component of major commercial ORDBMSs. In [DKD+ 02][DK 02], it was shown how GiST implemented access methods could be linked to a standard database system. The authors combined Oracle's

Cartridge technology with the GiST-framework so that they were able to index any kind of data stored in an ORDBMS with any kind of balanced trees realized within the GIST framework. Let us note that this approach is not limited to the GiST-framework, but can also be applied to all library approaches discussed in the last section, e.g. the XXL-library could be connected to a standard database system in a similar way.

The GiST concept basically delivers the desired properties to implement spatial access methods. It delegates crucial parts of the implementation to the database vendors. Unfortunately, its full functionality is not available in any major commercial database system at present.

3.3.3 Relational Approach

A natural way to avoid the above obstacles is to map the custom index structure to a relational schema organized by built-in access methods (cf. Figure 21d). Such *relational access methods* are designed to operate on top of a relational query language. They require no extension or modification of the database kernel, and, thus, any off-the-shelf ORDBMS can be employed as it is. We identify the following advantages for the relational approach:

- **Implementation:** As no internal modification or extension to the database server is required, a relational access method can be implemented and maintained with less effort. Substantial parts of the custom access semantics may be expressed by using the declarative DML. Thereby, the implementation exploits the existing functionality of the underlying ORDBMS rather than duplicating basic database services as done in the integrating and generic approaches. Moreover, if we use a standardized DDL and DML like SQL:1999 [SQL 99] to implement the low-level interface of our access method, the resulting code will be platform independent.
- **Performance:** The major challenge in designing a relational access method is to achieve both a high usability and performance. The capability and efficiency of the relational approach was proven for interval data [KPS 00] and 3D spatial data [KPPS 03a].
- **Availability:** By design, a relational access method is supported by any relational database system. It requires the same functionality as an ordinary database user or a relational database application.

By following the relational approach to implement new access methods, we obtain a natural distinction between the basic services of all-purpose database systems and specialized, application-specific extensions. By restricting database accesses to the common SQL interface, custom access methods and query procedures are well-defined on top of the core server components. In addition, a relational access method immediately benefits from any improvement of the ORDBMS infrastructure.

3.4 Basics of Relational Access Methods

The basic idea of relational access methods relies on the exploitation of the built-in functionality of existing database systems. Rather than extending any internal component of the database kernel, a relational access method just uses the native data definition and data manipulation language to process updates and queries on abstract data types. Without loss of generality, we assume that the underlying database system implements the standardized *Structured Query Language* SQL-92 [SQL 92] with common object-relational enhancements in the sense of SQL:1999 [SQL 99], including object types and collections.

3.4.1 Paradigms of Access Methods

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. Its robust and powerful abstraction from block-based secondary storage to the object-relational model can then be fully exploited. This concept also perfectly supports database appliances, i.e. dedicated database machines running the ORDBMS as a specialized operating system [KP 92] [Ora 00]. We add the class of relational access methods as a third paradigm to the known paradigms of access methods for database management systems:

- **Main Memory Access Methods** (Figure 24a). Typical applications of these techniques can be found in main memory databases [DKO+ 85] [GS 92] and in the field of computational geometry [PS 93]. A popular example taken from the latter is the binary *Interval Tree* [Ede 80]. It serves as a basic data structure for plane-sweep algorithms, e.g. to process intersection joins on rectangle sets. Main

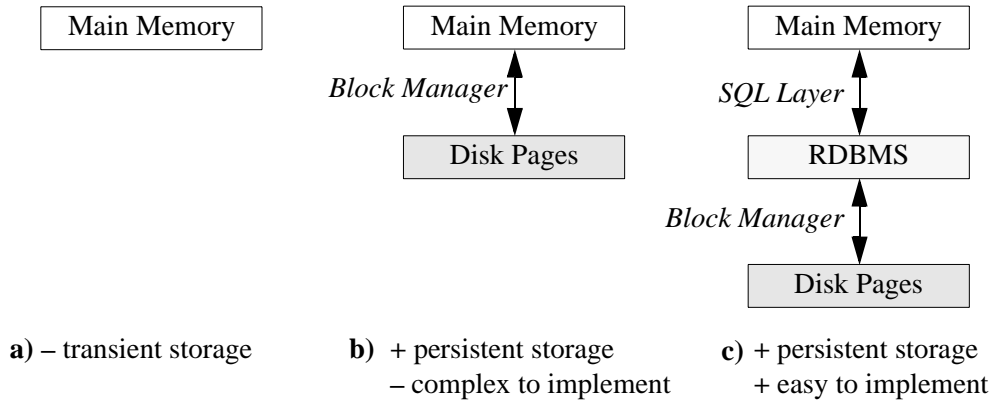


Figure 24: Paradigms and characteristics of access methods.

a) Main memory access methods, b) Block-oriented access methods,
c) Relational access methods

memory structures are not qualified for indexing persistent data, as they disregard the block-oriented access to secondary storage.

- **Block Oriented Access Methods** (Figure 24b). These structures are designed to efficiently support the block-oriented I/O from and to external storage and are well suited to manage large amounts of persistent data. The *External Memory Interval Tree* [AV 96] is an example for the optimal externalization of a main memory access method. Its analytic optimality is achieved by adapting the fanout of the Interval Tree to the disk block size. In the absence of a generalized search tree framework [HNP 95], the implementation of such specialized storage structures into existing database systems, along with custom concurrency control and recovery services, is very complex, and furthermore, requires intrusive modifications of the database kernel [RMF+ 00].
- **Relational Access Methods** (Figure 24c). In contrast, relational access methods are designed to operate on relations rather than on dedicated disk blocks. The persistent storage and block-oriented management of the relations is delegated to the underlying database server. Therefore, the robust functionality of the database kernel including concurrent transactions and recovery can potentially be reused. A primary clustering index can be achieved by also delegating the clustering to the ORDBMS. For this, the payload data has to be included into the index relations and the clustering has to be enabled by organizing these tables in a cluster or as index-organized tables [SDF+ 00].

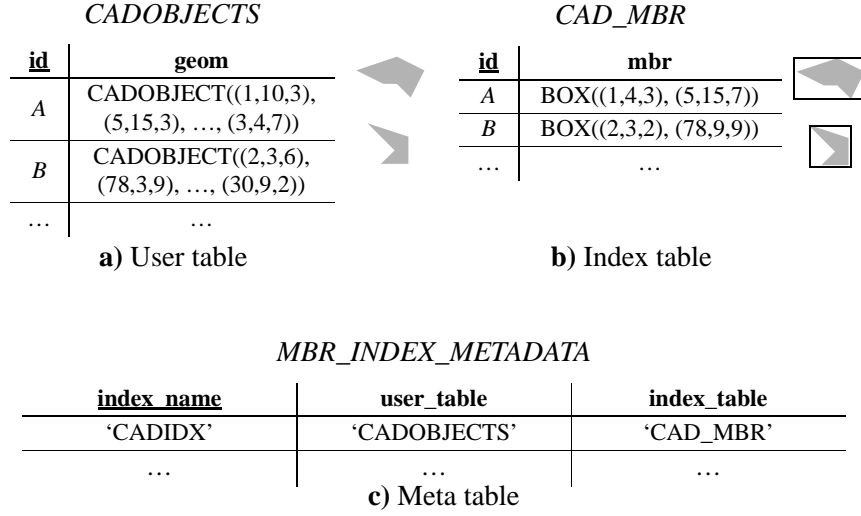


Figure 25: The *MBR-List*, a simple example for a relational access method.

3.4.2 Relational Storage of Index Data

In the remainder of this section, we will discuss the basic properties of relational access methods with respect to the storage of index data, query processing and the overhead for transaction semantics, concurrency control, and recovery services. We start with a basic definition:

Definition 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:

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

To illustrate the concept of relational access methods, Figure 25 presents the minimum bounding rectangle list (*MBR-List*), a very simple example for indexing three-dimensional CAD objects. The user table is given by the object-relational table *CADOBJECTS* (Figure 25a), comprising attributes for the geometry (*geom*) and the

object identifier (*id*). Any spatial query can already be evaluated by sequentially scanning this user table. In order to speed up spatial selections, we decide to define an MBR-List *CADIDX* on the user table. Thereby, an index table is created and populated (Figure 25b), assigning the minimum bounding rectangle (*MBR*) of each CAD object to the foreign key *id*. Thus, the index table stores information purely derived from the user table. All schema objects belonging to the relational index, in particular the name of the index table, and other index parameters are stored in a global meta table (Figure 25c).

In order to support queries on the index tables, a relational access method can employ any built-in secondary indexes, including hash indexes, B^+ -trees, and bitmap indexes. Alternatively, index tables may be clustered by appropriate primary indexes. Consequently, the relational access method and the database system cooperate to maintain and retrieve the index data [DDSS 95]. This basic approach of relational indexing has already been applied in many existing solutions, including *Linear Quadrees* [TH 81][RS 99][FFS 00] and *Relational R-trees* [RRSB 99] for spatial databases, *Relational X-trees* [BBKM 99] for high-dimensional nearest-neighbor search, or inverted indexes for information retrieval on text documents [DDSS 95].

3.5 Operations on Relational Access Methods

In the strict sense of the Definition 1, the procedural code of an arbitrary block-oriented storage structure can immediately be transformed to a relational access method by replacing each invocation of the underlying block manager by an SQL-based DML operation¹. Thus, the original procedural style of an index operation remains unchanged, whereas its I/O requests are now executed by a fully-fledged RDBMS. The object-relational database server is thereby reduced to a plain block manager. In consequence, only a fraction of the existing functionality of the underlying database server is exploited. In this section, we define operations on relational access methods which maximize the architecture-awareness postulated in [JS 99]. This can be achieved by using declarative operations.

¹ E.g. we replace “blocks.get(*block_id*)” by “select * from blocks where id = :*block_id*”.

3.5.1 Cursor-Bound Operations

In order to guarantee a better exploitation of the database infrastructure, we have to restrict the possible number of DML operations submitted from a procedural environment:

Definition 2 (*Cursor-Bound Operation*).

A query or update operation on a relational access method is termed *cursor-bound*, iff the corresponding I/O requests on the index data can be performed by submitting $O(1)$ DML statements, i.e. by sequentially and concurrently opening in total $O(1)$ cursors provided by the underlying RDBMS.

Cursor-bound operations on relational access methods are largely bound to the declarative DML engine of the underlying RDBMS rather than to user-defined opaque code. Thus, the database server gains the responsibility for significant parts of the query and update semantics. Advantages of this approach include:

- **Declarative Semantics:** Large parts of a cursor-bound operation are expressed by using declarative SQL. By minimizing the procedural part and maximizing the declarative part of an operation, the formal verification of the semantics is simplified if we can rely on the given implementation of SQL to be sound and complete.
- **Query Optimization:** Whereas the database engine optimizes the execution of single, closed-form DML statements, a joint execution of multiple, independently submitted queries is very difficult to achieve [Sel 88][CD 98][BEKS 00]. By using only a constant number of cursors, the RDBMS captures significant parts of the operational semantics at once. In particular, complex I/O operations including external sorting, duplicate elimination or grouping should be processed by the database engine, and not by a user-defined procedure.
- **Cursor Minimization:** The CPU cost of opening a variable number of cursors may become very high. For typical applications, the resulting overhead sums up to 30% of the total processing time [RMF+ 00]. In some experiments, we even reached barrier crossing cost of up to 75% for submitting a variable number of pre-parsed DML statements out of a stored procedure. For cursor-bound operations, the relatively high cost of opening and fetching multiple database cursors remains constant with respect to the complexity of the operation and the database size.

3.5.2 Cursor-Driven Operations

A very interesting case occurs if the potential result of a cursor-bound operation can be retrieved as the immediate output of a *single* cursor provided by the DBMS. Thus, the semantics is revealed to the database server at once in its full completeness:

Definition 3 (*Cursor-Driven Operation*).

A cursor-bound operation on a relational access method is called *cursor-driven*, iff it can be divided into two consecutive phases:

- (i) *Procedural phase*: In the first phase, index parameters are read from the meta tables. Query specifications are retrieved and data structures required for the actual query execution may be prepared by user-defined procedures and functions. Additional DML operations on user data or index data are not permitted.
- (ii) *Declarative phase*: In the second phase, only a single DML statement is submitted to the ORDBMS, yielding a cursor on the final results of the index scan which requires no post-processing by user-defined procedures or functions.

Note that any cursor-driven operation is also cursor-bound, while all I/O requests on the index data are driven by a single declarative DML statement. The major advantage of cursor-driven operations is their smart integration into larger execution plans. After the completion of the procedural phase, the single DML statement can be executed with arbitrary groupings and aggregations, supplemented with additional predicates, or serve as a row source for joins. Furthermore, the integration into extensible indexing frameworks is facilitated, as the cursor opened in the declarative phase can be simply pipelined to the index scan routine. Note that the ability to implement cursor-bound and cursor-driven operations heavily relies on the expressive power of the underlying SQL interface, including the availability of recursive queries [Lib 01].

The single DML statement submitted in the declarative phase may contain user-defined functions. The CPU cost of cursor-driven operations is significantly reduced, if the number of barrier crossings due to calls to user-defined functions is minimized [Kor 99]. We can achieve this by preprocessing any required transformation, e.g. of a query specification, in the procedural phase and by bulk-binding the prepared data to the query statement with the help of transient collections. If such data structures become very large, a trade-off has to be achieved between the minimization of barrier crossings and the main-memory footprint of concurrent

```
SELECT id FROM CADOBJECTS
WHERE intersect (geom, BOX((0,0,0),(100,100,100)));
```

a) Box volume query on the user table

```
SELECT usr.id AS id FROM CADOBJECTS usr, CAD_MBR idx
WHERE intersect (idx.mbr, BOX((0,0,0),(100,100,100)))
AND idx.id = usr.id
AND intersect (usr.geom, BOX((0,0,0),(100,100,100)));
```

b) Box volume query using the relational index as primary filter

```
SELECT id FROM CAD_TYPE
WHERE type = 'ENGINE'
AND id IN (
  SELECT usr.id FROM CADOBJECTS usr, CAD_MBR idx
  WHERE intersect (idx.mbr, BOX((0,0,0),(100,100,100)))
  AND idx.id = usr.id
  AND intersect (usr.geom, BOX((0,0,0),(100,100,100)));
```

c) Index-supported box volume subquery

Figure 26: Box queries on CAD data.

sessions [Pfe 01]. Splitting a single query into multiple cursor-driven operations can then be beneficial.

To pick up the *MBR-List* example of the previous section, Figure 26a shows a simple box query on the database of three-dimensional CAD objects, testing the exact geometry of each stored polygon for intersection with the query rectangle. In order to use the relational index as primary filter, the query has to be rewritten into the form of Figure 26b. An efficient execution plan for the rewritten query may first check the intersection with the stored bounding boxes, and refine the result by performing the equijoin with the *CADOBJECTS* table. Note that the box query is a cursor-driven operation on the MBR-List, having an empty procedural phase. Therefore, the index-supported query can be easily embedded into a larger context as shown in Figure 26c. Already this small example shows that an object-relational wrapping of relational access methods is essential to control redundant data in the index tables and to avoid manual query rewriting. The usage of an extensible indexing framework preserves the physical independence of DML operations and enables the usual query optimization.

3.6 Navigational Scheme of Index Tables

As an immediate result of the relational storage of index data and meta data, a relational index is subject to the built-in transaction semantics, concurrency control, and recovery services of the underlying database system. In the following two sections, we discuss the effectiveness and performance provided by the built-in services of the ORDBMS on relational access methods. For that purpose, we identify two generic schemes for the relational storage of index data, the *navigational* scheme and the *direct* scheme. For each of these schemes we exemplarily introduce appropriate index structures and show how we can process intersection queries on top of these index structures. Let us start with the navigational scheme.

Definition 4 (*Navigational Scheme*).

Let $P = (T, R_1, \dots, R_n)$ be a relational access method on a data scheme T and index schemes R_1, \dots, R_n . We call P *navigational* $\Leftrightarrow (\exists t \subseteq T) (\exists r_i \subseteq R_i, 1 \leq i \leq n)$: at least one $p \in r_i$ is associated with rows $\{\tau_1, \dots, \tau_m\} \subseteq t$ and $m > 1$.

Therefore, a row in an index table of a navigational index may logically represent many objects stored in the user table. This is typically the case for hierarchical structures that are mapped to a relational schema. Consequently, an index table contains data that is recursively traversed at query time in order to determine the resulting tuples. Examples for the navigational scheme include the *Oracle Spatial R-tree* [RRSB 99] and the *Relational X-tree* [BBKM 99] which store the nodes of a tree directory in a flat table. To implement a navigational query as a cursor-bound operation, a recursive version of SQL like SQL:1999 [SQL 99] [EM 99] is required.

Although the navigational scheme offers a straightforward way to simulate any hierarchical index structure on top of a relational data model, it suffers from the fact that navigational data is locked like user data. As two-phase locking on index tables is too restrictive, the possible level of concurrency is unnecessarily decreased. For example, uncommitted node splits in a hierarchical directory may lock entire subtrees against concurrent updates. Built-in indexes solve this problem by committing structural modifications separately from content changes [KB 95]. Unfortunately, this approach is not feasible on the SQL layer without braking up the user transaction. A similar overhead exists with logging, as atomic actions on navigational data, e.g. node splits, are not required to be rolled back in order to keep the index tables consis-

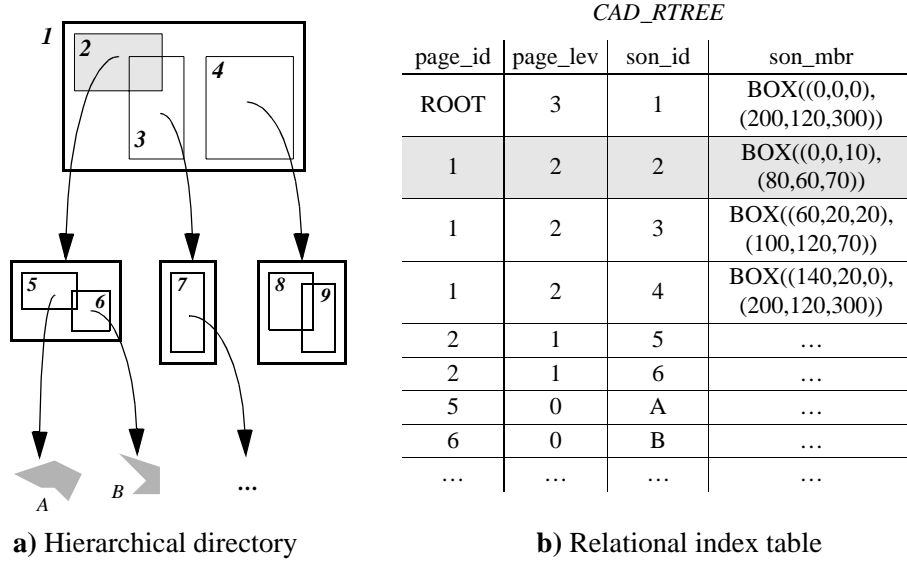


Figure 27: Relational mapping of an R-tree directory.

tent with the data table. Therefore, relational access methods implementing the navigational scheme are only well suited for read-only or single-user environments.

3.6.1 RR-tree – An Example for the Navigational Scheme

We illustrate the properties and drawbacks of the navigational scheme by the example of *Relational R-trees*, like they have been used by the Oracle developers Ravi Kanth et al. [RRSB 99]. Figure 27 depicts a hierarchical R-tree along with a possible relational mapping (*page_id*, *page_lev*, *son_id*, *son_mbr*). The column *page_id* contains the logical page identifier, while *page_lev* denotes its level in the tree. Thereby, 0 marks the level of the data objects, and 1 marks the leaf level of the directory. The attribute *son_id* contains the *page_id* of the connected entry, while *son_mbr* stores its minimum bounding box. Thus, *page_id* and *son_id* together comprise the primary key. In our example, the logical page 2 represents a partition of the data space which contains the CAD objects A and B. The corresponding index row (1, 2, 2, ...) is therefore logically associated with the rows (A, ...) and (B, ...) in the user table *CADOBJECTS* (cf. Figure 25). Thus, the Relational R-tree implements the navigational scheme of relational access methods.

The severe overhead of the navigational scheme already becomes obvious if a transaction inserts a new CAD object, and subsequently enlarges the bounding box of

```

WITH RECURSIVE TREE_TRAVERSAL (page_lev, son_id, son_mbr) AS (
    SELECT page_lev, son_id, son_mbr FROM CAD_RTREE
    WHERE page_id = ROOT
    UNION ALL
    SELECT next.page_lev, next.son_id, next.son_mbr
    FROM TREE_TRAVERSAL prior, CAD_RTREE next
    WHERE intersect (prior.page_mbr, BOX((0,0,0),(100,100,100)))
    AND prior.son_id = next.page_id
)
SELECT son_id AS id
FROM TREE_TRAVERSAL
WHERE page_lev = 0;
    
```

a) Recursive window query on a Relational R-tree using SQL:1999

```

SELECT son_id AS id FROM CAD_RTREE
WHERE page_lev = 0
START WITH page_id = ROOT
CONNECT BY
    intersect (PRIOR son_mbr, BOX((0,0,0),(100,100,100)))
    AND PRIOR son_id = page_id;
    
```

b) Recursive window query on a Relational R-tree using Oracle SQL

Figure 28: Cursor-driven window query on a Relational R-tree.

a node, e.g. of the root node. Due to the common two-phase locking, this transaction will hold an exclusive lock on the row (*ROOT*, 3, 1, ...) until commit or rollback. During this time, no concurrent transaction can insert CAD objects that induce an enlargement of the root region. The database server has to guarantee non-blocking reads [Ora 99c] to support at least concurrent queries on the Relational R-tree index.

To support the navigation through the R-tree table at query time, a built-in index can be created on the *page_id* column. Alternatively, the schema can be transformed to NF² (non-first normal form), where *page_id* alone represents the primary key and a collection of (*son_id*, *son_mbr*) pairs is stored with each row. A cursor-driven primary filter for a window query using recursive SQL is shown in Figure 28. We expect that future implementations of the SQL:1999 statement yield a depth-first traversal which is already hard-wired into the existing CONNECT BY clause of the Oracle server. The effectiveness of cursor-driven operations is illustrated by the fact that the depicted statements already comprise the complete, pipelined query processing on

the R-tree index. If the low concurrency of the Relational R-tree is acceptable, the relational mapping opens up a wide range of potential improvements. We have developed and evaluated various extensions to the presented concept [Str 04][KKP+ 03][KKPR 04d]:

- **Variable Fanout.** Due to the relational mapping, we are basically free to allow an individual fanout for each tree node. Similar to the concept of supernodes for high-dimensional indexing [BKK 96], larger nodes could be easily allocated, e.g. if the contained geometries show a very high overlap or are almost equal. Thus, splitting such pages would not improve the spatial clustering. Instead, page splits could be triggered by measuring the clustering quality with a proximity measure similar to [KF 92]. Especially for CAD databases, where many variants of the same parts occupy almost identical regions of the data space, this approach can be beneficial.
- **Page Clustering.** In order to achieve a good clustering among the entries of each tree node, a built-in primary index can be defined on the *page_id* column. For bulk-loads of Relational R-trees, the clustering can be further improved by carefully choosing the page identifiers: For instance, by assigning linearly ordered *page_ids* corresponding to a breadth-first traversal of the tree, a sibling clustering of nodes [KC 98] can be very easily achieved.
- **Positive Pruning.** By ordering the *page_ids* according to a depth-first tree traversal, a hierarchical clustering of the R-tree nodes is materialized in the primary index. In consequence, the page identifiers of any subtree form a consecutive range. Similarly, if the leaf pages are hierarchically clustered in a separate B^+ -tree, a single range query on the *page_id* column yields a blocked output of all data objects stored in any arbitrary subtree of the R-tree directory. Thus, the recursive tree traversal below a node completely covered by the query region can be replaced by an efficient range scan on the leaf table. Consequently, the tree traversal is not only pruned for all-negative nodes (if no intersection of the node region with the query region is detected), but also for all-positives (the node region is completely covered by the query region). Moreover, heuristics to prune already largely covered nodes can also be very beneficial (cf. Chapter 5).

3.7 Direct Scheme of Index Tables

Definition 5 (*Direct Scheme*).

Let $P = (T, R_1, \dots, R_n)$ be a relational access method on a data scheme T and index schemes R_1, \dots, R_n . We call P *direct* $\Leftrightarrow (\forall t \subseteq T) (\forall r_i \subseteq R_i, 1 \leq i \leq n)$: each $p \in r_i$ is associated with a single row $\tau \in t$.

In consequence, for a relational access method of the direct scheme, each row in the user table is directly mapped to a set of rows in the index tables. Inversely, each row in an index table exclusively belongs to a single row in the user table. In order to support queries, the index table is organized by a built-in index, e.g. a B^+ -tree. Examples for the direct scheme include our *MBR-List* (cf. Figure 25), the *Linear Quadtree* [Sam 90b] and the *Relational Interval Tree* [KPS 00].

The drawbacks of the navigational scheme with respect to concurrency control and recovery are not shared by the direct scheme, as row-based locking and logging on the index tables can be performed on the granularity of single rows in the user tables. For example, an update of a single row r in the user table requires only the synchronization of index rows exclusively assigned to r . As the acquired locks are restricted to r and its exclusive entries in the index tables, they do not unnecessarily block concurrent operations on other user rows. In contrast to navigational indexes, the direct scheme inherits the high concurrency and efficient recovery of built-in tables and indexes.

3.7.1 RQ-tree – An Example for the Direct Scheme

A paradigmatic example for a spatial access method implementing the direct scheme is the *Linear Quadtree* [Sam 90b]. Several variants of this well-known concept have been proposed for stand-alone balanced trees [TH 81][OM 84][Bay 96], for object-oriented database systems [Ore 86][OM 88][GR 94], and as relational access methods [Wan 91][IBM 98][Ora 99b][RS 99][FFS 00]. In this subsection, we present the basic idea of the Relational Quadtree, called RQ-tree throughout this thesis, according to the in-depth discussion of Freytag, Flasz and Stillger [FFS 00].

The RQ-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 [Güt 94]. The corresponding index representation of a spatial object comprises a set

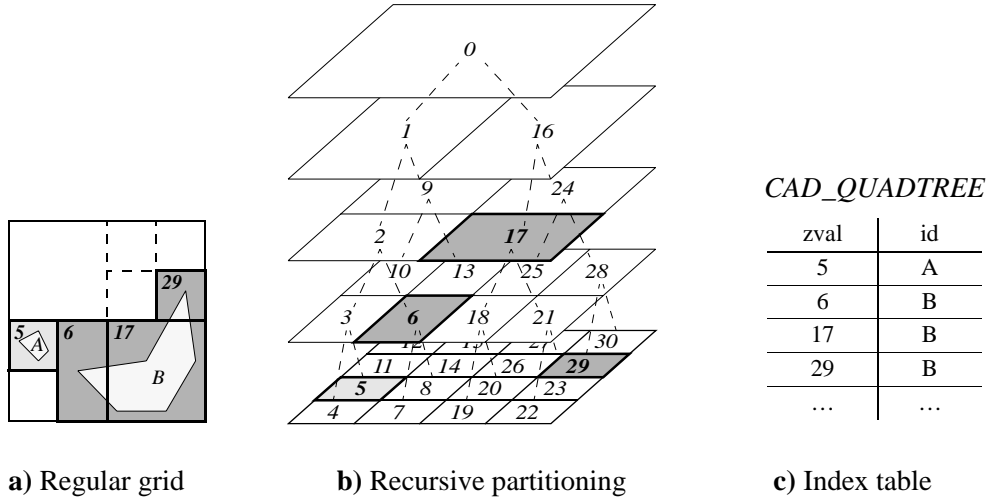


Figure 29: Relational mapping of a Linear Quadtree.

of *Z-tiles* which is computed by recursively bipartitioning 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 29 depicts some *Z-tiles* on a two-dimensional grid¹ 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. This relational mapping implements the direct scheme, as each row in the index table exclusively belongs to a single data object. The linear ordering positions each *Z-tile* of an object on its own row in the index table. Thus, if a specific row in the user table *CAD* is updated, e.g. (B, \dots) , only the rows $(6, B)$, $(17, B)$, and $(29, B)$ in the index table are affected, causing no problems with respect to the native two-phase locking.

In order to process spatial selection on the RQ-tree, the query region is also required to be decomposed to a set of *Z-tiles*. We call the corresponding function *ZDecompose*. 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 [FFS 00]. In the example of Figure 29, we obtain $ZLowerHull(17) = [17, 23]$. In a similar way, we

¹ We used a two-dimensional grid, instead of a three dimensional one for representation reasons.

```

SELECT DISTINCT idx.id                                //select data object
FROM CAD_QUADTREE idx,
      TABLE(ZDecompose(BOX((0,0,0),(100,100,100)))) tiles,
      TABLE(ZUpperHull(tiles.zval)) uh
WHERE (idx.zval BETWEEN tiles.zval AND ZHi(tiles.zval))
      OR (idx.zval = uh.zval);

```

Figure 30: Cursor-driven window query for the RQ-tree.

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. Figure 30 depicts the complete cursor-driven window query on an instance of the RQ-tree using SQL:1999. Alternatively, the transient rowsets generated by the functions $ZDecompose$ and $ZUpperHull$ can be precomputed in the procedural phase for all Z-tiles of the query box and passed to the SQL layer in one step by using bind variables. This approach reduces the overhead of barrier crossings between the declarative and procedural environments to a minimum.

3.7.2 RI-tree - Another Example for the Direct Scheme

The Relational Interval Tree (RI-tree) [KPS 00] is an application of extensible indexing for interval data. Based on the relational model, intervals can be stored, updated and queried with an optimal complexity. In this section, we briefly review the basic concepts of the RI-tree and introduce interval sequences as stored objects and queries. After discussing a naive application that simply considers an interval sequence to be a set of independent entities, we present an optimized version that exploits the connection between the elements of an interval sequence [KPS 01].

The Relational Interval Tree. The RI-tree strictly follows the paradigm of relational storage structures since its implementation is restricted to (procedural) SQL and does not assume any lower level interfaces. In particular, the built-in index structures of a DBMS are used as they are, and no intrusive augmentations or modifications of the database kernel are required.

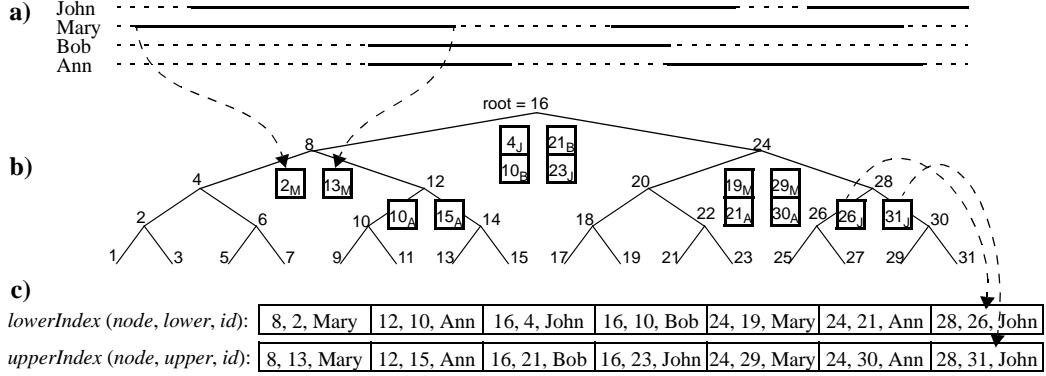


Figure 31: The Relational Interval Tree.

- a) Four sample interval sequences, b) The virtual backbone positions the intervals,
c) Resulting relational indexes

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 $[1 \dots 2^h - 1]$ of potential interval bounds. Traversals are performed purely arithmetically by starting at the root value 2^{h-1} and proceeding in positive or negative steps of decreasing length 2^{h-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 2^{h-1} is stored persistently in a metadata tuple. For the relational storage of intervals, the nodes of the tree are used as artificial key values: Each interval i is assigned to a fork node, i.e. the first node contained in i when descending the tree from the root node down to the interval location.

An instance of the RI-tree consists of two relational indexes which in an extensible indexing environment are at best managed as index-organized tables. The indexes then obey the relational schema $lowerIndex(node, lower, id)$ and $upperIndex(node, upper, id)$ and store the artificial fork node value $node$, the bounds $lower$ and $upper$ and the id of each interval. Any interval is represented by exactly one entry in each of the two indexes 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 labelling each associated interval with the sequence identifier. Figure 31 illustrates the relational interval tree by an example.

Interval Query Processing. To minimize barrier crossings between the procedural runtime environment and the declarative SQL layer, an interval intersection query ($lower, upper$) is processed in two steps. In the procedural query preparation step, range queries are collected in two transient tables, *leftNodes* and *rightNodes*, which are obtained 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 to the left of *lower* are collected in *leftNodes* since they may contain intervals which overlap *lower*. Analogously, nodes to the 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 range since their fork node value is contained in the query interval and, therefore, are reported without any further comparison by a so-called *inner query*. The query preparation procedure is purely main memory-based and, thus, yields no I/O operations.

In the second step, i.e. the declarative query processing, the transient tables are joined with the relational indexes *upperIndex* and *lowerIndex* by a single, three-fold SQL statement shown in Figure 32. The upper bound of each interval registered at nodes in *leftNodes* is checked against *lower*, and the lower bounds of intervals from *rightNodes* are checked against *upper*. We call the corresponding queries *left queries* and *right queries*, respectively. The *inner query* corresponds to a simple range scan over the nodes within (*lower*, *upper*). If b denotes the average number of index entries per disk block, the SQL query yields $O(h \cdot \log_b n + r/b)$ 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. Furthermore, output from the relational indexes is fully blocked for each join partner.

```
SELECT id FROM upperIndex i, :leftNodes left
  WHERE i.node = left.node AND i.upper >= :lower
UNION ALL
SELECT id FROM lowerIndex i, :rightNodes right
  WHERE i.node = right.node AND i.lower <= :upper
UNION ALL
SELECT id FROM lowerIndex i // or upperIndex i
  WHERE i.node BETWEEN :lower AND :upper;
```

Figure 32: SQL statement for a single query interval with bind variables.
Bind variables: *leftNodes*, *rightNodes*, *lower*, *upper*.

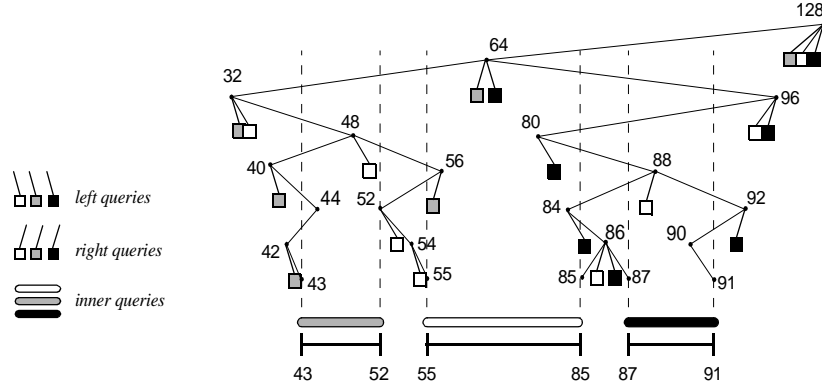


Figure 33: Naive query processing for an interval sequence.

Interval Sequence Intersections. The naive way to process interval sequence intersections on an RI-tree is to perform independent queries for each of the intervals. As an example, let us consider the interval sequence $\langle (43, 52), (55, 85), (87, 91) \rangle$. Figure 33 illustrates the resulting 24 queries for the three query intervals which we encode by the shadings white, gray and black for further reference. The traversed paths of the target RI-tree of height 8 are depicted, and the numbers denote the node values, e.g. 128 for the root of the virtual backbone. In the example, the 7 gray queries are generated for the first interval (43, 52), the 9 white queries for (55, 85), and the 8 black queries for (87, 91). From the total of 24 queries, 11 are left queries, 10 are right queries, and 3 are inner queries.

Gap Optimization for Interval Sequences. In this subsection we sketch a very efficient way to process boolean intersection queries. A naive approach for this kind of queries would consider each interval of a spatial interval sequence individually. However, this approach disregards the important fact that the individual intervals of an interval sequence all 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 which have to be eliminated. The basic idea of gap optimization is to avoid the generation of redundant queries, rather than to discard the respective queries after their generation.

In the example, the root node (128) is queried by three right queries. An interval registered at the root node is reported three times if its lower bound is less or equal to

52, and twice if its lower bound is greater than 52 but not greater than 85. The right query of the rightmost (black) interval suffices to report all resulting intervals from node 128, and discarding the gray and the white query prevents the generation of duplicates without yielding false dismissals.

In [KPS 01], it was shown that for a sorted interval sequence $q = \langle q_1, \dots, q_n \rangle$ with intervals $q_i = (lower_i, upper_i)$, the result of an intersection query is complete if for each q_i , query generation is restricted to nodes n with $upper_{i-1} < n < lower_{i+1}$ where $upper_0 = -\infty$ and $lower_{n+1} = \infty$. The optimized RI-tree is based on this fact together with an approach to integrate the inner queries into the set of left queries [KPS 01]. If we apply these two optimizations to the example in Figure 33 we get only 9 queries instead of the original 24 queries without producing false dismissals.

The optimized RI-tree leads to an average speed-up factor of 2.9 compared to the naive RI-tree, and outperforms competing methods by factors of up to 4.6 (Linear Quadtree) and 58.3 (Relational R-tree) for query response time [KPS 01].

Similar optimizations are possible for the RQ-tree by eliminating duplicates from the upper hulls resulting from different query tiles of a given query sequence. Let us note, that for both index structures these optimizations are only applicable to boolean intersection queries, but fail to support ranked intersection queries.

3.8 Industrial Application

In mechanical engineering, three-dimensional CAD is employed throughout the entire development process. From the early design phase to the serial production of vehicles or airplanes, thousands to millions of CAD files and associated documents are generated. Recently, a new class of CAD tools has emerged to support virtual engineering on this data, i.e. the evaluation of product characteristics without building a physical prototype. Typical applications include the digital mockup [BKP 98] or haptic rendering of product configurations [MPT 99].

EDM systems organize the huge underlying CAD databases by means of hierarchical product structures. Thus, structural queries such as “retrieve all documents that refer to the current version of the braking system” are efficiently supported. Virtual engineering, however, requires access to the product data by geometric predicates, such as “find all parts in the immediate spatial neighborhood of the disk brake” (cf. Figure 3). Unfortunately, spatial queries are not supported efficiently by common EDM systems. In this section, we shortly present the DIVE (Database

Integration for Virtual Engineering) architecture [KMPS 01a] [KMPS 01b][Pöt 01], a proposal to embed virtual engineering into the existing EDM infrastructure of an enterprise. Thereby, we focus on the integration of interactive spatial data management into off-the-shelf object-relational database systems.

3.8.1 Spatial Data Management

The geometry of a part occupies a specific region in the product space. By using this region as a spatial key, related documents such as native CAD files, VRML scenes or production plans may be spatially referenced. The key challenges in developing a robust and dynamic database layer for virtual engineering have been (1) to store spatial CAD data in a conventional relational database system, (2) to enable the efficient processing of the required geometric query predicates and (3) to determine efficient execution plans for queries consisting of geometric and structural predicates.

Only few spatial access methods have been designed to operate on the object-relational data model without any intrusive modifications or additions to the physical kernel of the database server. As outlined in Section 3.7.2, the RI-tree is a light-weight access method that efficiently manages extended data on top of any relational database system while fully supporting the built-in transaction semantics and recovery services. In [KPS 01], it was shown that its spatial application outperforms competing techniques with respect to usability and performance. Therefore the RI-tree is used as a spatial engine for the DIVE system.

The original parts are converted from the various native CAD formats to triangulated facets in VRML. Furthermore, 3D scan conversion on a regular grid is used to voxelize these geometries. To enable the generated voxel set to be used as a spatial key, it is transformed to an interval sequence on a space-filling curve and stored in the RI-tree. The redundancy and accuracy of each interval sequence can be controlled individually by size-bound or error-bound approximation (cf. Section 2.1).

3.8.2 Query Processing

The DIVE server maps geometric query predicates to region queries on the indexed data space. Our multi-step query processor performs a highly efficient and selective filter step based on the stored interval sequences. The non-spatial remainder

of the query, e.g. structural exclusions, is processed by the EDM system. The current DIVE release contains filters for the following spatial queries:

- **Volume Query:** Determine all spatial objects intersecting a given rectilinear box volume.
- **Collision Query:** Find all spatial objects that intersect an arbitrary query region, e.g. a volume or a surface of a query part.
- **Clearance Query:** Given an arbitrary query region, find all spatial objects within a specified Euclidean distance.

Furthermore, an optional refinement step for the digital mockup to compute intersections on high-accurate triangulated surfaces has been integrated. A part resulting from the previous spatial query may be used as a query object for the next geometric search. Thereby, the user is enabled to spatially browse a huge persistent database at interactive response times. The DIVE server also supports the ranking of query results according to the intersection volume of the query region and the retrieved spatial keys. For digital mockup, this ranking can be refined by computing the shape and length of the intersection segments on the part surfaces. Thus, the attention of the user is immediately guided to the most relevant problems in the current product design.

3.8.3 Efficient Execution Plans

In order to fully integrate the RI-tree into an ORDBMS, we need an appropriate cost model for intersection queries on interval sequences (cf. Chapter 4). The accuracy of the cost model is decisive for the efficiency of the generated execution plans. For queries containing geometric and structural predicates, the system has to decide which predicate has to be carried out first. In general, highly selective predicates should be carried out as early as possible and should be supported by efficient access methods. For both the geometric as well as the structural predicates there exist index supported implementations as well as functional implementations, i.e. without index support.

Figure 34 shows different possibilities in what order the predicates can be evaluated. If the predicates are evaluated consecutively, the index-based implementation of the first predicate can be used, whereas the second one is always based on the functional implementation. If both predicates are carried out in parallel, we can use the index based implementation of both and finally merge the two result sets.

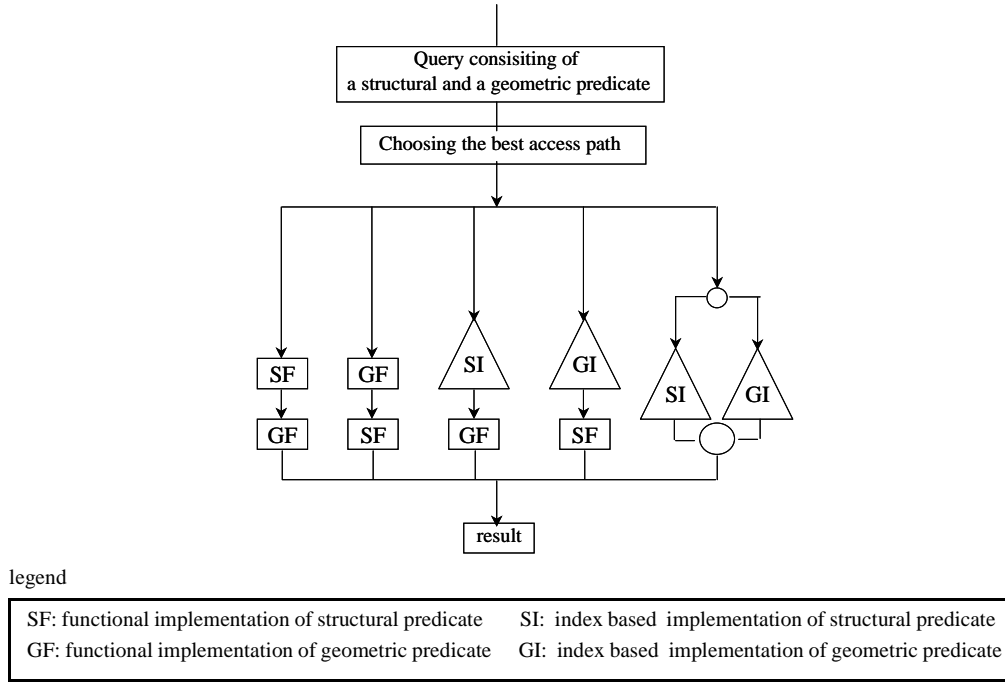


Figure 34: Different execution plans.

Execution plans for queries containing geometric and structural predicates.

For efficient query processing, it is decisive to choose the best possible execution plan. The best-possible plan heavily depends on the selectivity of both predicates. Figure 35 shows for a test data set, how the best plan depends on the selectivity of the predicates. The background shading of each cell symbolizes the best possible execution plan. The value in brackets denotes the percentage deviation of the second-best execution plan to the optimum one. Already the second-best execution plan can be much slower than the best one, not to speak of the other plans. For more detail we refer the interested reader to [Hof 01]. Already this small example points out that an exact selectivity estimation is essential for efficient query processing. In Section 4.5, we will discuss in detail how we can efficiently and accurately estimate the selectivity of spatial queries together with the related I/O cost.

3.8.4 Experimental Evaluation of Query Processing

We have evaluated the DIVE server in collaboration with the Volkswagen AG on real product data. An installation on an Athlon/750 machine with IDE hard drives

		selectivity of the structural predicate (in %)						
		0	1.4	4.6	14.8	25	50	100
selectivity of the geometric predicate (in %)	0	(183)	(15)	(21)	(35)	(44)	(77)	(140)
	1.4	(575)	(11)	(1.1)	(1.1)	(7.9)	(39)	(73)
	4.2	(825)	(3.8)	(2.9)	(1.8)	(11)	(24)	(55)
	8.8	(1375)	(3.4)	(0.6)	(1.3)	(4.5)	(21)	(36)
	32	(5087)	(142)	(6.2)	(6.4)	(4.1)	(0.2)	(6.2)
	64	(9988)	(352)	(105)	(4.8)	(7.4)	(6.3)	(2.1)
	100	(14588)	(538)	(191)	(50)	(17)	(55)	(58)

legend





	functional implementation of structural predicate, index based implementation of geometric predicate
	index based implementation of structural predicate, index based implementation of geometric predicate
	index based implementation of structural predicate, functional implementation of geometric predicate
	functional implementation of structural predicate, functional implementation of geometric predicate

Figure 35: Optimal access paths.

Access paths, depending on the selectivity of the geometric and the structural predicate for a test data set of the Volkswagen AG consisting of 2000 parts.

and a buffer pool of 800 KB performed average volume and collision queries in 0.7 seconds response time on a database containing 11.200 spatial keys (2 GB of compressed VRML data). Due to the logarithmic scale-up of the spatial query processor and an accurate selectivity estimation, interactive response times can still be achieved for much larger databases and complex queries consisting of both geometric and structural information.

3.8.5 System Architecture

Figure 36 presents the three-tier client/server architecture of the DIVE system. The client application runs on a conventional web browser and enables the user to specify spatial and non-spatial query conditions (1). The query evaluation is distributed to the DIVE and EDM servers (2). The DIVE server can be implemented on top of any relational database system, whereas extensible object-relational database systems facilitate the seamless embedding of complex spatial datatypes and operators.

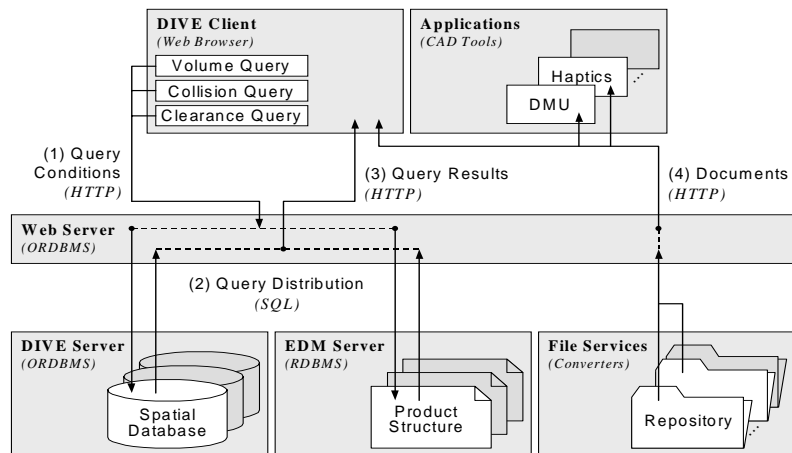


Figure 36: Processing a query on the DIVE system.

The DIVE server has been integrated into Oracle8i by using PL/SQL and Java Stored Procedures. Therefore, the queries are simply submitted in the standard SQL syntax via Oracle's Net8 protocol. After completion of the spatial and structural filter steps and the optional query refinement, the query result is returned to the client as a table of document URLs (3). Finally, the browser may be used to display the contents of the corresponding documents or, alternatively, their content may be downloaded to a specialized application (4).

Chapter 4

A Cost Model for Spatial Intersection Queries

The efficient management of interval sequences represents a core requirement for many temporal and spatial database applications. With the Relational Interval Tree (RI-tree), an efficient access method has been proposed to process intersection queries of spatial objects encoded by interval sequences on top of existing object-relational database systems. This chapter complements that approach by effective and efficient models to estimate the selectivity and the I/O cost of interval sequence intersection queries in order to guide the cost-based optimizer whether and how to include the RI-tree into the execution plan. By design, the models immediately fit to common extensible indexing/optimization frameworks, and their implementations exploit the built-in statistics facilities of the database server. According to our experimental evaluation on an Oracle database, the average relative error of the estimated cost to the actual cost of index scans ranges from 0% to 32%, depending on the resolution of the persistent statistics as well as the size and the structural complexity of the query objects.

4.1 Introduction

After two decades of temporal and spatial index research, the efficient management of one- and multi-dimensional extended objects has become an enabling technology for many novel database applications. The interval, or, more generally, the sequence of intervals, are basic datatypes for temporal and spatial data. Interval sequences are used to handle finite domain constraints [Ram 97] or to represent periods on transaction or valid time dimensions [TCG+ 93]. Typical applications of one-dimensional interval sequences include the temporal tracing of user activity for service providers: a query like “Find all customers who were online last month on any day between 5 and 6 pm” maps to an intersection query of interval sequences on a database storing online periods of all registered users. In general, any time series may be aggregated to an interval sequence, such as periods of “high” stock prices for technical chart analysis. When applied to space-filling curves, interval sequences naturally represent spatially extended objects with even intricate shapes. By expressing spatial region queries as interval sequence intersections, vital operations for two-dimensional GIS and environmental information systems [MP 94] can be supported. Efficient and scalable database solutions are also required for two- and three-dimensional CAD applications to cope with rapidly growing amounts of dynamic data and highly concurrent workflows.

Highly accurate but still efficient selectivity estimation and cost prediction are the fundamentals of effective query optimization. As pointed out in [SJS 01], standard selectivity estimation does not estimate well the result cardinalities of selections having temporal or spatial predicates, and standard built-in methods are not directly suitable for interval intersection queries, in particular. For complex query objects and query predicates, the recent object-relational database servers provide extensible optimization frameworks that come along with the extensible indexing frameworks, in order to complete the seamless integration of user-defined index structures into the declarative DML. As an example for such an extension, we propose a cost model for the RI-tree that fits well to the extensible frameworks by design. Though the RI-tree immediately maps intervals to built-in B^+ -trees, the built-in cost models for B^+ -trees do not estimate well the processing cost since they do not take the particular structure and partitioning of interval data into account.

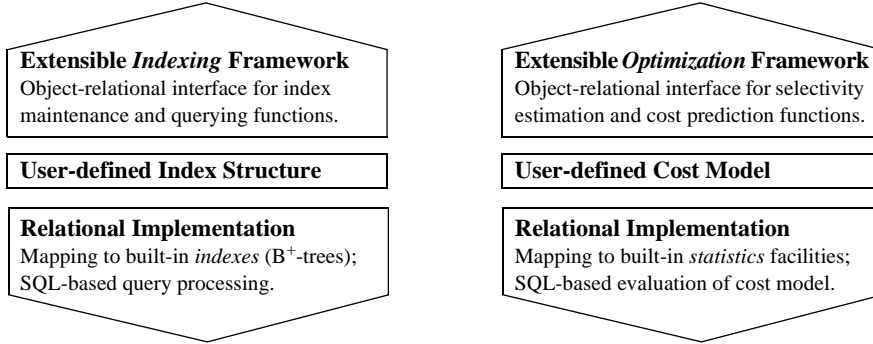


Figure 37: Extensible indexing / optimization frameworks.

Analogous architectures for the object-relational embedding of user-defined index structures and cost models into extensible indexing and optimization frameworks.

Our techniques aim at the collection of statistics, the estimation of selectivity, and the prediction of I/O cost. Thereby, the optimizer of the database system is enabled to place the user-defined index at its optimal position in the query execution plan. According to [BO 99] and [HS 93], such a cost-based approach is preferable to rule-based approaches when referencing user-defined methods as predicates. The two main design aspects for the above mentioned functions are:

Effectiveness. The extensible optimizer uses the selectivity estimation to determine a good join order for complex SQL queries. It then evaluates the available cost models to choose the most efficient access path to the data. The objective is to keep the relative error of selectivity and cost estimations sufficiently small to rank the user-defined index accurately among alternative access methods.

Efficiency. In order to obtain an efficient execution plan for a DML operation, the optimizer framework calls the estimation functions for each contained interval predicate. To reduce the total runtime of query optimization, the execution cost for the estimation functions should be kept minimal. Furthermore, data statistics required for the estimation functions should also be efficiently collected.

The architecture of extensible optimization is analogous to extensible indexing as illustrated in Figure 37. Whereas the new methods are built on top of the relational SQL layer, they are object-relationally embedded by implementing the respective interfaces of the frameworks. In case of our new cost models, we particularly propose methods to estimate the selectivity of a given range query on a database of intervals (function *getSelectivity*) and a method to predict the cost of processing that query

(function *getIndexCost*). In this chapter, we focus on the predicate *overlaps* (= *intersects*) which is considered to be the most important one [GG 98]. Furthermore, we extend the proposed techniques to spatial queries, i.e. interval sequences.

The organization of this chapter follows the requirements of extensible optimization frameworks and proceeds in the following way: First, we sketch in Section 4.2 the related work on selectivity estimation and cost prediction. In Section 4.3, we propose two approaches to estimate the selectivity of intersection queries on interval data. The first approach is based on user-defined histograms, whereas the second one relies on the built-in statistics of standard database systems. Section 4.4 derives a cost model for estimating the I/O cost of a given query on the RI-tree. In Section 4.5, we extend the proposed techniques to interval sequences. After an empirical evaluation of the presented methods in Section 4.6, this chapter is concluded in Section 4.7.

4.2 Related Work

4.2.1 Selectivity Estimation

In order to determine a good estimate for the selectivity of a specific predicate without retrieving the actual results, the predicate has to be evaluated on a sufficiently accurate approximation of the data distribution. The computation of such an approximation is known as one of the most difficult problems, for instance in case of selectivity estimation of extended objects [APR 99]. The many existing approaches fall into three different classes: parametric techniques, sampling, and statistics.

Parametric techniques. *Parametric techniques* approximate the given data by using a standard mathematical distribution. For databases comprising extended objects, many proposals exploit intrinsic characteristics of the stored data, including the usage of the *Correlation Fractal Dimension* on point sets by Belussi and Faloutsos [BF 95] or the *SLED* property of real segment data proposed by Proietti and Faloutsos [PF 99]. A limitation for parametric techniques results from the requirement of a-priori assumptions about the data distribution.

Sampling. In contrast, *sampling* adapts to the actual data distribution by processing a small fraction of the stored tuples. This paradigm has been pursued and evaluated by Lipton, Naughton and Schneider [LNS 90] and by Haas et al. [HNSS 95].

Statistics. *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. For linearly ordered domains, the most commonly used statistics type in commercial database servers are quantiles of the original data. For the selectivity estimation on non-uniform distributions of extended objects, histograms are a common technique. An extensive analysis on different kinds of spatial histograms has been published by Acharya, Poosala and Ramaswamy [APR 99]. 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. In this chapter, we present and evaluate techniques for interval data on both types of statistics, histograms as well as quantiles.

4.2.2 Cost Estimation

A wide range of cost models has been presented in the literature for various index structures for extended objects, including the technique of Kamel and Faloutsos [KF 93] for intersection queries on packed R-trees, or the *REGAL* law for R-tree entries by Proietti and Faloutsos [PF 99]. Recently, cost models have also been extended to handle joins of extended objects and the presence of database buffers as in the proposals of Huang, Jing and Rundensteiner [HJR 97], Leutenegger and Lopez [LL 98], or Theodoridis, Stefanakis and Sellis [TSS 00]. Whereas previous research has mainly concentrated on the design and evaluation of cost models for stand-alone access methods, the following sections develop an approach that can be fully implemented on top of existing object-relational database systems.

4.3 Selectivity Estimation

Accurate estimations of query result sizes are a necessary input for many components of the underlying database system. In particular, the selectivity estimation for an interval intersection query can be used by the built-in optimizer to find an efficient join order and to determine the best available access method [SAC+ 79][Ora 99b]. Selectivity estimation is also required to provide the user with an approximate prediction about the potential execution time of DML statements. In the following, we propose a histogram-based approach ('equi-width histograms') and a quantile-based approach ('equi-count histograms').

4.3.1 Histogram-Based Selectivity Estimation

In order to cope with arbitrary interval distributions, histograms can be employed to capture the data characteristics at any desired resolution. We start by giving the definition of an interval histogram.

Definition 6 (*Interval Histogram*).

Let $D = [1, 2^h - 1]$ be a domain of interval bounds, $h \geq 1$. Let the natural number $v \in \mathbb{IN}$ be the *resolution*, and $\beta_v = (2^h - 1)/v$ the corresponding *bucket size*. Let $b_{i,v} = [1 + (i - 1) \cdot \beta_v, 1 + i \cdot \beta_v)$ denote the *span of bucket i* , $i \in \{1, \dots, v\}$. Let further $I = \{(l, u), l \leq u\} \subseteq D^2$ be a database of intervals. Then, $H(I, v) = (n_1, \dots, n_v) \in \mathbb{IN}^v$ is called the *interval histogram* on I with *resolution v* , iff for all $i \in \{1, \dots, v\}$:

$$n_i = |\{\psi \in I \mid \psi \text{ intersects } b_{i,v}\}|$$

In order to compute an interval histogram on a database I of n intervals, $O(n/b)$ disk blocks have to be touched, assuming a blocked storage of I by a page size b . The computation is performed by standard SQL and wrapped by a stored procedure that complies with the statistics collection interface of the extensible optimization framework (function *getSelectivity*). Based on $H(I, v)$, we compute a selectivity estimate by evaluating the intersection of the query interval with each bucket span $b_{i,v}$ (cf. Figure 38).

Definition 7 (*Histogram-based Selectivity Estimate*).

Given an interval histogram $H(I, v) = (n_1, \dots, n_v)$ with bucket size β , we define the *histogram-based selectivity estimate* $\sigma_I(I, \tau)$, $0 \leq \sigma_I(I, \tau) \leq 1$ for an intersection query $\tau = (l_\tau, u_\tau)$ by the following formula:

$$\sigma_I(I, \tau) = \left[\sum_{i=1}^v \frac{\text{overlap}(\tau, b_{i,v})}{\beta} \cdot n_i \right] \cdot \left[\sum_{i=1}^v n_i \right]^{-1}$$

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 database.

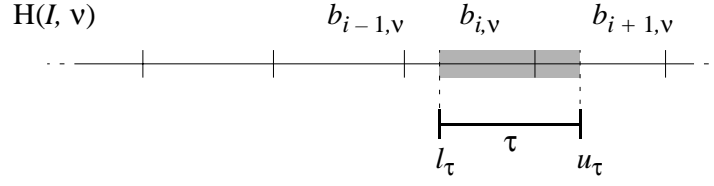


Figure 38: Selectivity estimation on an interval histogram.

In order to support query intervals with a very small duration, the average length of the stored intervals could also be considered for the estimation.

4.3.2 Quantile-Based Selectivity Estimation

Due to the replication of intervals across bucket boundaries, the accuracy of the histogram-based selectivity estimation may deteriorate with longer interval lengths or higher histogram resolutions. In addition, the runtime required for the histogram computation is increased by the cost of barrier-crossings between the declarative environment of the SQL layer and our stored procedure. Fortunately, most ORDBMS comprise efficient built-in functions to compute single-column statistics, particularly for cost-based query optimization. Available optimizer statistics are accessible to the user by the relational data dictionary. The basic idea of our quantile-based selectivity estimation is to exploit these built-in index statistics rather than to add and maintain user-defined histograms. We start with the definition of a quantile vector, the typical statistics type supported by relational database kernels. Then, we describe its application to node values of the RI-tree.

Definition 8 (*Quantile Vector*).

Let (S, \leq) be a totally ordered multi-set. Without loss of generality, let $S = \{s_1, s_2, \dots, s_k\}$ with $s_j \leq s_{j+1}$, $1 \leq j < k$. Then, $Q(S, v) = (q_0, \dots, q_v) \in S^v$ is called a *quantile vector* for S and a *resolution* $v \in \mathbb{IN}$, iff the following conditions hold:

- (i) $q_0 = s_1$
- (ii) $\forall i \in 1, \dots, v: \exists j \in 1, \dots, k: q_i = s_j \wedge \frac{j-1}{k} < \frac{i}{v} \leq \frac{j}{k}$

Definition 9 (*Node Quantiles*).

Let *lowerIndex* be the relational index on $(node, lower, id)$ for an instance T of the RI-tree. Let $N = \pi_{node}(lowerIndex)$ be the projected multi-set of node values. Then, $Q(N, v)$ is called the vector of *node quantiles* on T with *resolution* v .

Based on the node ordering materialized in the *lowerIndex* (or *upperIndex*), the computation of $Q(N, v)$ on an RI-tree storing n intervals has an I/O complexity of $O(n/b)$, where b is the disk block size. By using the node quantiles for an RI-tree index, we get an aggregated view on the locations of the stored intervals. In addition, we may use some knowledge about the one-dimensional durations which is given by the following definition that captures the average distances of the interval bounds to the respective fork node:

Definition 10 (*Average Node Distances*).

Let T be an instance of the RI-tree with *lowerIndex* and *upperIndex* relations. Then, the *average lower distance* $\delta_{lower}(T)$ and the *average upper distance* $\delta_{upper}(T)$ is defined as:

$$(i) \quad \delta_{lower}(T) = \text{avg}(node - lower)$$

$$(ii) \quad \delta_{upper}(T) = \text{avg}(upper - node)$$

The average values δ_{lower} and δ_{upper} are computed with $O(n/b)$ I/O complexity, and if possible, along with the quantile statistics. If the built-in statistics of the hosting database system comprise single-column averages on *node*, *lower*, and *upper*, then δ_{lower} and δ_{upper} can be simply derived from these existing statistics: $\delta_{lower} = \text{avg}(node) - \text{avg}(lower)$ and $\delta_{upper} = \text{avg}(upper) - \text{avg}(node)$. For interval databases with a highly skewed distribution of interval lengths, δ_{lower} and δ_{upper} can be replaced by quantiles on $\pi_{node - lower}(lowerIndex)$ and $\pi_{upper - node}(upperIndex)$.

Our goal is to compute the selectivity estimate in constant time, i.e. independent not only from the cardinality, but also from the granularity of the interval data. Instead of submitting the $O(h = \log_2 root + 1)$ node queries on the RI-tree, we evaluate the quantiles with respect to the span of nodes touched during the processing of a potential interval intersection query.

Definition 11 (*Span of Touched Nodes*).

For a given RI-tree T and an intersection query τ , the range $\theta(T, \tau) = (l_\theta, u_\theta)$ is called the *span of touched nodes*, iff l_θ is the minimal and u_θ is the maximal node on the virtual backbone that is touched while processing the query τ on T .

Lemma 1. Let $D = [1, 2^h - 1]$ be the interval domain covered by an RI-tree T with $root = 2^{h-1}$. For an intersection query $\tau = (l_\tau, u_\tau) \in D$, the span of touched nodes $\theta(T, \tau) = (l_\theta, u_\theta) \in D$ is computed by the following formulas:

- (i) $l_\theta = 2^k, k = \lfloor \log_2(l_\tau) \rfloor$,
- (ii) $u_\theta = 2^h - 2^k, k = \lfloor \log_2(2^h - u_\tau) \rfloor$.

Proof. (i) The leftmost node touched during the arithmetic traversal of the backbone is the last node before we first step into a right subtree. Following the left branch yields a 0-bit, following the right branch yields a 1-bit in the binary representation of the actual node value. Thus, the leftmost node l_θ has exactly one bit set at the first position of a 1-bit in l_τ . (ii) Analogously, the rightmost node u_θ is derived from the first 0-bit in the binary representation of u_τ by a mirrored consideration. ■

We estimate the number of results yielded by the *inner*, *left*, and *right queries* for an intersection query $\tau = (l_\tau, u_\tau)$ based on the node quantiles $Q(N, v) = (q_0, \dots, q_v)$. Figure 39 provides a graphical interpretation of the following calculations: the number of results r_{inner} from the *inner query* can be estimated by evaluating the overlap of τ with the quantiles (analogously to Section 4.3.1):

$$r_{inner} = \sum_{i=1}^v \left(\frac{\text{overlap}(\tau, (q_{i-1}, q_i))}{q_i - q_{i-1}} \cdot \frac{|N|}{v} \right)$$

To estimate the number of results r_{left} retrieved by the *left queries*, we only have to consider quantiles falling into the range $(\text{leftBound}_\tau, l_\tau)$, where $\text{leftBound}_\tau = \max(l_\theta, l_\tau - \delta_{upper}(T))$ and $\theta(T, \tau) = (l_\theta, u_\theta)$:

$$r_{left} = \sum_{i=1}^v \left(\frac{\text{overlap}((\text{leftBound}_\tau, l_\tau), (q_{i-1}, q_i))}{q_i - q_{i-1}} \cdot \frac{|N|}{v} \right)$$

The estimation of the number of results r_{right} of the *right queries* is done analogously to r_{left} . Finally, we define:

Definition 12 (*Quantile-based Selectivity Estimate*).

The quantile-based selectivity estimate $\sigma_N(I, \tau)$ of the intersection query τ on an interval database I is given by

$$\sigma_N(I, \tau) = \frac{r_{left} + r_{inner} + r_{right}}{|N|}$$

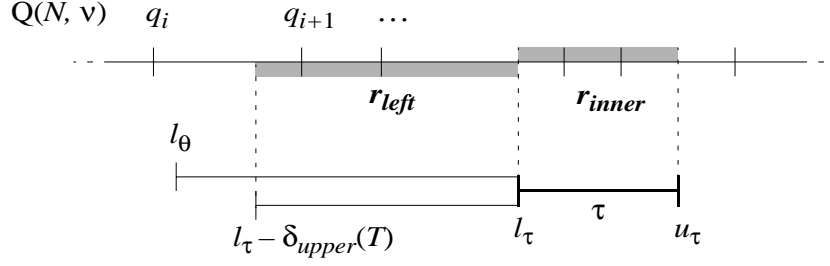


Figure 39: Selectivity estimation on node quantiles.

As desired, the quantile vector is a non-replicating statistics on interval data, and the data sets contributing to the results r_{left} , r_{inner} , and r_{right} are disjoint. In consequence, $0 \leq r_{left} + r_{inner} + r_{right} \leq |N|$ holds and, thus, $0 \leq \sigma_N(I, \tau) \leq 1$.

4.4 Model for I/O Cost

In order to achieve a seamless declarative integration of the Relational Interval Tree into extensible indexing frameworks as provided by modern object-relational database systems, a cost model has to be registered at the extensible optimization framework. In this section, we present a cost model for interval intersection queries on the RI-tree, based on the estimated selectivity and the range queries generated for the underlying B^+ -trees.

We assume the selectivity estimation $\sigma(I, \tau)$ for an intersection query $\tau = (l_\tau, u_\tau)$ on an interval data set I to be determined as shown above. In our derivation of a cost model to estimate the number of touched B^+ -tree blocks for arbitrary intersection queries τ , we use that expected selectivity as input for the estimation of the I/O operations.

Let us recall from Section 3.7.2 that the query preparation step does actually cause no I/O operations since the traversal of the backbone structure is done purely arithmetically, and the generated join partners are managed in main memory. The I/O complexity of $O(h \cdot \log_b n + r/b)$ for an intersection query retrieving r results from an RI-tree of height h comprises components of the following two types:

- First, the directories of the relational indexes (built-in B^+ -trees) have to be traversed in order to navigate on the disk to the first result, if any, for each join partner.

Let us denote this portion of I/O operations by $join_{I/O}$ and let us recall that $join_{I/O} = O(h \cdot \log_b n)$.

- Second, the results for each join partner are reported by scanning contiguous leaf blocks of the relational indexes. We call this portion of I/O operations $output_{I/O}$. Since the output is blocked, i.e. there are no gaps between the answers for a single range query, the complexity $output_{I/O} = O(r/b)$ is guaranteed.

In contrast to the very general complexity analysis, a cost model has to compute actual numbers of I/O operations for specific interval queries. Our model relies on the following two observations:

- In a real user environment with many concurrent queries, substantial parts of the B^+ -directories typically reside in the main memory and can be managed by the built-in LRU-cache of the DBMS [Lom 98]. According to a common assumption, we count two I/O operations for each leaf-block access in order to estimate the number of blocks actually read from disk.
- The transient join partners are processed in increasing order (*left queries*, *inner query*) or decreasing order (*right queries*) with respect to the *node* value in the composite indexes on $(node, upper, id)$ and $(node, lower, id)$, respectively. Due to this ordered access, pages that are read several times during query processing will rarely be displaced from the LRU cache between the accesses. We therefore assume that each leaf page is retrieved only once from secondary storage.

Based on these assumptions, we derive individual formulas for the components $output_{I/O}$ and $join_{I/O}$ in the following.

$output_{I/O}$. For a given RI-Tree T on a set I of intervals, let $L = \text{leaf-blocks}(upperIndex) \approx \text{leaf-blocks}(lowerIndex)$ be the number of leaf blocks in the B^+ -trees, $L = O(n/b)$, and τ be an interval intersection query performed on T . The answers retrieved from $upperIndex$ and from $lowerIndex$ are guaranteed to be disjoint, and we estimate $output_{I/O}(T, \tau)$ as the fraction of L predicted by the selectivity estimate $\sigma(I, \tau)$ on T :

$$output_{I/O}(T, \tau) = \sigma(I, \tau) \cdot L$$

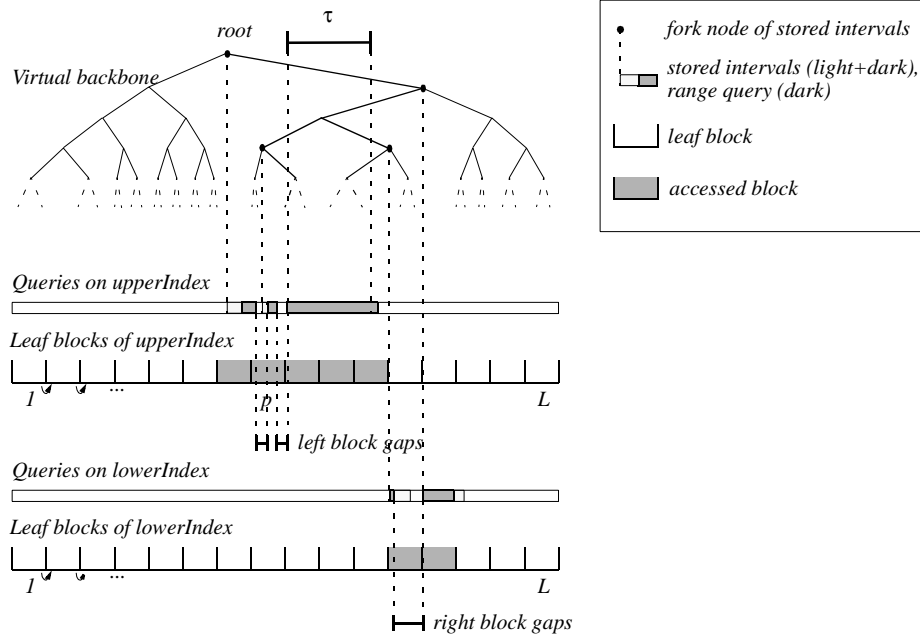


Figure 40: Touched leaf blocks and query gaps for an intersection query τ .

$join_{I/O}$. The formula for $join_{I/O}$ includes the number of leaf block accesses caused by the navigation in the B^+ -tree directories for the join partners. Since the leaf blocks are read from two independent B^+ -tree indexes, we capture the join overhead for the set of *left queries* and *inner queries* on the one hand and for the set of *right queries* on the other hand separately.

Figure 40 provides an illustration for our considerations and depicts the leaf blocks in the *lowerIndex* and *upperIndex* that are read for a query $\tau = (l_\tau, u_\tau)$. Note that the virtual backbone is drawn to the scale of the population in the indexes, and not to the original domain of $D = [1, 2^h - 1]$.

The leaf block p in the *upperIndex*, for example, is touched multiple times during query processing. According to the locality-preserving read schedules for LRU buffers, the multiple accesses to block p count for a single leaf access only. This estimation is complemented by the additional heuristics to count two physical disk accesses for a single leaf block access in order to take care of the I/O caused by traversing the index directory.

An important observation for $join_{I/O}$ is that the results of different join partners in general do not form a contiguous range of entries in the leaf blocks of the indexes. Although the results are blocked for each single *left query*, *inner query*, and *right*

query, there are typically gaps between the blocked result sets of different join partners. In order to model the distribution of gaps, we first determine the gaps between the node values, $NGaps_{left}(\tau)$ and $NGaps_{right}(\tau)$, for a given intersection query τ on an RI-tree T . Then, we derive the expected corresponding gaps between disk blocks, $BGaps_{left}(\tau)$ and $BGaps_{right}(\tau)$.

Estimation of Node Gaps. For the estimation of node gaps, we traverse the virtual backbone on $D = [1, 2^h - 1]$, and we collect the lengths $NGaps_{left}(\tau) = \{\zeta_1, \dots, \zeta_l\}$ of gaps to the left of the query interval τ , i.e. in the range $[1, l_\tau]$ between consecutive nodes touched by the *left* and *inner queries*, and the lengths $NGaps_{right}(\tau) = \{\xi_1, \dots, \xi_r\}$ of gaps to the right of τ , i.e. in the range $[u_\tau, 2^h - 1]$ between the *right queries*, respectively.

Estimation of Block Gaps. Let L_θ be the average number of nodes per leaf block in the span $\theta(T, \tau)$ of touched nodes for τ . We estimate the corresponding block gaps among the range queries for τ by the multi-sets $BGaps_{left}(\tau)$ and $BGaps_{right}(\tau)$ of real numbers:

$$BGaps_{left}(\tau) = \frac{\zeta_1}{L_\theta}, \dots, \frac{\zeta_l}{L_\theta}, BGaps_{right}(\tau) = \frac{\xi_1}{L_\theta}, \dots, \frac{\xi_r}{L_\theta}.$$

The value of L_θ is easily estimated by using the persistent statistics on T along with the cardinality n and the number of leaf blocks L , similarly to Section 4.3. After having computed the number and extension of gaps between the blocked sections of $output_{IO}$, we use this information to estimate $join_{IO}$. Depending on the length and the position of each block gap g , a specific number of leaf block accesses occurs. For gaps smaller than one disk block, i.e. $g \leq 1$, the I/O is increased by this very gap length g with a weight of 1 (cf. Figure 41a). According to Figure 41b, the I/O overhead for larger gaps depends on the gap offset to the leaf blocks and is restricted to blocks at the gap border. For gaps $g > 1$, our formula to estimate the contribution $gap_{IO}(g)$ of a gap g to $join_{IO}$ therefore focuses on the fraction $g' = g - \lfloor g \rfloor$. Since we assume a uniform distribution of gap offsets with respect to the leaf blocks in the *upperIndex* and *lowerIndex*, the mean contributions of the left and right borders of a gap $g > 1$ to $gap_{IO}(g)$ are $1 + g'$ with weight $1 - g'$ and g' with weight g' . The overall value then sums to $(1 - g') \cdot (1 + g') + g' \cdot g' = 1$, and the distinction of cases simplifies to

$$gap_{IO}(g) = \min(1, g).$$

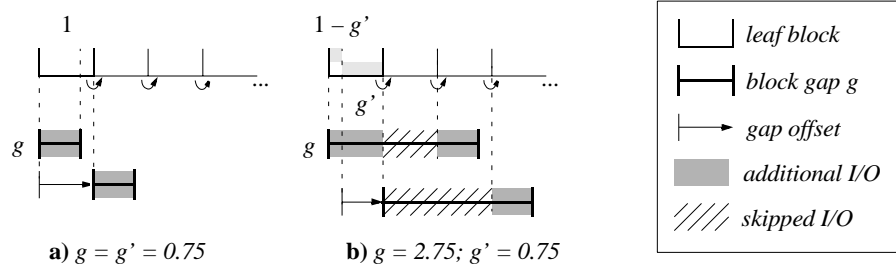


Figure 41: Additional I/O due to block gaps g between range queries.

With respect to I/O cost, random access to a leaf block is, therefore, only beneficial if the preceding block gap is larger than the size of a disk block. In consequence, gaps covering only fractions of a disk block could be sequentially scanned without causing any I/O overhead. This observation opens up a promising potential to further improve the performance of the RI-tree (cf. Chapter 5). For all gaps between the range queries on the *upperIndex* and *lowerIndex* for a given query interval τ on an RI-tree T , we estimate the additional I/O for the join processing as

$$join_{I/O}(T, \tau) = \sum_{g \in BGaps_{left}(\tau) \cup BGaps_{right}(\tau)} gap_{I/O}(g) .$$

The total I/O cost for an interval intersection query τ on an RI-tree T is then summarized by

$$total_cost_{I/O}(T, \tau) = output_{I/O}(T, \tau) + join_{I/O}(T, \tau).$$

4.5 Extension to Spatial Interval Sequences

The presented functions for selectivity and cost estimation of interval intersection queries can be extended naturally to interval sequences. To enable query optimization for spatial queries on multidimensional extended objects, the corresponding interval sequence query for the chosen space-filling curve has to be evaluated. Unfortunately, a straightforward application of the proposed techniques to each single interval of the interval sequence can be too inefficient due to the following reasons:

- A spatial query is specified by an extended, multidimensional object. For arbitrary query objects, in particular objects not already indexed in the RI-tree, we have to

compute the corresponding interval sequence before the above functions can be applied. This would typically be the case for window or box queries.

- As the cardinality of the interval sequence of a spatial object is proportional to its multidimensional surface [Gae 95][FJM 97][MJFS 96], the estimation for each single interval consumes very much CPU cost in the context of pure query optimization, even if the interval sequence of the spatial object has already been computed.

Thus, the evaluation on the full interval sequence anticipates substantial parts of the merely potential query processing on the RI-tree. The extensible optimizer might choose another access path instead, e.g. a full table scan on the base table or an index scan for a non-spatial predicate. In this case, the interval sequence filter may be skipped, and the query processor may refine the spatial predicate directly on the accurate representation of the spatial objects, e.g. on triangle meshes. Thus, the spatial decomposition and linearization of the query object into intervals would have been done in vain. The following paragraphs address these issues by extending the selectivity estimation and the cost model of the foregoing sections to a coarse aggregation of the potential query interval sequence. We will rely on the optimized approach to process interval sequence intersections, as presented in paragraph 3.7.2.

4.5.1 Aggregates on Interval Sequences

In order to minimize the cost of generating and evaluating the fine-grained interval sequence F of a spatial query, we compute a coarse size-bound approximation C [Ore 89] which conservatively approximates F with c intervals. The bound c could be set to a low constant number of intervals, or alternatively, c may depend on the spatial extension of the query object.

Let $F = (\psi_1, \psi_2, \dots, \psi_f)$ be the fine-grained interval sequence which would be the result of a granularity-, size- or error-bound decomposition of the spatial query object. Let $C = (\phi_1, \phi_2, \dots, \phi_c)$ be a coarse approximation of the query interval sequence (cf. Figure 42), where $c \ll f$. The basic idea is to materialize and use C instead of F for the query optimization phase, because deriving an estimation of the selectivity and the cost from F would already reach the CPU complexity of processing the exact query itself.

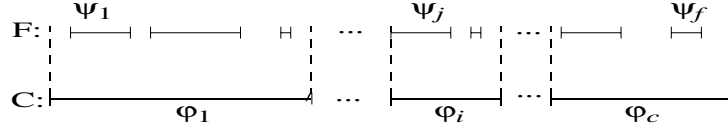


Figure 42: Approximation C of a fine-grained interval sequence F .

We first define an intersection ranking function $\rho_{\text{intersect}}$ and, based on this definition, two aggregates *coverage* and *cardinality* on interval sequences.

Definition 13 (*Intersection Ranking Function*).

Let $B \subseteq \mathbb{R}$ be a domain of interval bounds and let $D = \{(l, u) \in B^2 \mid l \leq u\}$ be the corresponding interval domain. For intervals $\tau = (l_\tau, u_\tau) \in D$ and $\kappa = (l_\kappa, u_\kappa) \in D$, the *intersection ranking function*, $\rho_{\text{intersect}}: D \times D \rightarrow \mathbb{R}$, is defined by

$$\rho_{\text{intersect}}(\tau, \kappa) = \begin{cases} \min(u_\tau, u_\kappa) - \max(l_\tau, l_\kappa), & \text{if } \tau \text{ and } \kappa \text{ intersect;} \\ 0, & \text{otherwise;} \end{cases}$$

Definition 14 (*Aggregates on Interval Sequences*).

Let $F = (\psi_1, \psi_2, \dots, \psi_f)$, $f \geq 1$, and $C = (\phi_1, \phi_2, \dots, \phi_c)$, $c \geq 1$, be two interval sequences representing the same spatial object. Let C be a conservative approximation of F , i.e. $c \leq f$ and

$$\bigcup_{i=1 \dots f} \psi_i \subseteq \bigcup_{i=1 \dots c} \phi_i$$

For each i , $1 \leq i \leq c$, we define the following *aggregates* on F with respect to C :

(i) Let $\rho_{\text{intersect}}$ be the intersection ranking function. The *coverage* along ϕ_i of F is

$$\text{coverage}(\phi_i, F) = \sum_{j=1}^f \rho_{\text{intersect}}(\phi_i, \psi_j).$$

(ii) The *cardinality* along ϕ_i of F is

$$\text{cardinality}(\phi_i, F) = |\{j, 1 \leq j \leq f \mid \phi_i \text{ intersects } \psi_j\}|.$$

Thus, *coverage* denotes the fine-grained hyper-volume within a single interval of the coarse interval sequence C , while *cardinality* denotes the number of the respective fine-grained intervals. These aggregates can be computed or estimated by the volume and surface of the query region [MJFS 96]. In the following, we will give a

short example of the computation of C and its aggregates without a prior materialization of F . Let the query region be a rectilinear box, and let $c = 1$: Assuming that the multi-dimensional data space is ordered according to the Z-ordering, the boundaries of the span $\phi_1 = (l_{min}, u_{max})$ of F can be easily computed by mapping just the two diagonal vertices with the lowest and highest x-, y- and z-values of the query box onto the Z-curve [RMF+ 00]. Thereby, we get the coarsest approximation that is possible, $C = (\phi_1)$. If F would be generated by a granularity-bound decomposition, $coverage(\phi_1, F)$ is equal to the grid volume of the query box, and $cardinality(\phi_1, F)$ is equal to the number of intervals in F . This number can be computed in a similar way as the number of respective quadtree tiles, as it has been presented by Faloutsos, Jagadish, and Manolopoulos [FJM 97].

4.5.2 Extended Selectivity Estimation

For each interval $\phi_i = (l_i, u_i)$ in C , we can assume a data space of $coverage(\phi_i, F)$ to be occupied by the corresponding intervals in F . Starting from a selectivity estimation $\sigma(I, \phi_i)$ for an intersection query ϕ_i on the intervals I , we estimate the total selectivity $\sigma(I, C, F)$ for the fine-grained interval sequence F by

$$\sigma(I, C, F) = \sum_{i=1}^c \left(\sigma_i(I, \phi_i) \cdot \frac{coverage(\phi_i, F)}{u_i - l_i} \right),$$

where $\sigma_1(I, \phi_1) = \sigma(I, \phi_1)$. The computation of $\sigma_i(I, \phi_i)$, $1 < i \leq c$, is similar to $\sigma(I, \phi_i)$, but considers only the portions of the ranges for r_{left} , r_{inner} and r_{right} which have not already contributed to $\sigma_{i-1}(I, \phi_{i-1})$. Thereby, the range queries on the quantile statistics are kept disjoint, and $0 \leq \sigma(I, C, F) \leq 1$ holds.

More typically, spatial objects are stored by multiple intervals. As the quantiles or histogram statistics do not recognize the intervals of an interval sequence as an entity, the above selectivity is estimated with respect to the total number of intervals, and not with respect to the total number of interval sequences, i.e. stored spatial objects. In Section 4.6, we will empirically evaluate the deviation to the object-based selectivity. Nevertheless, as the interval-based selectivity directly reflects the actual cost of query processing on the database of replicated objects, it will be used as input for the cost estimation.

4.5.3 Extended I/O Cost Model

Next, we present an I/O cost model for coarse interval sequences with aggregation. Whereas the computation of $output_{I/O}$ for F can be done straightforward, the non-blocked overhead $join_{I/O}$ has to be estimated for a sequence of interval intersections. We assume that the two observations of Section 4.4 also hold for interval sequences. In particular, the locality assumption of random I/Os is correct, as for sorted query sequences, the transient join partners are generated in ascending order over all gaps [KPS 01]. The local descending ordering of the *right queries* within each gap is typically absorbed even by small LRU caches. Thus, our former approach to estimate $join_{I/O}$ is applicable with the slight modification that the fine-grained intervals of F are not known this time.

For each interval $\phi_i = (l_i, u_i)$ in C , we assume a number of $cardinality(\phi_i, F)$ of covered intervals in F . The average interval length among these intervals is

$$avgInt(\phi_i, F) = \frac{coverage(\phi_i, F)}{cardinality(\phi_i, F)}.$$

Similarly, the average gap length along ϕ_i is estimated by

$$avgGap(\phi_i, F) = \frac{(u_i - l_i) - coverage(\phi_i, F)}{cardinality(\phi_i, F) - 1}.$$

For each ϕ_i , the number and distribution of contained *left* and *right* queries could be estimated, and the multi-sets $NGaps_{left}(\phi_i, F)$ and $NGaps_{right}(\phi_i, F)$ of query gaps among the fine-grained representation of ϕ_i in F could be populated similarly as in Section 4.4. But, as this extensive analysis would already be based on the estimated input of C and would increase the computational complexity of the cost model, we simply assume that the mean value of $NGaps_{right}(\phi_i, F)$ is equal to the sum of $avgGap(\phi_i, F)$ and $avgInt(\phi_i, F)$. As the *inner queries* are integrated with the *left queries*, the mean value of $NGaps_{left}(\phi_i, F)$ then corresponds to $avgGap(\phi_i, F)$. Thereby, we subsume all potential queries within gaps of F by single *left* and single *right queries* (cf. Figure 43):

$$(i) \quad avg(NGaps_{left}(\phi_i, F)) = avgGap(\phi_i, F),$$

$$(ii) \quad avg(NGaps_{right}(\phi_i, F)) = avgGap(\phi_i, F) + avgInt(\phi_i, F).$$

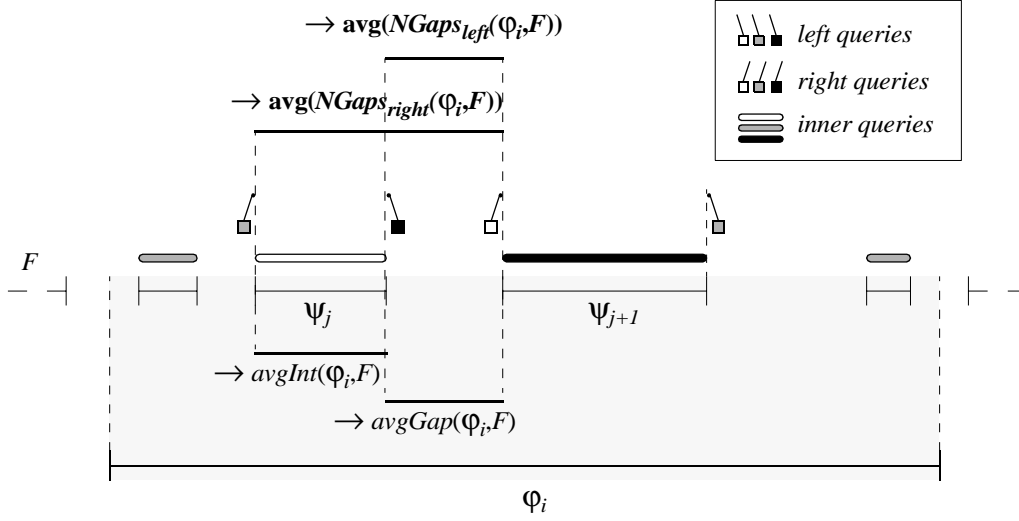


Figure 43: Extended RI-tree cost model.

Contributors (\rightarrow) to the averages $avgInt(\Phi_i, F)$ and $avgGap(\Phi_i, F)$ for a coarse interval, and the mean of $NGaps_{left}(\Phi_i, F)$ and $NGaps_{right}(\Phi_i, F)$.

The respective averages on the block gaps $BGaps_{right}(\Phi_i, F)$ and $BGaps_{left}(\Phi_i, F)$ can be computed analogously to Section 4.4.

When applied to the lexicographic ordering, these plain averages suffice since particularly for convex spatial objects, the variance among the interval gaps is very small. Unfortunately, on the far more powerful concept of fractal space filling curves, including the Z- and Hilbert-ordering, the variance among the gap lengths is extremely high. In this case, the above averages provide a very weak characterization for the expected gap distribution. We have observed that the binary logarithms of fractal gaps typically obey an exponential distribution. Furthermore, histograms on fractal gaps show local peaks at whole multiples of the original data dimension d , i.e. at gap lengths around $2^{k \cdot d}$, $k \geq 0$. This behavior is caused by the fact that many gaps represent empty square-shaped (2D) or cubical (3D) regions at the boundary of spatial objects [Pfe 01]. The shaded area in Figure 44 depicts a real gap distribution among 100 Z-ordered query regions for a two-dimensional real-world database. In Section 6.6, it is shown that similar distributions occur for 3D CAD data. Our proposed estimation gap_{est} of this distribution is as simple as effective: the overall exponential shape is approximated by a geometric distribution on the base points

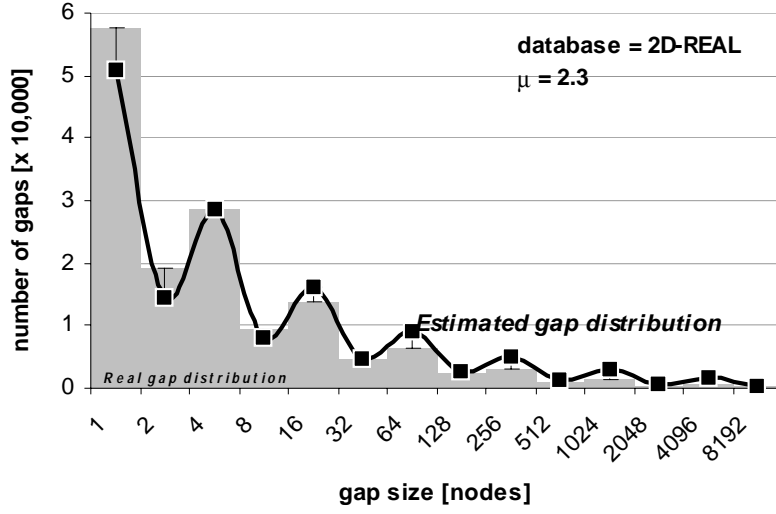


Figure 44: Real and estimated gap distribution.
(Z-ordered interval sequences of 2D spatial objects)

$p_k = (2^{k \cdot d})_k, k \geq 0$. We use a constant mean value μ according to the complexity of the spatial objects and to the chosen space-filling curve:

$$gap_{est}(p_k) = \frac{1}{\mu} \cdot \left(1 - \frac{1}{\mu}\right)^k, k \geq 0.$$

For typical Z-ordered spatial objects, for example, we observed that $\mu = 2.3$ seems to be a good choice. Between two base points, we assign fractions of the cardinality of the larger base point to model the observed decreasing frequency of the corresponding tile shapes:

$$gap_{est}(2^i \cdot p_k) = \frac{1}{2^i} \cdot gap_{est}(p_{k+1}), k \geq 0, 1 \leq i < d.$$

Figure 44 depicts the estimated gap distribution gap_{est} , normalized to the total cardinality of gaps. In most cases, gap_{est} is an accurate estimation of the real distribution of fractal gaps among a fine-grained interval sequence F which can be used instead of $avgGap(\phi_i, F)$ to get better distributions for $BGaps_{right}(\phi_i, F)$ and $BGaps_{left}(\phi_i, F)$.

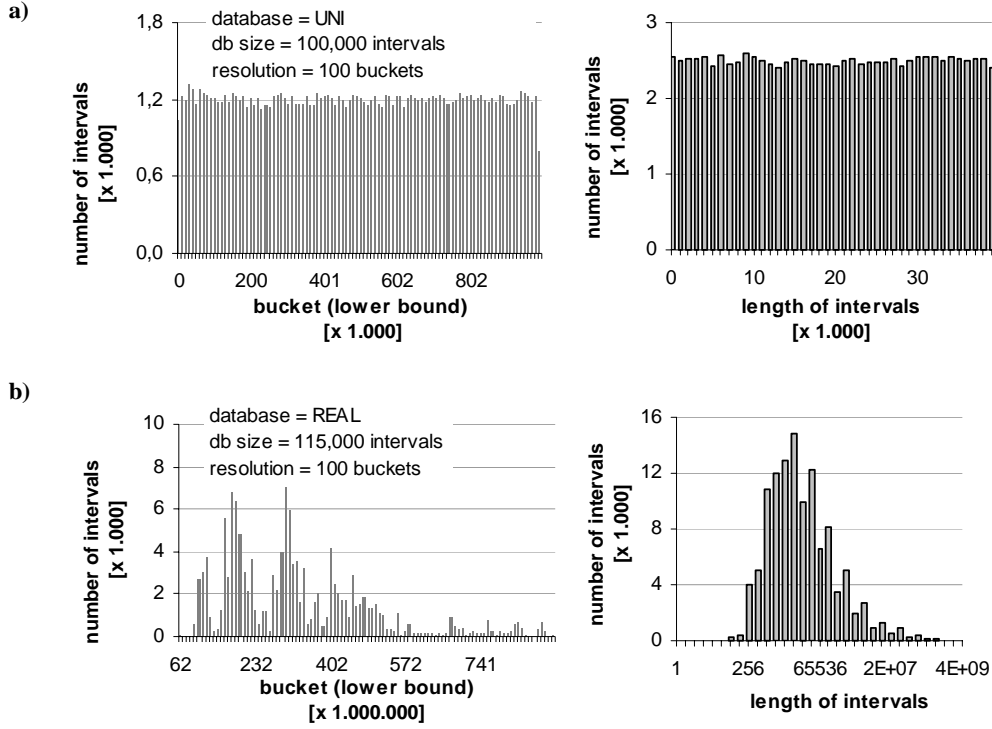


Figure 45: Histograms of interval distributions.

a) for uniform data, **b)** for real data

4.6 Empirical Evaluation

4.6.1 Experimental Setup

We implemented the proposed functions for the estimation of selectivity and execution cost on the Oracle Server Release 8.1.6 using built-in methods for statistics collection, analytic SQL functions, and the PL/SQL procedural runtime environment. All experiments were performed on an Athlon/750 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.

The experiments for the evaluation of statistics, selectivity estimation, and cost model have been executed on various interval databases. We have used a synthetic data set of intervals following a uniform starting point and length distribution (*UNI*) and intervals derived from a real spatial data set (*REAL*). For both databases *UNI* and *REAL*, Figure 45 depicts the histogram statistics. A peak in the histogram

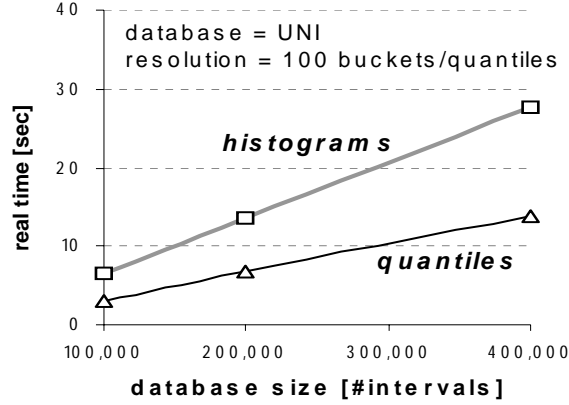


Figure 46: Computation cost of histogram-based and quantile-based statistics.

visualization denotes a high density of interval data. In case of the UNI data set, the data space $[1..1,000,000]$ is covered with a uniform density. Furthermore, we used Z-ordered interval sequences representing polygons of real world spatial objects ($REAL_{sequence}$).

To evaluate the quality of the selectivity and cost prediction, we determined the average relative error of the estimates. This measure denotes the ratio of the absolute estimation error to the actual query result, averaged over a set of queries S . If e_i is the estimated and r_i is the actual result size of a query q_i , the average relative error of the estimated selectivity for S is defined as:

$$\text{Avg relative error (selectivity)} = \left(\sum_{q_i \in S} |r_i - e_i| \right) / \left(\sum_{q_i \in S} r_i \right)$$

For the estimations of the actual I/O cost, the average relative error is defined analogously. This measure is a common technique to evaluate selectivity estimations and cost models, see e.g. [APR 99]. It is undefined if all queries in a query set produce zero output or zero cost. The following results show the averages of, in total, 100 intersection queries for the UNI, REAL and $REAL_{sequence}$ databases.

4.6.2 Computation of Statistics

The persistent statistics must be recomputed in order to adapt to changing data distributions. For highly dynamic data, the database administrator might even decide to trigger the computation of important statistics periodically. Therefore, a low execution cost for the creation of statistics is essential. Figure 46 compares the total

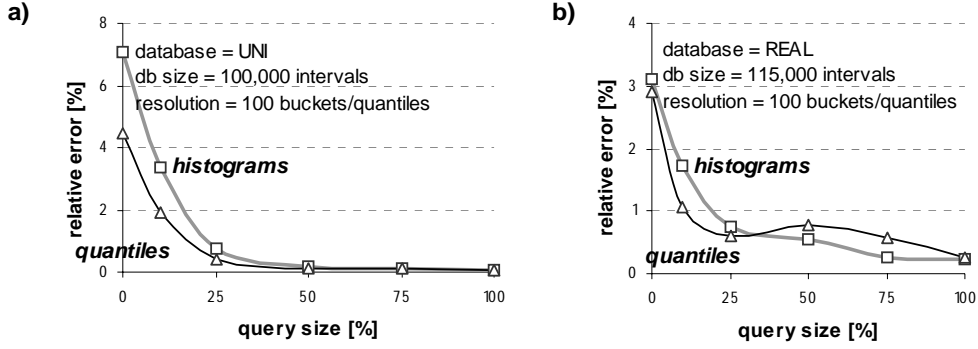


Figure 47: Relative error of selectivity estimation for histograms and quantiles.
a) on uniform data, b) on real data

runtime of computation for the histogram statistics to the quantile statistics for increasing database size, using 100% samples. Due to the overhead of barrier crossing between PL/SQL and SQL, the quantile-based approach outperforms the histogram-based approach by a factor of 2.

4.6.3 Selectivity Estimation

In the next set of experiments, we evaluate the average relative error with respect to the query size, i.e. the percentage of the data space covered by the query region. Figure 47 shows the relative error of the histogram-based and quantile-based statistics on the UNI and REAL database. The resulting accuracy of both, the quantile-based approach and the histogram-based approach is very high. For higher selectivities, the quantile-based approach performs slightly better, yielding estimation errors around 4.5% and 2.9% for the UNI and REAL database, respectively. This result can be explained by the fact that quantiles adapt to the local density of the data, whereas histograms partition the whole data space using buckets of identical size. The next experiment in Figure 48 depicts the average relative error for different resolutions of the persistent statistics, evaluated for a set of intersection queries having 10% average query size. As expected, the estimation error increases significantly for coarser resolutions. Beyond a global optimum at some 100 buckets, the error of the histogram-based approach increases for higher resolutions, due to the replication of intervals spanning multiple histogram buckets. Therefore, we focus on the quantile-based approach in the following experiments, as the representation of intervals is non-redundant.

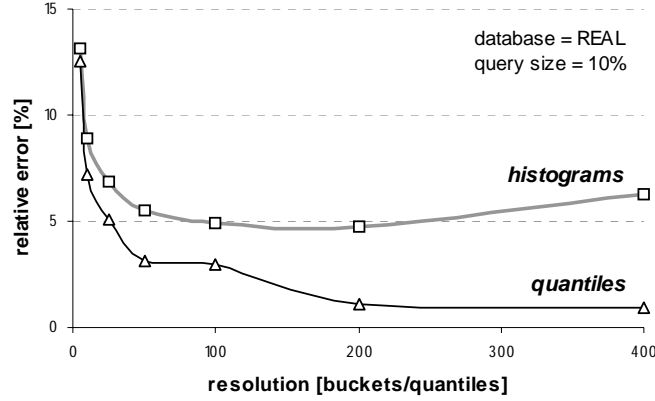


Figure 48: Relative error of selectivity estimation for varying statistic resolutions.

Figure 49 also shows the results on the $REAL_{sequence}$ database. For a coarseness of $c = 1$, we get relative estimation errors between 15% and 30%. Thus, even with the maximal aggregation, the computed estimate gives the query optimizer a good impression of the spatial selectivity. The quality of this hint improves with increasing c , as for $c = \max$ (i.e. $c = f$) in the experiment. For the actual query results we measured both selectivities: with respect to the total number of stored intervals, and with respect to the stored interval sequences, i.e. polygons. Note that the relative error to the actual polygon-based selectivity (*Polygons*) is roughly in the order of the relative error to the actual interval-based selectivity (*Intervals*). Thus, the selectivity on the single intervals largely reflects the selectivity on the original spatial objects. Never-

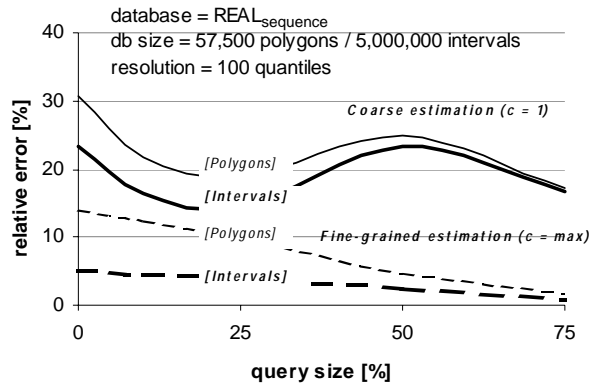


Figure 49: Relative error of selectivity estimation.
($REAL_{sequence}$ database)

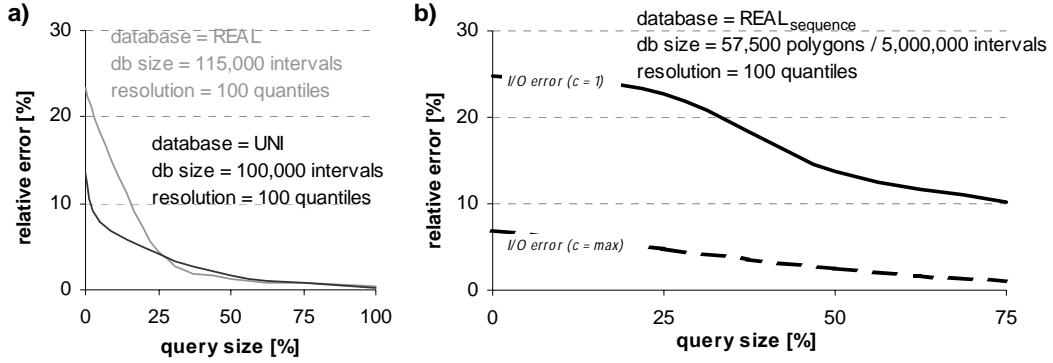


Figure 50: Relative error for cost estimation.
a) on *UNI* and *REAL*, b) on *REAL_{sequence}*

theless, what we have to provide as input for the cost estimation is the more accurate interval-based estimate. The achieved results seem to be comparable to the performance of originally multidimensional statistics [APR 99].

Regarding the runtime, a single selectivity estimation using statistics with a resolution of 100 quantiles for the *UNI*, *REAL* and *REAL_{sequence}* ($c = 1$) databases took about 0.05 seconds on the average.

4.6.4 Cost Estimation

We used the estimated selectivity of the previous section as input for the I/O cost model. The extensible query optimizer uses the resulting estimations to decide upon the usability of the RI-tree for specific queries. Figure 50a presents the relative error of the estimated cost for the *UNI* and *REAL* databases. The relative errors stay below 14% and 23%, respectively. Figure 50b depicts the corresponding results for a coarseness of $c = 1$ and $c = \max$ on the *REAL_{sequence}* database. The I/O error for $c = 1$ at a query size near 0% averages 24.9% and decreases to 10.2% at 75% query size.

Figure 51 and Figure 52 compare the absolute estimations and the actual cost for the blocked output of results ($output_{I/O}$). In addition, $join_{I/O}$ denotes the overhead due to the nested-loop join with the transient query tables. For the sake of comparability to the analytical I/O complexity, the results are shown with respect to the actual query selectivity. Our interpretation of these results is twofold: First, the real I/O cost show that the total I/O is largely determined by the cardinality of the query result, whereas the overhead for the join processing remains almost constant. The relative cost of the join overhead decreases from 100% at 0% selectivity to almost 0% at 100% selectiv-

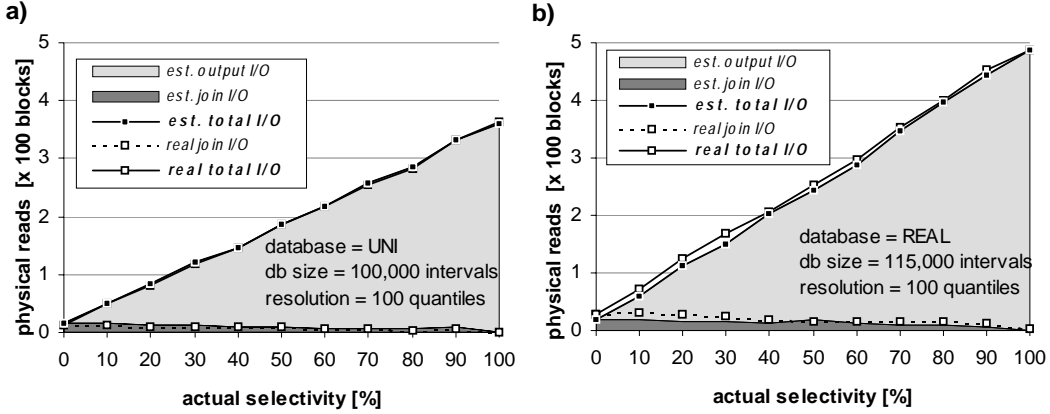


Figure 51: Output cost and join overhead for queries evaluated.
a) on uniform data, b) on real data

ity. According to these empirical results, the overhead of $join_{I/O}$ is negligible for higher values of the query selectivity. Second, we observe that our cost model not only yields tight estimations for the total query cost, but also reflects the distribution between the output and join cost rather accurately.

As expected, Figure 52 shows that the accuracy of the cost estimation increases with a higher granularity c of the coarse interval sequence. Considering the comparable empirical results for cost-models on stand-alone R-trees [HJR 97][TSS 00], and the often significant difference to the cost of alternative access paths including

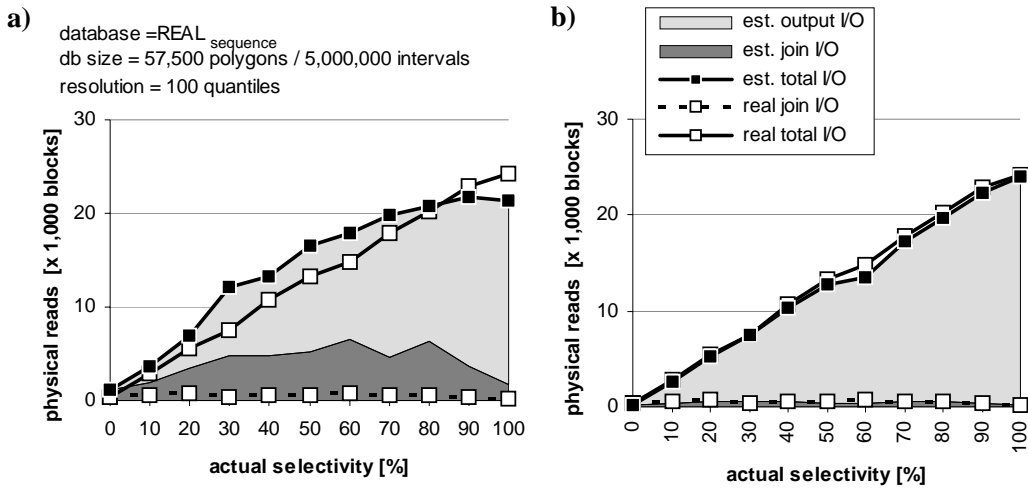


Figure 52: Output cost and join cost for queries using interval sequences.
a) $c = 1$ and b) $c = \max$ (on the $REAL_{sequence}$ database)

full-table scans, we conjecture that already a coarse estimation with $c = 1$ is well suited for spatial queries on the RI-tree. A fine-grained computation is, of course, much more accurate, but already anticipates a significant amount of the cost of the potential query.

Regardless of the actual query selectivity, the cost computation on the databases UNI, REAL, and REAL_{sequence} ($c = 1$) took about 0.05 seconds.

4.7 Summary

High quality selectivity estimation and cost prediction are the fundamentals of effective query optimization. Particularly for complex query objects and complex query predicates, the recent object-relational database servers provide extensible optimization frameworks that go along with the extensible indexing frameworks, in order to complete the seamless integration of user-defined index structures into the declarative DML. In this chapter, we presented an example for such an extension focusing on the RI-tree that already fits well to modern object-relational extensible indexing frameworks. We particularly propose models to estimate the selectivity of interval (sequence) intersection queries and to predict the cost for query processing. With respect to the generation and management of statistics, the proposed quantile-based selectivity estimation reuses as much built-in functionality of the RDBMS as possible. According to our experimental evaluation, the computed estimations are very accurate.

Chapter 5

Statistic-Driven Acceleration of Relational Index Structures

Relational index structures, as for instance the RI-tree, the RQ-tree or the RR-tree, support efficient query processing on top of existing object-relational database systems. Furthermore, there exist effective and efficient models to estimate the selectivity and the I/O cost in order to guide the cost-based optimizer whether and how to include these index structures into the execution plan. By design, the models immediately fit to common extensible indexing/optimization frameworks, and their implementations exploit the built-in statistics facilities of the database server. In this chapter, we show how these statistics can also be used for accelerating the access methods themselves. For space-partitioning index structures we propose to reduce the number of generated join partners which results in less logical reads and consequently improves the overall runtime. We cut down on the number of join partners by grouping different join partners together according to a statistic driven grouping algorithm. For hierarchical data-partitioning index structures, we propose to reduce the navigational index traversal cost by using “extended index range scans”. If a directory node is “largely” covered by the actual query, the recursive tree traversal for this node can beneficially be replaced by a scan on the leaf level of the index instead of navigating through the directory any longer. Our experiments on an Oracle9i database yield an average speed-up between 20% and 10,000% for spatial collision queries on the RI-tree, the RQ-tree and on the RR-tree.

5.1 Introduction

The efficient management of complex objects has become an enabling technology for many novel database applications. Former approaches which try to generate efficient read schedules for a given set of disk pages must have access to the exact information where the blocks are located on the disk [SLF 93]. As this information is not available on top of an ORDBMS, we pursue another idea which exploits already existing statistics in order to accelerate spatial query processing. We introduce our approach in general for space- and data-partitioning index structures as well as exemplarily for spatial intersection queries performed on the RR-tree, the RQ-tree and the RI-tree.

The remainder of this chapter is organized as follows. In Section 5.2, we show how we can use the already existing statistics to accelerate the query process on space-partitioning relational access methods. In addition to the known error- and size-bound decomposition approaches, we present a new statistic-bound decomposition approach for decomposing spatially extended objects. In Section 5.3, we present our approach for hierarchical data-partitioning index structures where at each directory node it is individually decided whether it is beneficial to switch to a range scan on the leaf level of the index or whether it is beneficial to take further advantage of the index-directory. In Section 5.4, we present convincing experimental results and conclude the chapter, in Section 5.5, with a short summary.

5.2 Acceleration of Relational Space-Partitioning Access Methods

For commercial use, a seamless and capable integration of spatial indexing into industrial-strength databases is essential. In order to integrate these index structures into modern ORDBMSs, we need suitable cost models (cf. Chapter 4), which exploit the built-in statistics facilities of the database server. In this section, we discuss how we can use these statistics to accelerate the query processing itself.

In addition to the query optimizer of an ORDBMS, which uses statistics for cost-based optimizations, we use the statistics to minimize the overall navigational cost of a relational index structure. Our approach accelerates relational access methods by trying to reduce the total number of logical reads for a given query. The relational access method can be any custom index structure mapped to a fine granular

relational schema which is organized by built-in access methods, as for instance the B^+ -tree.

There already exist variants of basic relational index structures which try to minimize the number of join partners and, consequently, the number of logical reads and the overall query response time. For instance, there exist index structures which were especially tuned for coping efficiently with sequences. One example is the RI-tree as introduced in Section 3.7.2. It supports the efficient detection of intersecting spatial objects, which are represented by interval sequences. The main idea of this index structure is to neglect such nodes as join partners which are already handled by the previous query interval or which will be handled by the following one. The main disadvantage of this approach is that only specific predicates are supported by this kind of index structures. For instance the RI-tree according to [KPS 01] only supports boolean intersection queries, but already fails to compute the intersection volume. Similar optimizations are possible for the RQ-tree by eliminating duplicates from the upper hulls resulting from different query tiles of a given query sequence. In contrast to these optimizations which do not take the actual data distribution into account, our new approach accelerates both boolean and ranked intersection queries by exploiting existing statistics, i.e. taking the data distribution into consideration.

In this section, we first look at very comprised statistic values, which can already be very useful for accelerating spatial relational index structures (cf. Section 5.2.1). In Section 5.2.2, we show how we can benefit from the statistics, used by the cost-models belonging to a relational access method. Finally, in Section 5.2.3, we introduce a new statistic-based decomposition approach in addition to the existing error- and size-bound decomposition approaches. In all three subsections, we first introduce our ideas in general and then show how to adapt them to the RI-tree and the RQ-tree.

5.2.1 Statistics Related to the Relational Access Method

As already indicated in Definition 1 and Figure 25, 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. 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 dramatical-

ly. If we assume for instance that one half of the data space is completely empty, and we carry out a box volume 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 if we store coarse information about the *actual data distribution* along with the *fixed data space* extension in the metadata table.

In Table 1, we summarized some optimizations which are suitable for the RI-tree and the RQ-tree.

denotation	explanation
<i>MaxNodeLevel</i> <i>MinNodeLevel</i> (RI-tree)	These two parameters reflect the highest and lowest level of the <i>fork-nodes</i> 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.
<i>MaxLeftDist</i> <i>MaxRightDist</i> (RI-tree)	These two parameters reflect the maximum distance of the boundary values of any database interval to its corresponding <i>fork-node</i> . 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.
<i>MaxTileLevel</i> <i>MinTileLevel</i> (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 1: Simple Statistics for the RI-tree and the RQ-tree

These simple statistics are especially useful for indexing extended spatial objects. If we use the RI-tree or the RQ-tree for indexing extended 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 [KPPS 03a] (cf. Section 5.4).

5.2.2 Statistics Related to the Built-in Index Structure

In Chapter 4, it was shown that using quantiles (‘equi-count histograms’) is more suitable for estimating the selectivity and the corresponding I/O cost than using histograms (‘equi-width histograms’). In addition, the runtime required for the histo-

gram computation is increased by the cost of barrier-crossings between the declarative environment of the SQL layer and our stored procedure. Fortunately, most ORDBMS comprise efficient built-in functions to compute single-column statistics, particularly for cost-based query optimization. Available optimizer statistics are accessible to the user by the relational data dictionary. The basic idea of our quantile-based selectivity estimation is to exploit these built-in index statistics rather than to add and maintain user-defined histograms.

We will now discuss how we can use this information to accelerate the query process itself. Any query for a relational index structure, e.g. RI-tree or RQ-tree, leads to several index range scans on the built-in index structures, e.g. B^+ -tree. The general idea of our approach is to minimize the overall navigational cost of the built-in index by applying extended index range scans. Thereby, we read false hits from the index, which are filtered out by a subsequent refinement step. Our approach closes the gaps between the index range scans if and only if the number of additionally 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. This decision whether to close a gap is based on the built-in statistics. We will now formally introduce this idea.

Index Range Scan Sequences. For spatial intersection queries, the query object Q leads to many disjoint range queries on the built-in index I , e.g. the B^+ -tree. We consider them as a sequence $Seq_{Q,I} = (\langle s_1, \dots, s_n \rangle)$ of index range scans for which the following assumptions hold (cf. Figure 53a):

- The elements r_i stored in the index are of the same type as l_i and u_i . Furthermore, we assume that the elements r_i can be regarded as a linear ordered list $L(I) = \langle r_1, \dots, r_N \rangle$ for which $r_1 \leq \dots \leq r_N$ 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 $Seq_{Q,I} = (\langle s_1, \dots, s_n \rangle)$ 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.

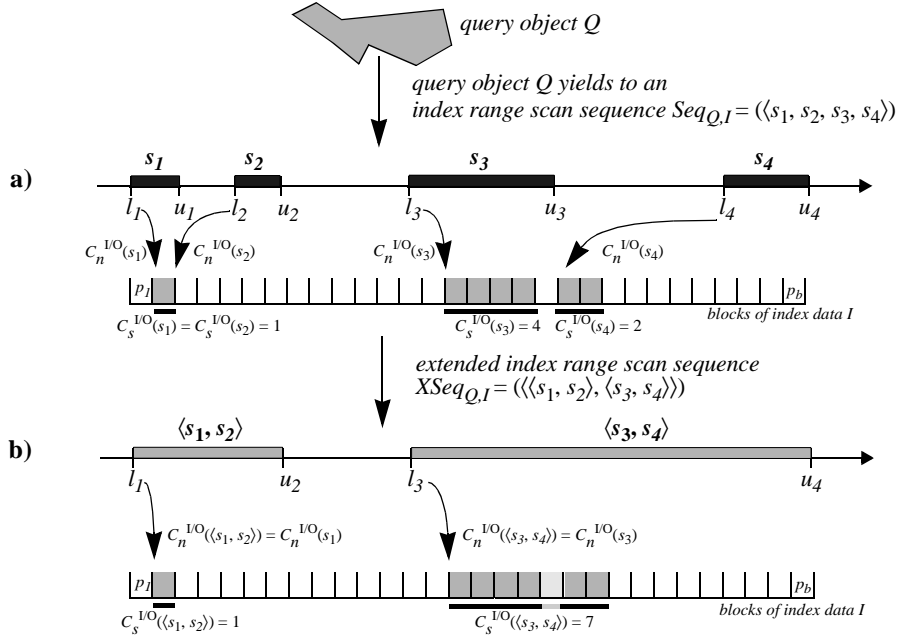


Figure 53: Accelerated query processing.

a) Index range scan sequence, b) Extended index range scan sequence

Formally, $C^{I/O}(s) = C_n^{I/O}(s) + C_s^{I/O}(s)$, with the following two properties:

- (i) $C_n^{I/O}(s) = C_n^{I/O}(p(r'))$ (navigational cost)
- (ii) $C_s^{I/O}(s) = C_s^{I/O}(\langle p(r'), \dots, p(r'') \rangle)$ (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 $C(Seq_{Q,I})$ associated with $Seq_{Q,I} = (\langle s_1, \dots, s_n \rangle)$ are determined by:

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

Extended Index Range Scan Sequences. The main purpose of our approach is to minimize the overall cost for the navigational part of the built-in 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 $xs = \langle s_r, \dots, s_s \rangle$ 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 53b. 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 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. The I/O cost $C^{I/O}(xs)$ associated with one extended range scan $xs = \langle s_r, \dots, s_s \rangle$ are composed from two parts $C^{I/O}(xs) = C_n^{I/O}(xs) + C_s^{I/O}(xs)$, with the following properties:

- (i) $C_n^{I/O}(xs) = C_n^{I/O}(s_r)$ (navigational cost)
- (ii) $C_s^{I/O}(xs) = C_s^{I/O}(l_r, u_s)$ (scan cost)

CPU cost. The CPU cost $C^{CPU}(xs)$ associated with one extended range scan $xs = \langle s_r, \dots, s_s \rangle$ denote the cost which are required to perform the filter operation for all tuples resulting from the extended range scan:

$$C^{CPU}(xs) = C^{CPU}(\langle r', \dots, r'' \rangle), \text{ where } \forall r \in L(I) : (r' \leq r \leq r'') \Leftrightarrow (l_r \leq r \leq u_s).$$

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

$$C(XSeq_{Q,I}) = \sum_{j=1}^m C(xs_j) = \sum_{j=1}^m (C^{I/O}(xs_j) + C^{CPU}(xs_j)).$$

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, whether 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.

The multi-set S of our quantile vector (q_0, \dots, q_v) (cf. Definition 8) is formed by the values of the first attribute A_1 of the domain values of our index I . 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 average number of index entries per disk block, 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.

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

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

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, \dots, s_n of the index range scan sequence $Seq_{Q,I} = \langle s_1, \dots, s_n \rangle$, whether we close this gap or skip it. Thus we obtain an *extended index range scan* sequence $XSeq_{Q,I} = \langle \langle s_{i_0+1}, \dots, s_{i_1} \rangle, \dots, \langle s_{i_{m-1}+1}, \dots, s_{i_m} \rangle \rangle$, which satisfies the following property:

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

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

In the next paragraphs, we will show how our approach can be applied to the RI-tree and the RQ-tree.

Adoption to the RQ-tree. Assume object Q in Figure 54 is used as query object. Then there are multiple exact match and range scan queries 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,

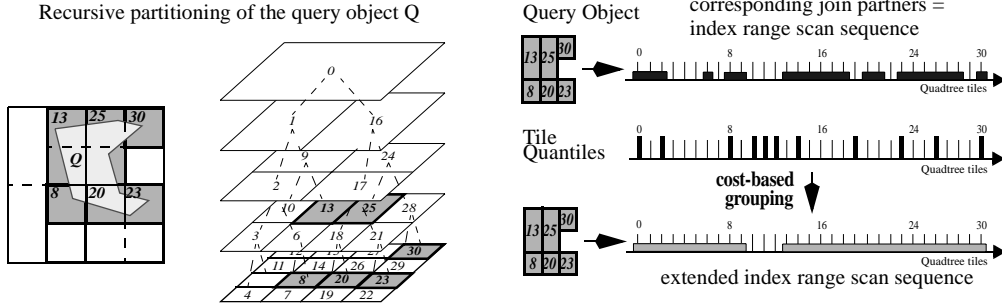


Figure 54: Cost-based tile grouping.

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 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 h_B . If we close the gap, we reduce the navigational cost as follows: $C_n^{I/O} = C_n^{I/O} - h_B$. 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 h_B , we close this gap. Thus we reduce the *join cost* $C_n^{I/O}$ by h_B , while not increasing the *output cost* C_s by more than h_B . This process is depicted in Figure 54.

The above mentioned *cost-based grouping* step can be carried out in a procedural preparation step *JoinPartGen*, leading to one single SQL-statement (cf. Figure 55). The resulting table *tiles* contains entries of a type which consists of three attributes *ZvalLow*, *ZvalHigh* and *ExactZvalList*. The attribute *ExactZvalList* is a collection of tile ranges, representing the accurate query information. It is needed for an additional refinement step to filter out false index hits, by calling *TestZval()*.

For more detail, we refer the interested reader to [Müh 03].

```
SELECT DISTINCT idx.id
FROM   DBTILES idx, TABLE(JoinPartGen(BOX((0,0,0),(10,10,10)))) tiles,
WHERE  (idx.zval BETWEEN tiles.ZvalLow AND tiles.ZvalHigh) AND
        TestZval(idx.zval, tiles.ExactZvalList);
```

Figure 55: Accelerated window query on a Relational Quadtree.

Adoption to the RI-tree. Similarly to the RQ-tree, we can integrate the cost-based grouping algorithm into the procedural query preparation step of the RI-tree. This grouping algorithm is independent of the high-level relational index-structure. It is only based on a B^+ -tree and on a quantile vector. The quantile vector in the case of the RI-tree, is formed by the *fork-nodes* of the intervals stored in the database. Note that this node quantile was also used for an effective and efficient cost-model for intersection queries on RI-trees (cf. Chapter 4).

For more detail, we refer the interested reader to [Mai 03].

5.2.3 Statistics Related to the Object Decompositioning

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. In this section, we introduce an additional decomposition approach which decomposes the object based on the expected I/O cost. The expected I/O cost can be estimated in the same way the optimizer estimates the cost for a given query. Like the approach of the last section, the decomposition of the query object is controlled by the statistics which are available for free and maintained by the cost models of the available index structures.

Figure 56 depicts this top down grouping algorithm which is beneficial for the RI-tree and the RQ-tree. 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 [Ore 89] 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,

```

ALGORITHM Decompose ( $Q, QV$ )
BEGIN
     $query\_sequence\_list := split\_at\_maximum\_gap(Q);$ 
     $cost_0 := statistic\_look\_up(Q, QV);$ 
     $cost_{dec} := 0;$ 
    FOR EACH  $q$  in  $query\_sequence\_list$  DO
         $cost_{dec} := cost_{dec} + statistic\_look\_up(q, QV);$ 
    END FOR;
    IF  $cost_0 > cost_{dec}$  THEN
        FOR EACH  $q$  in  $query\_sequence\_list$  DO
             $Decompose(q, QV);$ 
        END FOR;
    ELSE
         $report(Q);$ 
    END IF;
END.

```

Figure 56: Grouping algorithm *Decompose*.

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 comparably 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 (cf. Section 6.3.3). In the refinement step, we further decompose the query intervals or tiles on demand from the compact geometric information (cf. Section 6.4.1). In Chapter 6, we discuss in detail the problem of finding an appropriate object decomposition for complex spatial objects.

5.3 Acceleration of Relational Data-Partitioning Access Methods

In contrast to space-partitioning index structures, data-partitioning index structures naturally adapt to the actual data distribution which results in a very good query response behavior. In this section, we show how we can achieve efficient query processing on data-partitioning index structures within general purpose database systems. Again, we reduce the navigational index traversal cost by using extended index range scans. If a directory node is “largely” covered by the actual query, the recursive

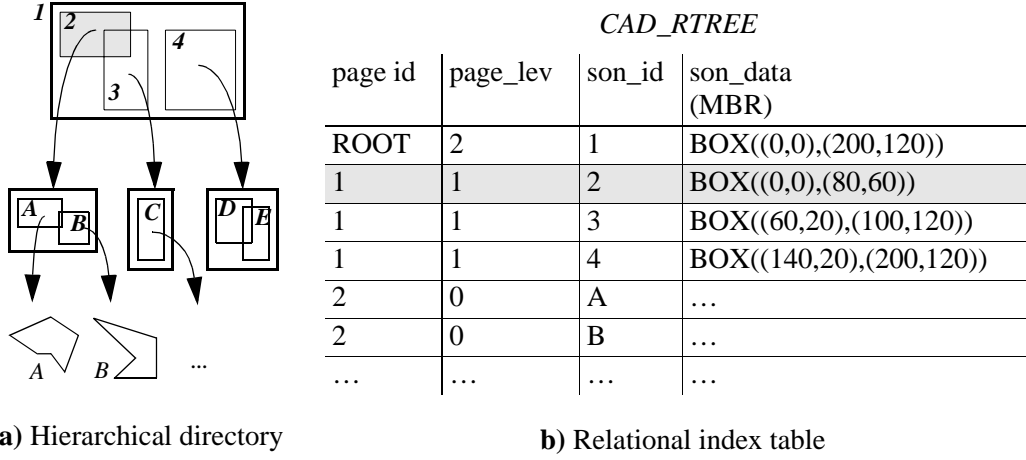


Figure 57: An example for an hierarchical index structure (the RR-tree).

tree traversal for this node can beneficially be replaced by a scan on the leaf level of the index instead of navigating through the directory any longer. On the other hand, for highly selective queries, the index is used as usual. In this section, we demonstrate the benefits of this idea for spatial collision queries on the Relational R-tree. In Section 10.3, it is shown how we can apply the presented concept to similarity range queries on the Relational M-tree.

In contrast to an optimizer which has to decide “once and for all” whether to include a specific access method into the execution plan, the approach of this section is much more fine-grained. At each directory node of a hierarchical index structure it is individually decided whether it is beneficial to switch to a range scan on the leaf level of the index or whether it is beneficial to take further advantage of the index-directory. The experiments show that our new approach always adapts to the best of the two worlds “index” and “sequential scan”. Therefore, the optimizer can under all circumstances include our new approach into the query execution plan.

The remainder of this section is organized as follows: In Section 5.3.1, we shortly discuss the relational mapping of hierarchical index structures. In Section 5.3.2, we formally present our general idea. In Section 5.3.3, we show how to apply this idea to the Relational R-tree.

5.3.1 Relational Mapping of a Hierarchical Index Structure

Generally any tree structure can be mapped to a relation (*page_id*, *son_id*, *son_data*). Figure 57 depicts such a mapping for the hierarchical R-tree. This map-

```

SELECT son_id AS id FROM CAD_RTREE
WHERE page_lev = 0 AND                                // select data object
      intersect (son_data, BOX((0,0),(100,100)))
START WITH page_id = ROOT
CONNECT BY
      intersect (PRIOR son_data, BOX((0,0),(100,100))) AND
      PRIOR son_id = page_id;                          // declarative tree traversal

```

Figure 58: SQL box intersection query on a RR-tree (Oracle syntax).

ping is slightly different compared to the one presented in Section 3.6.1. It allows that a *page_id* of level 0 can be assigned to many tuples in the relational index table. We chose the mapping of Figure 57 as basis for this section as the following theoretical outline can be presented more clearly. A primary filter for a window intersection query using SQL is shown in Figure 58. The recursive tree traversal is realized by the “connect by”-clause. The intersect-statement within this “connect by”-clause allows to prune parts of the directory, whereas the intersect statement in the “where”-clause filters out elements on the leaf level.

5.3.2 General Idea

We assume that the *page_ids* are ordered according to a depth-first tree traversal and that we have a B^+ -tree on this attribute. Furthermore, we assume that an additional B^+ -tree exists on the attributes *page_level*, *page_id* so that we can easily scan over all data entries, i.e. all entries where the *page_level* is 0. The general idea is that we skip the recursive tree traversal at a certain point and perform an extended range scan on the leaf-level of our index. Thereby, we try to minimize the overall navigational cost on the hierarchical index while allowing to read false hits from the leaf-level of the index which are filtered out by a subsequent refinement step. Figure 59b depicts this general idea. The main advantages of our new approach is that we can reduce the navigational cost related to the hierarchical index structure (filled triangles in Figure 59) and to the built-in B^+ -trees (arrows in Figure 59). On the other hand, we have higher cost related to the scanning of the leaf-level and higher CPU cost related to the additionally required refinement step.

In this section, we will discuss the cost related to a hierarchical tree traversal and the cost related to an extended range scan in general. We will first introduce the general I/O cost and CPU cost related to a certain directory node which presumably

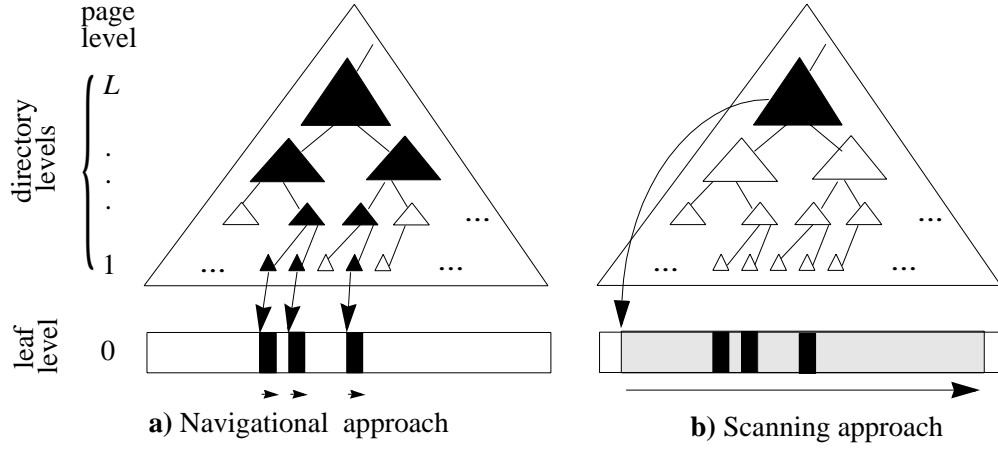


Figure 59: Acceleration of hierarchical index structures.

arise when continuing the tree traversal (cf. Figure 59a) and the cost related to an extended range scan starting at this directory node (cf. Figure 59b). These cost heavily depend on the “overlap-factor” $\sigma = \sigma(q, n)$ which denotes the percentage of accessed tuples during a query q in a certain subtree of a directory node n if the index structure is used as usual. In the following sections, we will show how we can estimate this “overlap-factor” σ for the intersect predicate on the Relational R-tree (cf. Section 5.3.3). We will use the following notations.

symbol	meaning
m	average number of index entries per directory node
b	average number of index entries per disk page
$L(n)$	level of the current directory node n
h_B	height of the B^+ -tree
k_{CPU}	CPU cost for testing one index directory entry
$k_{I/O}$	I/O cost for reading one page from the disk
$\sigma(q, n)$	value between 0 and 1 which denotes the percentage of accessed tuples in the subtree belonging to node n , if the index structure is used as usual for the query processing of a query q
$cost_{NAVI}(q, n)$	the navigational cost related to a node n and a query q when further using the hierarchical index
$cost_{SCAN}(q, n)$	the scanning cost related to a node n and a query q when applying an extended range scan for n

Our reasoning is based on the assumptions that we have a uniform data distribution and uniformly filled nodes. If this is not the case, we can improve the estimation by storing the data distribution, the actual number of directory nodes, and the number of leaf-nodes beneath a certain directory node along with this directory element. For the sake of clarity, we refrain from this more complex approach, and assume that we have a uniform data distribution and that all nodes are uniformly filled.

We will first discuss the cost related to the navigational approach, before we look at the cost related to the scanning approach. Finally, we introduce a combined approach which exploits the advantages of the navigational and the scanning approach.

Navigational Approach. The cost related to a directory node n when using the hierarchical index structure without further modifications for a query q (cf. Figure 59a) consist of an I/O and a CPU part and can be expressed as follows:

$$\text{cost}_{NAVI}(q, n) = \text{cost}_{NAVI}^{I/O}(q, n) + \text{cost}_{NAVI}^{CPU}(q, n)$$

In the following, we will discuss the detailed I/O and CPU cost of the navigational approach.

I/O cost. We have to access $\text{cnt_}n$ directory nodes:

$$\text{cnt_}n_{NAVI}^{I/O}(q, n) = 1 + \sigma(q, n) \cdot \sum_{i=1}^{L(n)-1} m^i$$

Each of these nodes has m entries. For locating these nodes on the disk we use a built-in B^+ -tree which has a height of h_B . Additional to the navigational cost on the B^+ -tree, we have cost related to the reading of $\text{cnt_}t_{NAVI} = m \cdot \text{cnt_}n_{NAVI}^{I/O}$ index entries, i.e. tuples, distributed over $\text{cnt_}t_{NAVI}/b$ disk pages. We penalize each page read with a factor $k_{I/O}$. To sum up, we have the following I/O cost:

$$\text{cost}_{NAVI}^{I/O}(q, n) = k_{I/O} \cdot \left(h_B + \frac{m}{b} \right) \cdot \left(1 + \sigma(q, n) \cdot \sum_{i=1}^{L(n)-1} m^i \right)$$

CPU cost. The CPU cost related to the evaluation of $\text{cnt_}t_{NAVI}$ index entries are:

$$\text{cost}_{NAVI}^{CPU}(q, n) = k_{CPU} \cdot m \cdot \left(1 + \sigma(q, n) \cdot \sum_{i=1}^{L(n)-1} m^i \right)$$

Scanning Approach. If we scan all data belonging to a directory node n on level $L(n)$, the following cost occur:

$$\text{cost}_{SCAN}(q, n) = \text{cost}_{SCAN}^{I/O}(q, n) + \text{cost}_{SCAN}^{CPU}(q, n)$$

The detailed I/O and CPU cost are as follows.

I/O cost. We have to locate the starting point of the scanning area once by using a B^+ -tree. Then, we read $m^{L(n)}$ index entries of the leaf level distributed over $m^{L(n)}/b$ disk pages. Again, we penalize each page read with a factor $k_{I/O}$. To sum up, we have the following I/O cost for the scanning approach:

$$\text{cost}_{SCAN}^{I/O}(q, n) = \left(h_B + \frac{m^{L(n)}}{b} \right) \cdot k_{I/O} \approx \frac{m^{L(n)}}{b} \cdot k_{I/O}$$

CPU cost. The cost related to the evaluation of $m^{L(n)}$ values on the leaf-level are $k_{CPU} \cdot m^{L(n)}$. Thus we have the following CPU cost for the scanning approach:

$$\text{cost}_{SCAN}^{CPU}(q, n) = m^{L(n)} \cdot k_{CPU}$$

Combined Approach. Our approach starts with applying the navigational approach. For each visited node we estimate the navigational and the scanning cost. If $\text{cost}_{SCAN}(q, n) < \text{cost}_{NAVI}(q, n)$, we abort the recursive tree traversal and apply an extended range scan. This mixed approach is a kind of greedy approach, which tries to combine the advantages of the navigational and the scanning approach.

The main point in accurately estimating $\text{cost}_{SCAN}(q, n)$ and $\text{cost}_{NAVI}(q, n)$ is to forecast the overlap-factor σ as precise as possible. For each hierarchical index structure such a selectivity estimation function has to be provided for the optimizer anyway. When the execution plan for a given query q is determined, the optimizer evokes $\sigma(q, n_{root})$ in order to decide whether to include this index in the query execution plan or not. We propose to evoke this selectivity estimation function for each visited directory node in order to decide whether to use the tree directory further or to switch to an extended range scan. Let us note that our approach inherently benefits from a good selectivity estimator which can be used as black box by our new indexing method. Nevertheless, we will present a simple heuristic which aims at estimating the selectivity efficiently and effectively for collision queries on the Relational R-tree. Needless to say that you can also use more sophisticated selectivity estimation functions (cf. Section 4.2) to get better results.

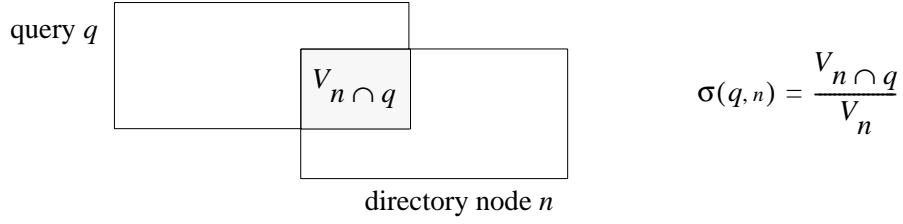


Figure 60: Determination of the overlap-factor σ (RR-tree).

5.3.3 Adaption to the RR-tree

In this section, we adapt the results presented in the foregoing section to the intersect predicate on the Relational R-tree.

The overlap-factor $\sigma(q, n)$ can easily be determined as shown in Figure 60. The overlap-factor $\sigma(q, n)$ is equal to the ratio of the intersection volume $V_{n \cap q}$ between the query object q and the directory node n and the hyper-volume V_n of the directory node.

$$\sigma(q, n) = \frac{V_{n \cap q}}{V_n}$$

As the operation whether two boxes intersect or not can be performed very efficiently, we neglect the CPU cost and concentrate in this section on the accruing I/O cost. Thus we perform an extended index range scan for a directory node n on level $L(n)$ and a query q if

$$\text{cost}_{SCAN}^{I/O} < \text{cost}_{NAVI}^{I/O}$$

i.e.

$$\left(h_B + \frac{m^{L(n)}}{b}\right) < \left(h_B + \frac{m}{b}\right) \cdot \left(1 + \sigma(q, n) \cdot \sum_{i=1}^{L(n)-1} m^i\right)$$

If we assume rather high values of m , a significant overlap factor $\sigma(q, n)$ and a directory level $L(n)$ higher than 2, we scan if the following simplified condition is fulfilled:

$$\frac{m^{L(n)}}{b} < \left(h_B + \frac{m}{b}\right) \cdot \sigma(q, n) \cdot m^{L(n)-1}$$

Or, slightly modified, we scan if:

$$1 < \sigma(q, n) \cdot \left(1 + \frac{b \cdot h_B}{m}\right)$$

If m is equal to b , i.e. we do not use the “supernode” concept of the X-tree [BKK 96] (cf. the discussion about variable fanout of relational index structures in Section 3.6.1), and we assume that we have to perform two reads for navigating through the B^+ -tree directory, it is beneficial to scan if the overlap-factor is higher than $1/3$. Note that the resulting simplified formula is independent of the actual level of the directory nodes.

Let us note, that we can also apply the approach presented in this section to accelerate the Relational M-tree. If nodes are already largely covered by a given range query, it is beneficial not to use the M-tree directory any further, but perform a single range query on the leaf pages of the B^+ -tree (cf. Section 10.3).

5.4 Experimental Evaluation

The tests are based on two test data sets *CAR* and *PLANE*. These test data sets 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* data set consists of approximate 14 million voxels and 200 parts, whereas the *PLANE* data set 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 enumerate the voxels.

We have implemented our approach for the RI-tree, the RQ-tree and the RR-tree on top of the Oracle9i Server using PL/SQL for the computational main memory based programming. 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.

5.4.1 Space-Partitioning Index Structures

Test Data Sets. Figure 61 depicts the interval and gap histograms for our two test data sets. Both test data sets consist of many short intervals and short gaps and only a few longer ones. Consequently, mainly the lowest levels of the RQ-tree and the RI-tree contain index entries.

As shown by Gaede [Gae 95] and Faloutsos et al. [FJM 97], the number of intervals generated via a space-filling curve out of a real-world object mainly depends on

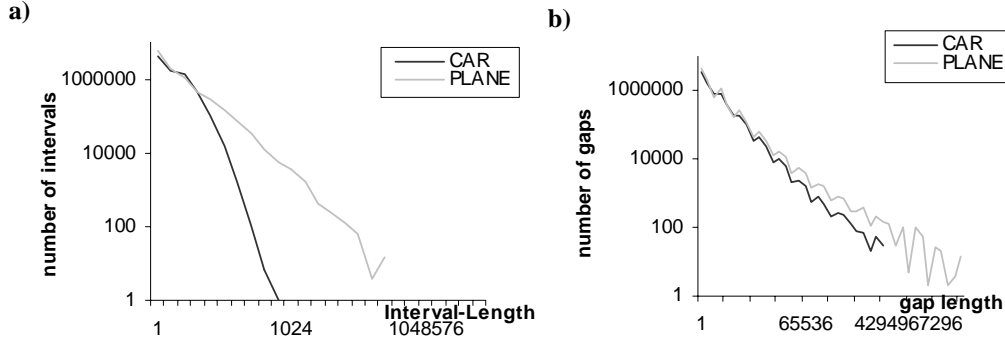


Figure 61: RI-tree histograms.

a) Intervals, b) Gaps

the surface and the shape of the objects and on the resolution of the underlying grid. Unfortunately, there is nothing mentioned about the distribution of the intervals or the corresponding gap distribution. In Chapter 4, it is asserted that the gap histograms show local peaks at gap lengths around 2^{3k} with $k \geq 0$ for 3D data. This behavior is caused by the fact that many gaps represent empty cube-like (3D) regions at the boundary of the spatial objects. Figure 61b, which depicts the gap distributions of our test data sets, supports this assertion.

Figure 62a shows that in the case of the RQ-tree on the CAR data set only the seven lowest of 33 levels are occupied. Similar observations hold for the RI-tree (cf. Figure 62b) 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 data sets but hold for all spatially extended objects [Gae 95][KPPS 03a]. Therefore, the statistics presented in Section 5.2.1 are very beneficial for efficient query processing on spatial objects in general.

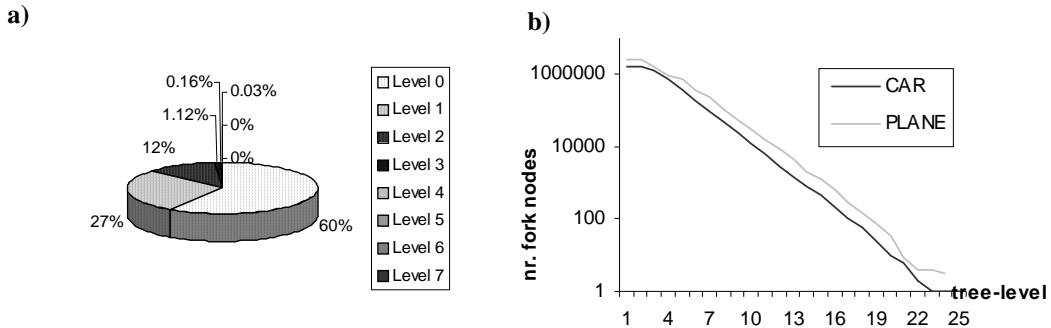


Figure 62: Used index levels.

a) Used tile levels (RQ-tree (CAR)), b) Used *fork-node* levels ((RI-tree) (CAR & PLANE))

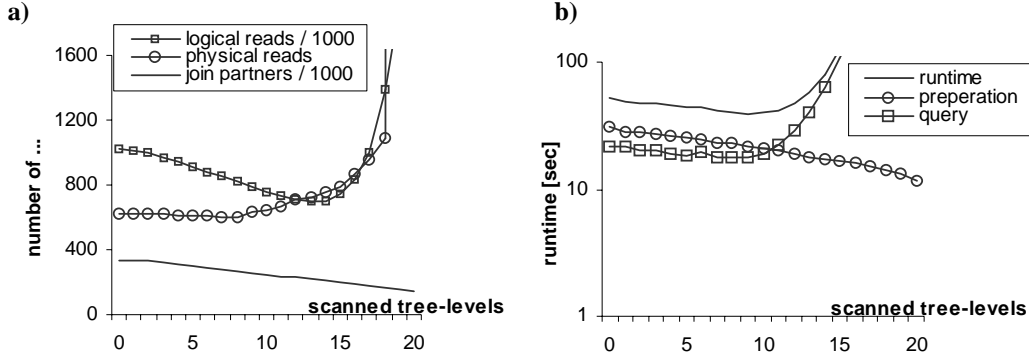


Figure 63: RI-tree optimizations without using statistics.

Query Processing. In the following, we examine the benefits of using extended index range scans for space-partitioning index structures. We first form these extended range scans based on a simple heuristic without using statistics. In the following subsection, we use the presented statistics in order to accelerate the query process. For the RI-tree and the RQ-tree, we used 10% of the database objects as query objects and report the average results from these queries.

In the last subsection, we elaborate the advantages of our selectivity-based decomposition approach by showing that we can accelerate box volume queries by several orders of magnitude when using available statistics for the decomposition of the query objects.

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 5.2.2). 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 63a 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 63b, it is shown that the preparation time decreases with an increasing number of scanned tree levels. Due

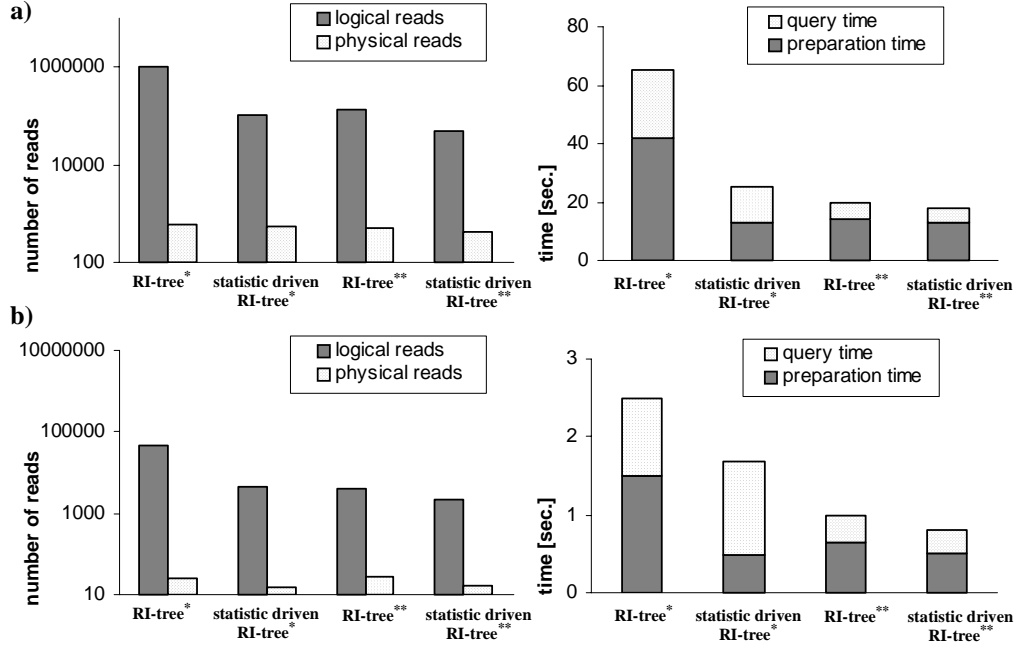


Figure 64: Statistic based acceleration for two variants of the RI-tree.

([KPS 00]^{*}, [KPS 01]^{**})

a) CAR data set, **b)** PLANE data set

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 64, it is shown in detail that our new statistic-based approach accelerates both the basic variant of the RI-tree [KPS 00] and the variant which is optimized for efficient handling of sequences [KPS 01] (cf. Section 3.7.2). Figure 64 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 63). 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.

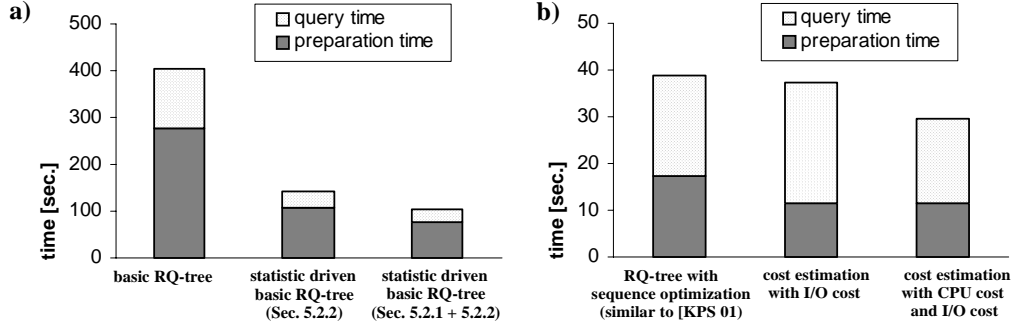


Figure 65: Statistic based accelerated RQ-tree on the CAR data set.

- a) Runtime for the basic RQ-tree, the RQ-tree optimized according to Section 5.2.2, and the RQ-tree optimized according to Section 5.2.1 and 5.2.2,
b) Runtime for the RQ-tree optimized for sequences (similar to [KPS 01])

In the next experiments, we applied the statistic based approach to the RQ-tree (cf. Figure 65). Figure 65a shows that the use of our quantile statistics (cf. Section 5.2.2) accelerates the RQ-tree by 200%. A further improvement can be achieved by using the information of the highest and lowest level of stored tiles within the database (cf. Section 5.2.1), leading to a speed-up of about 300%. Figure 65b depicts the acceleration of the sequence optimized RQ-tree, where we compare the variant without incorporating the CPU cost of the refinement step with the variant including the CPU cost (cf. Section 5.2.2). The first variant considers only the I/O cost and neglects the CPU cost for forming the extended range scan sequences. Figure 65b 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 of the query process by 30% to 300%, 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 the next experiments, we carried out different box volume queries on the RI-tree for the PLANE database. Figure 66 depicts the average runtime for three different boxes, where we moved each box to 10 different locations. As shown in Figure 66, our statistic-based decomposition approach can

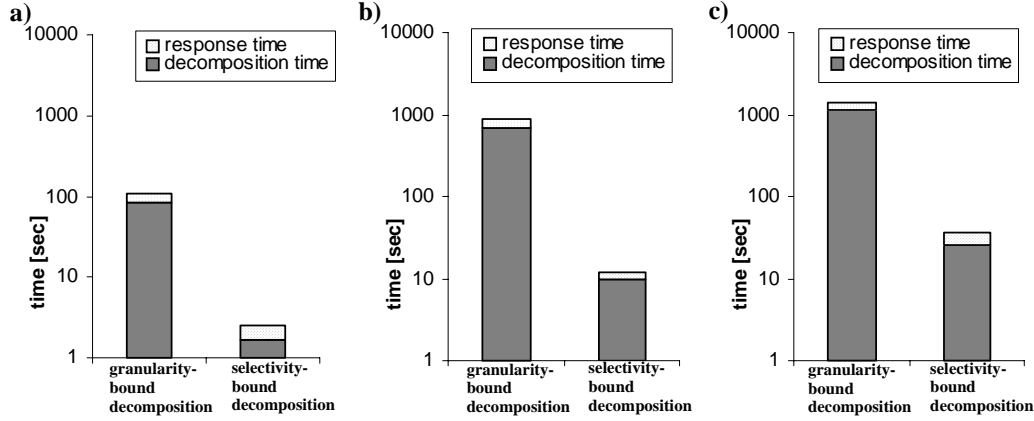


Figure 66: Box queries for the RI-tree on the *PLANE* data set.

(Decomposition and response time).

- a) Box size equals 0.00002% of data space yielding 0.03% selectivity,
- b) Box size equals 0.003% of data space yielding 0.1% selectivity,
- c) Box size equals 0.008% of data space yielding 1.0% selectivity

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 useful for commonly used box volume queries.

5.4.2 Data-Partitioning Index Structures

Test Data Set. The test data set for the following experiments consists of approximately 1,400,000 boxes representing approximately 200 car objects. In order to show how our approach depends on the selectivity of a given query object, we evaluated the RR-tree by using varying box-volume queries.

Query Processing. We applied our new scanning approach proposed in Section 5.3.2 to the Relational R-tree for varying overlap-factors σ which were compared to a full table scan and an unchanged R-tree implementation. Figure 67 shows that the best results for a large range of selectivity parameters was obtained by using an overlap-factor of $\sigma = 1/3$ which is identical to the theoretically derived value (cf. Section 5.3.2). For $\sigma = 0$ an extended range scan is triggered as soon as the query

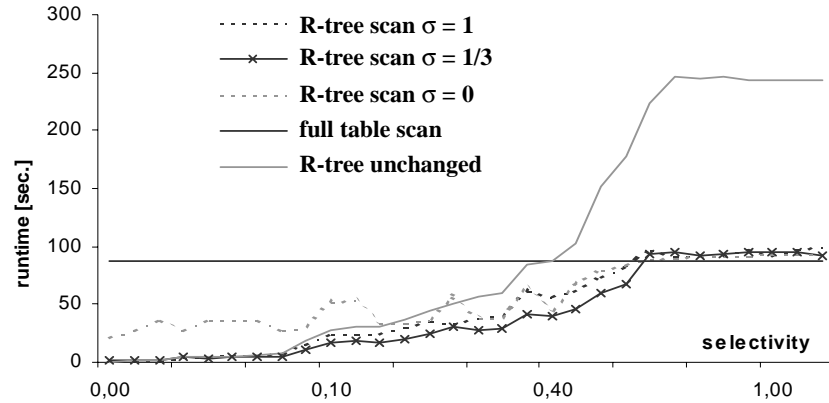


Figure 67: RR-tree for varying selectivity.

box intersects a directory box. This results in rather high query response times for highly selective queries compared to the original R-tree. On the other hand, a parameter value of $\sigma = 1$ forces an extended range scan, if the directory node is completely covered by the query object resulting in rather good query response times over the complete range of selectivity parameters. Figure 67 shows that the decision whether to use the directory of the relational R-tree any longer or to switch to an extended range scan can be decided for each node with negligible overhead. Our combined approach can improve the overall query response time by more than 150% for queries of low selectivity compared to the navigational approach (R-tree). Furthermore, for highly selective queries our combined approach outperforms the sequential scan by more than 10,000%. To sum up, the combined approach naturally adapts to the best of the two worlds, “index” and “sequential scan”.

5.5 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 index structures or are index inherently available. We have implemented our approach for the RI-tree, the RQ-tree as well as for the RR-tree on top of the Oracle9i database system. According to our experiments, we achieved speed-up factors of up to two orders of magnitude. Our new statistic-driven approach accelerates the query processing considerably. This acceleration is

due to the fact, that we can dynamically switch 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.

This statistic-based acceleration approach can fruitfully be applied to time critical applications as for instance the digital mockup process which is based on collision queries for complex spatial objects.

Chapter 6

Cost-based Decompositioning of Complex Spatial Objects

Modern database application 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, and from the design goal to use general purpose database management systems in order to guarantee industrial-strength. In the past two decades, various stand-alone spatial index structures have been proposed but their integration into fully-fledged database systems is problematic. Most of these approaches are based on the decomposition of spatial objects leading to replicating index structures. In contrast to common black-and-white decompositions which suffer from the lack of intermediate solutions, we introduce *gray containers* as a new and general concept. These gray containers are stored in a spatial index structure. Additionally, we store the exact information of these gray containers in a compressed way. The gray containers are created by using a cost-based decompositioning algorithm which takes the access probability and the decompression cost of the gray containers into account. We demonstrate the benefits of our new method for the RR-tree, the RQ-tree and the RI-tree as well as for spatial join processing. The experimental evaluation on real-world test data points out that our new concept leads to an acceleration of up to two orders of magnitude with respect to the overall query response time. The experiments show that our generic approach is especially useful for high resolution spatial data, which is becoming the standard case for modern spatial database applications.

6.1 Introduction

As a common and successful approach, spatial objects can conservatively be approximated by a set of voxels, i.e. cells of a grid covering the complete data space. By means of space filling curves, each voxel can be encoded by a single integer and, thus, an extended object is represented by a set of enumerated voxels. As a principal design goal, space filling curves achieve good spatial clustering properties since cells in close spatial proximity are encoded by contiguous integers. As explained in Chapter 2, adjacent cell values can be grouped together to black intervals, black tiles or black boxes which are basic datatypes for spatial applications. By expressing spatial region queries as intersections of these spatial primitives, vital operations for CAD applications can be supported. As outlined in Chapter 3, a seamless and capable integration of spatial indexing into industrial-strength databases is essential.

Besides the integration into fully-fledged database systems, a further new requirement for large spatial objects, including cars, planes or space stations, is a high approximation quality which is primarily influenced by the resolution of the grid covering the data space. High resolution spatial objects may consist of several hundreds of thousands of voxels. Although the voxels can further be grouped together to black intervals, black tiles or black boxes, the number of the resulting spatial primitives still remains very high. On the other hand, one-value approximations of spatially extended objects often are far too coarse. In many applications, GIS or CAD objects feature a very complex and fine-grained geometry. The rectilinear bounding box of the brake line of a car, for example, would cover the whole bottom of the indexed data space. A non-replicating storage of such data causes region queries to produce too many false hits that have to be eliminated by subsequent filter steps.

In this chapter, we introduce a cost-based decompositioning algorithm for complex spatial objects which helps to range between the two extremes of one-value approximations and the use of unreasonably many approximations. Our new approach takes compression algorithms for the effective storage of decomposed spatial objects and access probabilities of these decompositions into account.

The remainder of this chapter is organized as follows. In Section 6.2, we shortly review the related work in the area of object decompositioning. In Section 6.3, we introduce gray container objects, which can be stored within a spatial index. Furthermore, we discuss in detail our cost-based grouping algorithm which can be used

together with arbitrary packing algorithms. In Section 6.4, we discuss how intersection queries based on compressed gray containers can be posted on top of the SQL engine. In Section 6.5, we adapt the presented techniques to the efficient processing of spatial joins. In Section 6.6, we present the empirical results, which are based on two real-world test data sets of our industrial partners, a German car manufacturer and an American plane producer, dealing with high resolution voxelized CAD data. We resume our work in Section 6.7 and close with a few final remarks on future work.

6.2 Related Work

In this section we will shortly discuss different aspects related to an *effective decomposition of complex spatial objects*.

Complex Spatial Objects. Gaede pointed out that the number of voxels representing a spatially extended object exponentially depends on the granularity of the grid approximation [Gae 95]. Furthermore, the extensive analysis given in [MJFS 96] and [FJM 97] shows that the asymptotic redundancy of an interval- and tile-based decomposition is proportional to the surface of the approximated object. Thus, in the case of large high-resolution parts, e.g. wings of an airplane, the number of tiles or intervals can become unreasonably high.

Decompositioning Algorithm. In [SK 93], 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 in $O(\sqrt{n})$ many index entries. As this root-criterion was designed for 2D polygons and was not based on any analytical reasoning, it cannot be adapted to complex 3D objects. In this chapter, in contrast, we will present an analytical cost-based decomposition approach which can be used for all kinds of spatially extended objects.

6.3 Decompositioning of High-Resolution Spatial Objects

High resolution spatial objects may consist of several hundreds of thousands of voxels (cf. Figure 68a). For each object, there exist a lot of different possibilities to decompose it into approximations by grouping numerous voxels together. We call these groups gray containers throughout the rest of this chapter. In the one-dimensional case, we call the gray containers gray intervals. In the multi-dimensional case, they are called gray boxes (cf. Figure 68b). Informally spoken, gray containers bridge the gap between black containers; a formal definition follows later. The question at issue is, which grouping is most suitable for efficient query processing. A good grouping should take the following “*grouping rules*” into consideration:

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

The first rule guarantees that the number of index entries is small, as the hulls of the gray containers, i.e. the minimum bounding hyper-rectangles covering the complete gray containers, are stored in appropriate index structures, e.g. the RI-tree, the RR-tree, or the RQ-tree (cf. Figure 68d). The second rule guarantees that many unnecessary candidate tests can be omitted, as the number and size of gaps included in the gray container is small. Finally, the third rule guarantees that a candidate test can be carried out efficiently. A good query response behavior results from an optimum trade-off between these grouping rules.

In the remainder of this section, we will introduce a cost-based grouping algorithm which finds an optimum trade-off between these grouping rules. In Section 6.3.1, we first introduce our gray containers formally. Section 6.3.2 is dedicated to the storage of these gray containers. In Section 6.3.3, we discuss why it is beneficial to store the gray containers in a compressed way. Furthermore, we introduce a new spatial packer, called QSDC. In Section 6.3.4, we introduce our cost-based grouping algorithm for complex spatial objects.

6.3.1 Gray Containers

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

Definition 15 (*Voxelized Object*).

Let O be the domain of all object identifiers and let $id \in O$ be an object identifier. Furthermore, let IN^d be the domain of d -dimensional points. Then we call a pair $O_{voxel} = (id, \{v_1, \dots, v_n\}) \in O \times 2^{IN^d}$ a d -dimensional voxelized object. We call each of the v_i an object voxel, where $i \in \{1, \dots, n\}$.

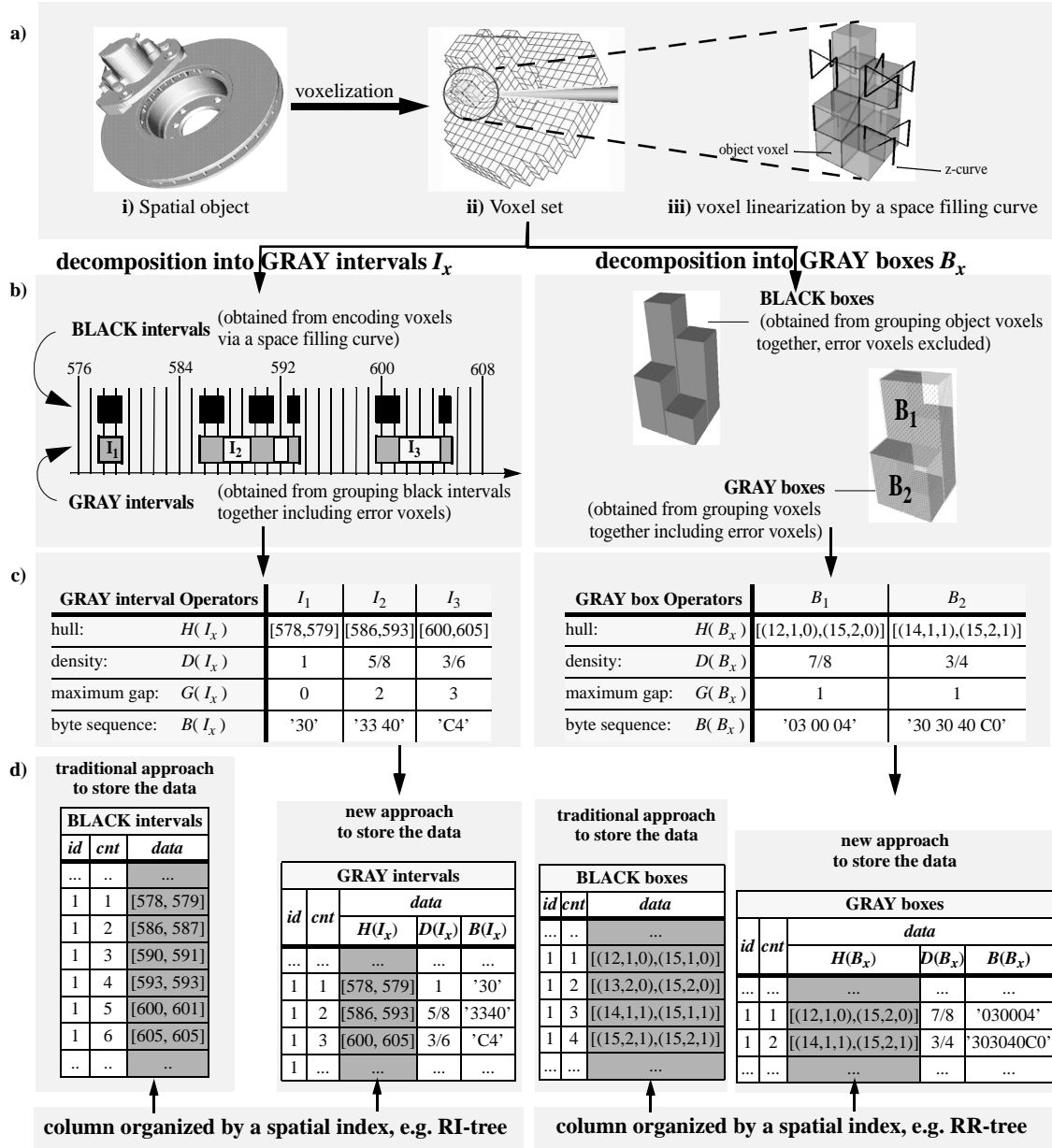
A voxelized object consists of a set of d -dimensional points, which can be naturally ordered in the one-dimensional case. If $d > 1$, such an ordering does not longer exist. By means of space filling curves, $p: IN^d \rightarrow IN$, all multidimensional voxelized objects can be mapped to one-dimensional voxelized objects (cf. Figure 68a (iii) and left side of Figure 68b).

Obviously, we can group adjacent voxels together to d -dimensional hyper-rectangles and store these hyper-rectangles in appropriate index structures. If we conservatively approximate an object by only one hyper-rectangle, numerous *error voxels* are included in this approximation, which requires an expensive refinement step during the query process. A promising trade-off between the filter and refinement step can be found by means of gray containers. Intuitively, a gray container sequence is a covering of our voxelized object by means of hyper-rectangular boxes, where each voxel is assigned to exactly one gray container (cf. Figure 68b).

Definition 16 (*Gray Container, Gray Container Sequence*).

Let $(id, \{v_1, \dots, v_n\})$ be a d -dimensional voxelized object, and let π be a bijective function $\pi: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$. Moreover, let $m \leq n$ and $0 = i_0 < i_1 < i_2 < \dots < i_m = n \in IN^+$. Then we call $O_{gray} = (id, \langle \{v_{\pi(i_0+1)}, \dots, v_{\pi(i_1)}\}, \dots, \{v_{\pi(i_{m-1}+1)}, \dots, v_{\pi(i_m)}\} \rangle)$ a *gray container sequence*. We call each of the $j = 1, \dots, m$ groups $C_j = \{v_{\pi(i_{j-1}+1)}, \dots, v_{\pi(i_j)}\}$ of O_{gray} a *gray container*.

In the following, we present operators for gray containers which enable us to introduce our approach formally. Throughout this chapter, we refer repeatedly to the definitions summarized in Table 2 which assumes that $C_{gray} = \{v_1, \dots, v_c\}$ is a d -dimensional gray container and $p: IN^d \rightarrow IN$ an arbitrary space filling curve. For clar-

**Figure 68:** Gray containers.

a) Voxelized spatial object, b) Black and gray containers,
c) Operators on gray containers, d) Storage of gray containers

ity, Figure 68c demonstrates the values of the most important operators for gray container intervals and boxes. Intuitively, the byte sequence $B(C_{gray})$ describes the enumerated set of voxels as byte-string which might contain long sequences of zero bytes "00".

Operator	Description and Definition
	volume
$V(C_{gray})$	$\prod_{k=1}^d (\max(v_i^k, i \in 1 \dots c) - \min(v_i^k, i \in 1 \dots c) + 1)$
	number of black voxels
$N_b(C_{gray})$	c
	number of white voxels
$N_w(C_{gray})$	$V(C_{gray}) - N_b(C_{gray})$
	density
$D(C_{gray})$	$N_b(C_{gray}) / V(C_{gray})$
	hull
$H(C_{gray})$	$\bigcap_{k=1}^d [h_l^k, h_u^k] = \bigcap_{k=1}^d [H_l^k(C_{gray}), H_u^k(C_{gray})] =$ $\bigcap_{k=1}^d [\min(v_i^k, i \in 1 \dots c), \max(v_i^k, i \in 1 \dots c)]$
	maximum gap
$G(C_{gray})$	$\max \left(\prod_{k=1}^d (u^k - l^k + 1), \forall k \in \{1 \dots d\} (h_l^k \leq l^k \leq u^k \leq h_u^k) \wedge \right.$ $\left. \forall i \in \{1 \dots c\} \exists k \in \{1 \dots d\} (v_i^k < l^k \vee u^k < v_i^k) \right)$
	set of all voxels
$S(C_{gray})$	$\{v \in IN^d, \forall k \in \{1 \dots d\} : (h_l^k \leq v^k \leq h_u^k)\}$
	byte sequence
$B(C_{gray})$	$\langle s_0, \dots, s_n \rangle$ where $0 \leq s_j < 2^8$, $n = \lfloor (\max(\rho(v_i), i \in 1 \dots c))/8 \rfloor - \lfloor (\min(\rho(v_i), i \in 1 \dots c))/8 \rfloor$ $s_j = \sum_{k=0}^7 2^{7-k} \begin{matrix} \text{if } (\exists v \in \{v_1, \dots, v_c\} : (\rho(v) = \lfloor (\min(\rho(v_i), i \in 1 \dots c))/8 \rfloor \cdot 8 + 8j + k)) \\ \text{otherwise} \end{matrix}$

Table 2: Operators on the gray container C_{gray}

In the following, we confine ourselves to non-overlapping gray approximations, which are more suitable for efficient query processing.

Definition 17 (*Non-Overlapping Gray Containers*).

Let $O_{gray} = (id, \langle C_1, \dots, C_m \rangle)$ be a gray container sequence. O_{gray} is *non-overlapping*, iff

$$\forall i, j \in \{1 \dots m\} \forall v \in IN^d : (i \neq j) \wedge (v \in S(C_i)) \Rightarrow v \notin S(C_j)$$

6.3.2 Storing Gray Containers in an ORDBMS

The traditional approach for storing an object id in a database is to map its black containers to a set of tuples in an object-relational table *BlackContainers* (\underline{id} , \underline{cnt} , $data$) (cf. Figure 68d). The primary key is formed by the object identifier id and a unique number cnt for each black container. If we map our 3D spatial object via space-filling curves onto one dimension, the black container is an interval. Otherwise, it is a 3D bounding box which contains no error voxels. A spatial index on the attribute $data$ supports efficient query processing. For high resolution spatial data, this approach yields a high number of table and index entries and, consequently, leads to a critical query response behavior.

A key idea of our approach is to store the non-overlapping gray container sequence $O_{gray} = (id, \langle C_1, \dots, C_m \rangle)$ in a set of m tuples in an object-relational table *GrayContainers* (\underline{id} , \underline{cnt} , $data$) (cf. Figure 68d). Again, the primary key is formed by the object identifier id and a unique number cnt for each gray container. The set of voxels $\{v_1, \dots, v_c\}$ of each gray container $C_{gray} = \{v_1, \dots, v_c\}$ is mapped to the complex attribute $data$ which consists of aggregated information, i.e. the hull $H(C_{gray})$ and the density $D(C_{gray})$, and a byte sequence $B(C_{gray})$ containing the complete information of the voxel set. In order to guarantee efficient query processing, we apply spatial index structures on $H(C_{gray})$ and store $B(C_{gray})$ in a compressed way within a binary large object (BLOB). $H(C_{gray})$ is a multi-dimensional bounding box having a volume $V(C_{gray})$. If we map our 3D spatial object via space-filling curves onto one dimension, $H(C_{gray})$ is a gray interval which can be managed by an index structure suitable for intervals, e.g. the RI-tree (cf. left side of Figure 68d). Otherwise, it is a 3D gray bounding box which can efficiently be indexed by using for instance the RR-tree (cf. right side of Figure 68d). The two important advantages of this approach are as follows: First, the number of table and index entries can be controlled and reduced dramatically. Secondly, the access to the gray containers is efficiently supported by established relational access methods, e.g. the RI-tree, the RQ-tree, or the RR-tree. These access methods have to be created on $H(C_{gray})$.

There are two different problems related to the storage of gray container sequences: the *compression* problem and the *grouping* problem which will be discussed in the following two sections.

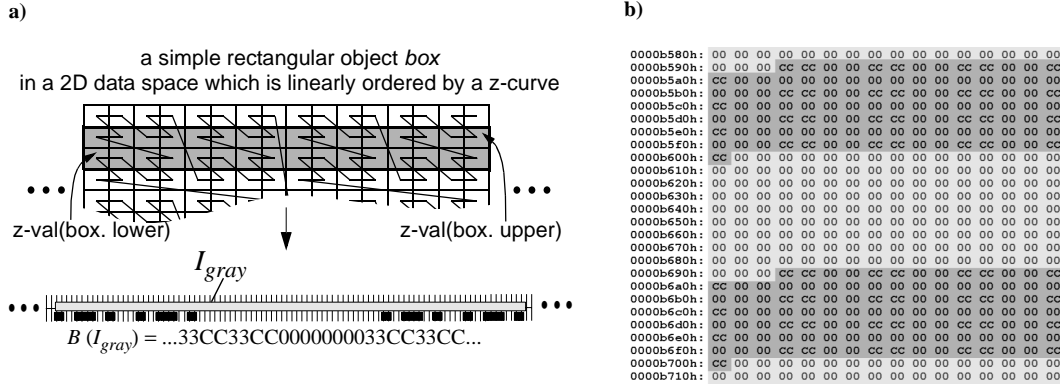


Figure 69: Patterns.

- a) Voxelized object linearized by using a space-filling curve,
b) Extract from $B(C_{gray})$ viewed in a Hex-Editor

6.3.3 Compression of Gray Containers

In this section, we first motivate the use of packers, by showing that $B(C_{gray})$ contains patterns. Therefore, $B(C_{gray})$ can efficiently be shrunk down by using data compressors. After discussing the properties which a suitable compression algorithm should fulfill, we review some of the packers introduced in the literature. Then, we introduce a new effective packer which exploits gaps and patterns included in the byte sequence $B(C_{gray})$ of our gray container C_{gray} .

Patterns. To describe a rectangle in a 2D vector space we only need 4 numerical values, e.g. we need two 2 dimensional points. In contrast to the vector representation, an enormous redundancy is contained in the corresponding voxel sequence of this object as depicted in Figure 69a. As space filling curves enumerate the data space in a structured way, we can find such “structures” in the resulting voxel sequence representing simply shaped objects. We can pinpoint the same phenomenon not only for simply shaped parts but also for more complex real-world spatial parts. Assuming we cover the whole voxel sequence of an object id by one container, i.e. $O_{gray} = (id, \langle C_{gray} \rangle)$, and survey its byte representation $B(C_{gray})$ in a hex-editor, we can notice that some byte sequences occur repeatedly. This phenomenon is depicted in Figure 69b for one complex object from our test data sets. We will now discuss how these patterns can be used for the efficient storage of gray containers in an ORDBMS.

Compression Rules. The voxel set $\{v_1, \dots, v_c\}$ belonging to a gray container $C_{\text{gray}} = \{v_1, \dots, v_c\}$ can be materialized and stored in a BLOB in many different ways. A good materialization should consider two “*compression rules*”:

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

A good query response behavior is based on the fulfillment of both aspects. The first rule guarantees that the I/O cost $c_{I/O}^{BLOB}$ are relatively small whereas the second rule is responsible for low CPU cost c_{CPU}^{BLOB} . The overall cost $c^{BLOB} = c_{I/O}^{BLOB} + c_{CPU}^{BLOB}$ for the evaluation of a BLOB is composed of both parts. Unfortunately, these two requirements are not necessarily in accordance with each other. If we compress the byte sequence $B(C_{\text{gray}})$, we can reduce the demand of secondary storage and consequently $c_{I/O}^{BLOB}$. Unfortunately, c_{CPU}^{BLOB} might rise because we first have to decompress the data before we can evaluate it. On the other hand, if we store $B(C_{\text{gray}})$ without compressing it, $c_{I/O}^{BLOB}$ might become very high whereas c_{CPU}^{BLOB} might be low. Furthermore, a good update behavior also depends on the fulfillment of both rules.

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

Related Work. In this section, we will shortly describe some of the most prominent lossless decompression techniques. For a more detailed survey on lossless and lossy compression techniques, we refer the reader to [RH 93] and [SN 02].

Run-Length Coding. Data often contains sequences of identical bytes. By replacing these repeated byte sequences by the number of occurrences, a substantial reduction of data can be achieved. This is known as run-length coding. A variant of this approach was pursued in [KPPS 03a], which is called differential compression in [RH 93].

Pattern Substitution. This technique substitutes single characters for patterns that occur frequently. The eliminated patterns are often stored in a separate dictionary. A widespread pattern substitution algorithm is LZ77 [LZ 77]. This compression algorithm detects sequences of data that occur repeatedly by using a *sliding window*. An n byte sliding window is a record of the last n characters in the input respectively the output data stream. If a sequence of characters is identical to one that can be found

within the sliding window, the current sequence is replaced by two numbers: a distance value, indicating the starting point of the found sequence within the sliding window, and a length value, corresponding to the maximum number of characters for which the two sequences are identical.

Statistical Coding. There is no fundamental reason that different characters need to be coded by a fixed number of bits. For instance, in the case of morse code, frequently occurring characters are coded with short strings, while rarely occurring characters are coded using longer strings. Such statistical coding depends on the frequency of individual characters or byte sequences. There are different techniques based on statistical coding, e.g. arithmetic coding [SN 02] and Huffman coding.

Huffman Coding. Given the characters that have to be encoded, together with their probabilities of occurrence, the Huffman algorithm determines the optimal coding using the minimum number of bits [Huf 52]. The resulting Huffman coding table is necessary for the compression and decompression process and has to be stored along with the encoded data.

ZLIB. This popular approach compresses data first with *LZ77*, followed by Huffman coding. A compressed data set consists of a series of chunks, corresponding to successive chunks of input data. The chunk sizes are arbitrary. Each chunk has its own Huffman tree, whereas the *LZ77* algorithm is not confined to these chunks and may refer to identical byte sequences in previous chunks [Deu 96].

BZIP2. This rather new approach implements the Burrows-Wheeler transform (*BWT*) followed by Move To Front (*MTF*) transformation and Huffman coding. The *BWT* algorithm takes a chunk of data and applies a sorting algorithm to it. The rearranged output chunk contains the same characters in a different order. The original ordering of the characters can be restored, i.e. the transformation is reversible [BW 94]. *MTF* is a transformation algorithm which does not compress data but can help to reduce redundancy, especially after a *BWT* where the data is likely to contain a lot of repeated characters.

Spatial Compression Techniques. In this section, we look at new specific compression techniques, which are designed for storing the voxel set of a gray container in a BLOB. The first technique is only suitable for simply structured objects whereas our new quick spatial data compressor (QSDC) is generally applicable.

Simply Structured Objects. Simply structured objects can often be described by a few parameters. For instance, a rectilinear box can be described by two three-

dimensional points, and a sphere can be described by one three-dimensional point and one additional numerical value. These few parameters can be stored in the BLOB of each gray container instead of the exact voxel set. They contain the whole information in the most compressed way. Storing the exact geometric parameters in the BLOB, instead of the inflated voxel set, forms a very efficient type of spatial compression technique, which leads to minimal I/O cost. We can create $B(C_{gray})$ at any time from this simple and compact geometric information. For elementary objects, as for instance spheres, tubes or boxes, this can be done efficiently by means of simple mathematical functions. As outlined in Section 5.2.3, this compression technique forms the foundation of the efficient processing of box volume queries.

Quick Spatial Data Compressor (QSDC). The *QSDC* algorithm is especially designed for high resolution spatial data and includes specific features for the efficient handling of patterns, gaps and extremely long gaps. It is optimized for speed and does not perform time intensive computations as for instance Huffman compression. *QSDC* is a derivation of the LZ77 technique. However, it compresses data in only one pass and much faster than other Lempel-Ziv based compression schemes as for example *SEQUITUR* [NW 97], *RAY* [CWZ 99] or *XRAY* [CW 00].

QSDC operates on two main memory buffers. The compressor scans an input buffer for patterns, gaps and extremely long gaps (cf. Figure 70). *QSDC* replaces the patterns with a two- or three-byte compression code, the gaps with a one- or two-byte compression code and the extremely long gaps with a six byte compression code. Then it writes the code to an output buffer. *QSDC* packs an entire BLOB in one piece, the input is not split into smaller chunks. At the beginning of each compression cycle *QSDC* checks if the end of the input data has been reached. If so, the compression stops. Otherwise another compression cycle is executed. Each pass through the cycle adds one item to the output buffer, either a compression code or a non-compressed character. Unlike other data compressors, no checksum calculations are performed to detect data corruption because the underlying ORDBMS ensures data integrity.

The decompressor reads compressed data from an input buffer, expands the codes to the original data, and writes the expanded data to the output buffer. When an extremely long run-length sequence occurs, the actual output buffer containing the decompressed data is returned to the calling process, and a new output buffer is allocated.

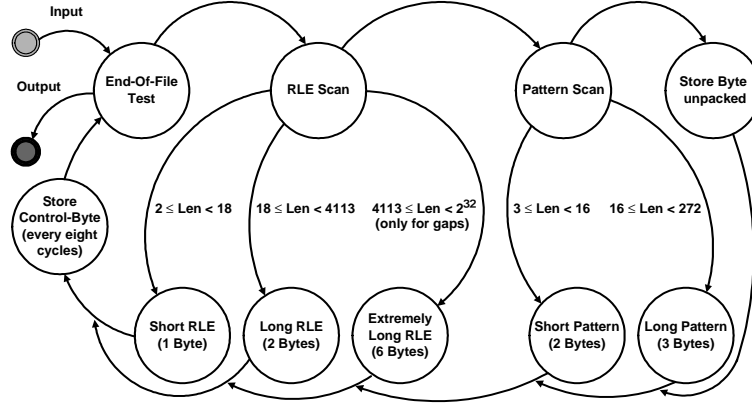


Figure 70: Flow diagram of QSDC compression algorithm.

Lemma 2. The worst packed vs. unpacked ratio of QSDC for an input sequence of L bytes is $(\lceil L/8 \rceil + L)/L$.

Proof. The *QSDC* packing algorithm makes sure that a compression code is always shorter than the original unpacked sequence. This means the only situation where the output can grow is when the packer encounters unpackable bytes. An input byte which cannot be packed is directly passed through. Plus, an additional bit is consumed by the control byte. This sums up to L control bits, i.e. $\lceil L/8 \rceil$ control bytes, which proves the lemma. ■

The *QSDC* algorithm achieves a much better worst case compression ratio than the highly sophisticated *BZIP2* and *ZLIB* compressors, which store additional information in the stream header of the compressed data. For instance, the *ZLIB* compressor has a worst case ratio of 12:1 [RG 02]. This worst case behavior is especially important for the storage of small BLOBs. Not only does the *QSDC* approach achieve a good worst case compression ratio, but it also reaches a high unpack speed. The high unpack speed of our algorithm is due to the fact, that the entire code of the decompression function compiles down to about 1000 bytes 80x86 assembly code, fitting into the first level cache of all modern 80x86 processors which reduces the instruction cache misses to a minimum. Furthermore, *QSDC* benefits from the fact that the compression codes are byte aligned.

For more detail we refer the reader to [Kun 02].

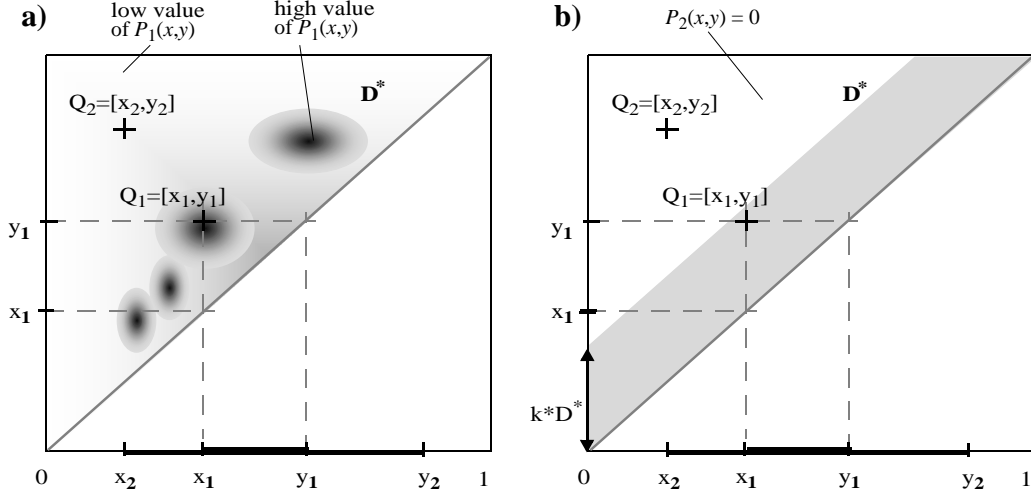


Figure 71: Query distribution functions $P_i(x,y)$.

a) Complex query distribution $P_1(x,y)$,

b) Simple query distribution $P_2(x,y)$

6.3.4 Grouping into Gray Containers

Our grouping algorithm takes the expected access cost of the gray containers into account. The expected cost $cost(C_{gray})$ related to a gray container C_{gray} depend on the average access probability of C_{gray} and on the cost related to the evaluation of the exact byte sequence $B(C_{gray})$.

First, the access probability is computed by assuming that we know the average query distribution for each dimension. Then, the evaluation cost are introduced which heavily depend on the used data compressor. Finally, our cost-based grouping algorithm *GroupCon* is introduced which is used for storing complex objects in an ORDBMS.

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

First, we transform an arbitrary d -dimensional box query into a $2 \times d$ -dimensional normalized data space D^* (cf. Figure 71 for one-dimensional query intervals Q_i). We start with normalizing the coordinates of our d -dimensional query container to ensure

that all data lies within the hyper cuboid $X_{i=1}^d [0, 1]$. For clarity, we will first examine the one-dimensional case looking at intervals and their point transformation into the upper triangle $D^* := \{(x, y) \in [0, 1]^2 \mid x \leq y\}$ of the two-dimensional hyper cuboid. An interval $Q = [x, y]$ therefore corresponds to the point (x, y) with $x \leq y$. Examples are visualized in Figure 71. To each of these two-dimensional points $Q=(x,y)$ we assign a numerical value $P(Q)$ where $0 \leq P(Q) \leq 1$ holds. As the probability is equal to one that a query is somewhere located in the upper triangle D^* , the following equation has to hold:

$$\int \int_{D^*} P(x, y) dx dy = 1$$

Figure 71 shows two different query distribution functions. A potential query Q_2 is very unlikely in Figure 71a and does not occur at all in Figure 71b. On the other hand, query Q_1 is very likely in both cases.

Let us note, that we used the simple query distribution function of Figure 71b throughout our experiments. In all considered application areas the common query objects only comprise a very small portion of the data space D^* . Therefore, we introduce the parameter k^* , which restricts the extension of the possible query objects. For the computation of the access probability we only consider query objects whose extensions do not exceed $k^* \cdot D^*$ in each dimension.

Access Probability. The *access probability* $P(C_{gray})$ related to a container object C_{gray} denotes the probability that an arbitrary query object has an intersection with the d -dimensional hull $H(C_{gray})$. All possible query intervals that intersect C_0 are visualized by the shaded area $A(C_0)$ in Figure 72a. The area displays all intervals whose lower bounds are smaller or equal to b and whose upper bounds are larger or equal to a . These query intervals are exactly the ones that have a non empty intersection with C_0 . The probability that an interval $C_0 = [a_0, b_0]$ is intersected by an arbitrary query interval is:

$$P(C_0) = \frac{\int \int_{A(C_0)} P(x, y) dx dy}{\int \int_{D^*} P(x, y) dx dy} = \int \int_{A(C_0)} P(x, y) dx dy$$

Assuming that the dimensions of the data space are independent from each other, the derived access probability for the one-dimensional data space can easily be expanded to an arbitrary number of dimensions. The probability for the multi-dimen-

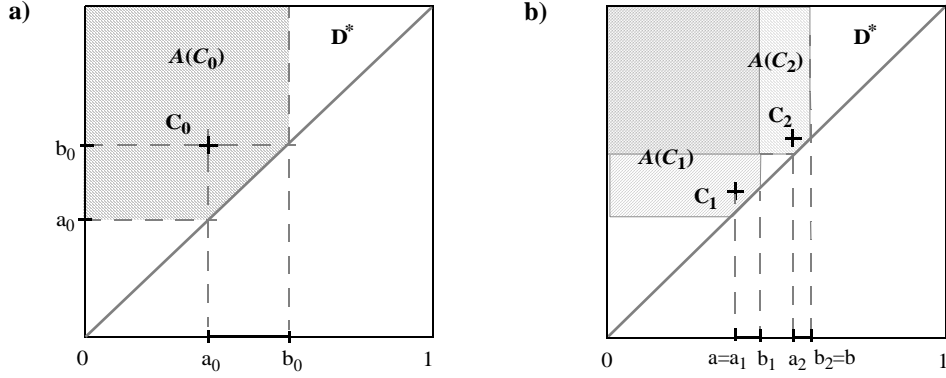


Figure 72: Computation of average access probabilities of gray containers.

a) Intersection area for the one-dimensional container $C_0=[a_0, b_0]$,

b) Intersection area for the decomposed container objects C_1 and C_2

sional case is equal to the product of all one-dimensional probabilities which can be derived for each dimension individually.

Evaluation Cost. Furthermore, the expected query cost depend on the cost related to the evaluation of the byte sequence stored in the BLOB of an intersected gray container C_{gray} . The evaluation of the BLOB content requires to load the BLOB from disk and decompress the data. Consequently, the evaluation cost depends on both the size $V(C_{gray})$ of the uncompressed BLOB and the size $V_{comp}(C_{gray}) \ll V(C_{gray})$ of the compressed data. Additional, the evaluation cost $cost_{eval}$ depend on a constant $c_{load}^{I/O}$ related to the retrieval of the BLOB from secondary storage, a constant c_{decomp}^{cpu} related to the decompression of the *BLOB*, and a constant c_{test}^{cpu} related to the intersection test. The cost c_{decomp}^{cpu} and $c_{load}^{I/O}$ heavily depend on how we organize $B(C_{gray})$ within our BLOB, i.e. on the used compression algorithm. A highly effective but not very time efficient packer, e.g. *BZIP2*, would cause low loading cost but high decompression cost. In contrast, using no compression technique, leads to very high loading cost but no decompression cost. Our *QSDC* is an effective and very efficient compression algorithm which yields a good trade-off between the loading and decompression cost. Finally, c_{test}^{cpu} solely depend on the used system. The overall evaluation cost are defined by the following formula:

$$cost_{eval}(C_{gray}) = V_{comp}(C_{gray}) \cdot c_{load}^{I/O} + V(C_{gray}) \cdot (c_{decomp}^{cpu} + c_{test}^{cpu})$$

```

ALGORITHM GroupCon ( $C_{gray}$ ,  $P$ )
BEGIN
     $container\_list := split\_at\_maximum\_gap(C_{gray})$ ;
     $cost_{gray} := P(C_{gray}) \cdot cost_{eval}(C_{gray})$ ;
     $cost_{dec} := 0$ ;
    FOR EACH  $c$  in  $container\_list$  DO
         $cost_{dec} := cost_{dec} + P(c) \cdot cost_{eval}(c)$ ;
    END FOR;
    IF  $cost_{gray} > cost_{dec}$  THEN
        FOR EACH  $c$  in  $container\_list$  DO
            GroupCon ( $c$ ,  $P$ );
        END FOR;
    ELSE
        report ( $C_{gray}$ );
    END IF;
END.

```

Figure 73: Grouping algorithm *GroupCon*.

Grouping Algorithm. Orenstein [Ore 89] introduced the size- and error bound decomposition approach. Our first grouping rule “the number of gray 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 gray containers should be small”. For fulfilling both rules, we introduce the following top-down grouping algorithm for gray containers, called *GroupCon* (cf. Figure 73). *GroupCon* is a recursive algorithm which starts with an approximation $O_{gray} = (id, \langle C_{gray} \rangle)$, i.e. we approximate the object by one gray container. In each step of our algorithm, we look for the maximum gap g within the bounding box of the actual gray container. We carry out the split along this gap, if the average query cost caused by the decomposed containers is smaller than the cost caused by our input container C_{gray} . The expected cost related to a gray container C_{gray} can be computed as described in the foregoing paragraph. A gray container which is reported by the *GroupCon* algorithm is stored in the database and no longer taken into account in the next recursion step. Data compressors which have a high compression rate and a fast decompression method, result in an early stop of the *GroupCon* algorithm generating a small number of gray intervals. Our experimental evaluations suggest that this grouping algorithm yields results which are very close to the optimal ones for many combinations of index-structures, data compression techniques and data space resolutions.

6.4 Object Relational Query Processing

In contrast to the last section, where we focused on organizing the object within the database, we turn our attention in this section to the query process. We first discuss when two objects intersect, and how we can express this intersect-operator on top of the SQL engine. In Section 6.4.2, we introduce useful optimizations for the discussed intersect predicate. We concentrate on the intersect predicate, as it is the most important predicate for the digital mockup process of complex spatial objects.

6.4.1 The *intersect* SQL Statements

In this section we first define formally the *intersect_{boolean}* predicate, which indicates whether two complex objects share at least one voxel. The *intersect_{ranked}* predicate measures the intersection volume, and the *interlace* predicate indicates whether there exist two gray containers with intersecting hulls.

Definition 18 (*Object Intersection*).

Let $O_{\text{voxel}} = (id, \{v_1, \dots, v_n\})$ and $O'_{\text{voxel}} = (id', \{v'_1, \dots, v'_{n'}\})$ be two d -dimensional voxelized objects of the domain O . Furthermore, let $O_{\text{gray}} = (id, \langle C_1, \dots, C_m \rangle)$ and $O'_{\text{gray}} = (id', \langle C'_1, \dots, C'_{m'} \rangle)$ be two corresponding gray container sequences.

Then we define the operators *intersect_{boolean}*: $O \times O \rightarrow \text{boolean}$, *intersect_{ranked}*: $O \times O \rightarrow IN$, and *interlace_{boolean}*: $O \times O \rightarrow \text{boolean}$ as follows:

- $\text{intersect}_{\text{boolean}}(O_{\text{voxel}}, O'_{\text{voxel}}) = \text{true} \Leftrightarrow \exists i \in \{1, \dots, n\}, \exists j \in \{1, \dots, n'\} : v_i = v'_j.$
- $\text{intersect}_{\text{ranked}}(O_{\text{voxel}}, O'_{\text{voxel}}) = |\{i | i \in \{1, \dots, n\}, \exists j \in \{1, \dots, n'\} : v_i = v'_j\}|.$
- $\text{interlace}_{\text{boolean}}(O_{\text{voxel}}, O'_{\text{voxel}}) = \text{true} \Leftrightarrow \exists i \in \{1, \dots, m\}, \exists j \in \{1, \dots, m'\} : S(C_i) \cap S(C'_j) \neq \emptyset.$

As we defined gray container sequences as a conservative approximation of voxelized objects, we can use the hulls of the containers in a first conservative filter step.

Thereby, we can take advantage of standard spatial access methods. As shown in Section 6.3.1, the gray container sequences can be mapped to an object-relational schema *GrayContainers* (*id*, *cnt*, *data*) (cf. Figure 68d). Following this approach, we can easily express the intersect predicates on top of the SQL engine (cf. Figure 74).

In the case of the *intersect_{boolean}* predicate (cf. Figure 74a), we use *table* as a nesting function that groups references of gray query and database pairs together,

- a)
- ```

SELECT candidates.id FROM
(
 SELECT db.id AS id, table (pair(db.rowid, q.rowid)) AS ctable
 FROM GrayContainers db, :GrayQueryContainers q
 WHERE intersect (hull(db.data), hull(q.data))
 GROUP BY db.id) candidates
WHERE EXISTS
(
 SELECT 1
 FROM GrayContainers db, :GrayQueryContainers q, candidates.ctable ctable
 WHERE db.rowid = ctable.dbrowid AND q.rowid = ctable.qrowid AND
 blobintersectionboolean (db.data, q.data))

```
- b)
- ```

SELECT db.id, blobintersectionranked(pair(db.data, q.data))
FROM GrayContainers db, :GrayQueryContainers q
WHERE intersect (hull(db.data), hull(q.data))
GROUP BY db.id

```

Figure 74: SQL intersection-statements based on gray containers.

a) *intersect_{boolean}*-predicate, b) *intersect_{ranked}*-predicate

where the hulls of both containers intersect. In our implementation, we realized this NF2-operator *table* by a user-defined aggregate function as provided in the SQL:1999 standard. As we want to find out which database objects are intersected by a specific query object, we have to test the candidate pairs for intersection. This test is carried out by a stored procedure *blobintersection_{boolean}*. If we find one intersecting gray database and query container pair, we can stop investigating other gray container pairs belonging to the same database object, and we issue the object's *id*. This *skipping principle* is realized by means of the *exists*-clause within the SQL-statement. This approach actually creates a new spatial access method which can easily be integrated into common extensible indexing frameworks (cf. Chapter 3).

In the case of the *intersect_{ranked}* predicate (cf. Figure 74b), the intersection volume has to be determined for each candidate pair. No BLOB tests can be skipped. The results are summed up in the user-defined aggregate function *blobintersection_{ranked}*.

In both *blobintersection* routines, we first decompress the data and then test the two byte sequences for intersection. As already mentioned in Section 6.3.3, it is important that the compressed BLOB size is small in order to reduce the I/O cost. Obviously, the small I/O cost should not be at the expense of the CPU cost. Therefore, it is

important that we have a fast decompressing algorithm in order to evaluate the BLOBs quickly.

6.4.2 Optimizations

For the *intersect_{boolean}* predicate, it suffices to find a single intersecting query and database container pair in order to issue the database *id*. Obviously, it is desirable to detect such intersecting pairs as early as possible in order to avoid unnecessary *blobintersection* tests. In this section, we present two optimizations striking for this goal. First, we introduce a fast second filter step which tries to determine intersecting pairs without examining the BLOBs. This test is entirely based on aggregated information of the gray containers. Secondly, we introduce a probability model which leads to an ordering for the candidate pairs such that the most promising *blobintersection* tests are carried out first. In order to put these optimizations into practice, we pass $D(C_{gray})$ and $H(C_{gray})$ as additional parameters to the user-defined aggregate function *table*. Thus, the following two optimizations can easily be integrated into this user-defined aggregate function. If the fast second filter step determines an intersecting container pair, all other candidate pairs are deleted so that the resulting table of candidate pairs, called *ctable*, consists only of one intersecting pair. Nevertheless, there might be database objects where this second filter step does not determine an intersection for any of the corresponding candidate pairs. In this case, we sort the candidate pairs at the end of our user-defined aggregation function *table* such that the most promising blobintersection tests are carried out first.

Optimizations for one-dimensional gray intervals. Let us first mention that a gray interval with maximum density is also called a black interval. In this paragraph, we will discuss how gray and black intervals have to look like so that we can decide whether two interlacing intervals intersect each other or not without accessing their BLOBs. These optimizations are only suitable for the *intersect_{boolean}*-predicate.

Two Black Intervals. If two black intervals interlace, they necessarily intersect as well.

Black and Gray Intervals. In this case, the situation is a little bit more complicated. In almost all cases where a black interval $I_{black} = (l_{black}, u_{black})$ interlaces a gray interval I_{gray} with $H(I_{gray}) = (l_{gray}, u_{gray})$, there is an intersection of I_{black} and I_{gray} as well. If any of the two conditions depicted in Table 3 holds, then I_{black} and I_{gray} intersect.

Condition	Explanation	Figure
$L(I_{black})$ $>$ $G(I_{gray})$	If the black interval is longer than the maximum gap between two black intervals of the detailed black interval sequence of I_{gray} than the two intervals intersect.	
$u_{interlace} = u_{gray}$ or $l_{interlace} = l_{gray}$	If one of the two conditions presented in the box on the left holds, then the black and gray interval intersect. This is due to the fact that the gray intervals end and start with black intervals.	

Table 3: Intersection between an interlacing black and gray interval.

Two Gray Intervals. Obviously, two gray intervals I_{gray} and I'_{gray} , with $H(I_{gray}) = (l_{gray}, u_{gray})$ and $H(I'_{gray}) = (l'_{gray}, u'_{gray})$, which interlace do not have to intersect. Fortunately, there are two cases where we can assert that they intersect without examining the detailed black interval sequences. The two cases are illustrated in Table 4.

Condition	Explanation	Figure
$u_{gray} = u'_{gray}$ or $l_{gray} = l'_{gray}$ or $l_{gray} = u'_{gray}$	If one of the three conditions depicted in the box on the left holds, then the two gray intervals intersect (gray intervals start and end with black intervals). This test is similar to the <i>polygon boundary test</i> in [HJR 97].	
$N_w(I_{gray}) +$ $N_w(I'_{gray})$ $<$ $L_{interlace}$	If the sum of the number of the “white cells” of two gray interlacing intervals is smaller than the length of the interlacing area, then the two intervals necessarily intersect. This is the generalization of the first case of Table 3. This test is similar to the <i>false area test</i> in [BKSS 94].	

Table 4: Intersection between two interlacing gray intervals.

No intersection. There are only two situations, depicted in Table 5, where we can determine that two interlacing intervals do *not* intersect. In this case, the interlacing

interval pair is not added to *ctable*, i.e. we do not carry out the blobintersection test for this interval pair.

Condition	Explanation	Figure
$N_b(I_{gray}) = 2$ and $l_{gray} < l'_{gray}$ and $u_{gray} > u'_{gray}$	If I_{gray} consists only of two black cells and I'_{gray} is totally “included” in I_{gray} , then we know that the two intervals cannot intersect each other, although they interlace.	
$N_b(I_{gray}) = 2$ $N_b(I'_{gray}) = 2$ and $l_{gray} \neq l'_{gray} \neq$ $u_{gray} \neq u'_{gray}$	If both gray intervals consist only of two black cells and, furthermore, have distinct interval bounds, then the two intervals certainly do not intersect.	

Table 5: No intersection between two interlacing gray intervals.

False Area Test. In this paragraph we generalize the *false area test* [BKSS 94] to multi-dimensional intervals, i.e. boxes and tiles. If there exist two containers which interlace, and both containers have a high density, i.e. the number of white cells is rather small, then we know that the two objects intersect without accessing the data stored in the BLOB.

Lemma 3. Let $O = (id, \langle C_1, \dots, C_m \rangle)$ and $O' = (id', \langle C'_1, \dots, C'_{m'} \rangle)$ be two objects with their hulls $H(C_{gray}) = \bigotimes_{k=1}^d [h_l^k, h_u^k]$ and $H(C'_{gray}) = \bigotimes_{k=1}^d [h'_l{}^k, h'_u{}^k]$. Then the following statement holds:

$$\exists i \in \{1..m\} \exists j \in \{1..m'\} : N_w(C_i) + N_w(C'_j) < \prod_{k=1}^d (\min(h_u^k, h'_u{}^k) - \max(h_l^k, h'_l{}^k) + 1) \\ \Rightarrow \text{intersect}(O, O') = \text{true}$$

Proof. Let $V = \prod_{k=1}^d (\min(h_u^k, h'_u{}^k) - \max(h_l^k, h'_l{}^k) + 1)$ be the intersection volume between two gray containers C_i and C'_j . Then $N_b(C_i) \geq V - N_w(C_i)$ and $N_b(C'_j) \geq V - N_w(C'_j)$ holds. As $N_b(C_i) + N_b(C'_j) \geq V - N_w(C_i) + V - N_w(C'_j) > V$ holds, there must be at least one object voxel v belonging to O and O' , i.e. O and O' intersect. ■

Ranking. As shown above, we can pinpoint, based on relatively little information, whether two container pairs intersect or not. Nevertheless, there might be cases where we cannot do this for any of the database and query candidate pair. But if the hulls of two gray containers intersect, it is still helpful if we can predict how likely an object intersection according to Definition 18 might be in order to rank the pairs properly in the set of all candidate pairs belonging to the same database object.

Lemma 4. Let C_{gray} and C'_{gray} be two gray intervals with densities $d = D(C_{gray})$, $d' = D(C'_{gray})$ and hulls $H(C_{gray}) = \bigcup_{k=1}^d [h_l^k, h_u^k]$ and $H(C'_{gray}) = \bigcup_{k=1}^{d'} [h_l'^k, h_u'^k]$. Furthermore, let $V = \prod_{k=1}^d (\min(h_u^k, h_u'^k) - \max(h_l^k, h_l'^k) + 1) > 0$ denote the intersection volume. Then the probability for an object intersection is:

$$P = 1 - \frac{\binom{V - d \times V}{d' \times V}}{\binom{V}{d' \times V}}$$

Proof. As we assume that the black cells of both gray containers are equally distributed, we can conclude that $N = d \times V$ black cells of C_{gray} are included in the testing area and likewise $N' = d' \times V$ black cells from C'_{gray} . The number of all different possibilities how N' black cells of C'_{gray} can be placed over the V cells of the testing area is $\binom{V}{N'} = \binom{V}{d' \times V}$. Assuming that all N black cells of C_{gray} are already distributed over V , then the number of different possibilities how N' black cells of C'_{gray} can be placed over the remaining $V - N$ white cells such that no intersection occurs is $\binom{V - N}{N'} = \binom{V - d \times V}{d' \times V}$.

Thus, the probability for a *non-intersection* is equal to $\frac{\binom{V - d \times V}{d' \times V}}{\binom{V}{d' \times V}}$. ■

6.5 Spatial Join

One of the most common query types in spatial database management systems is the *spatial intersection join* [GG 98]. This join retrieves all pairs of intersecting objects. 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 more complex 3D high-resolution objects is required. One query example is to find all temperature sensitive parts touching hot parts of the engine or the exhaust pipe. The high complexity of the objects and the fact that these objects are highly clustered yield a great challenge for efficient join processing.

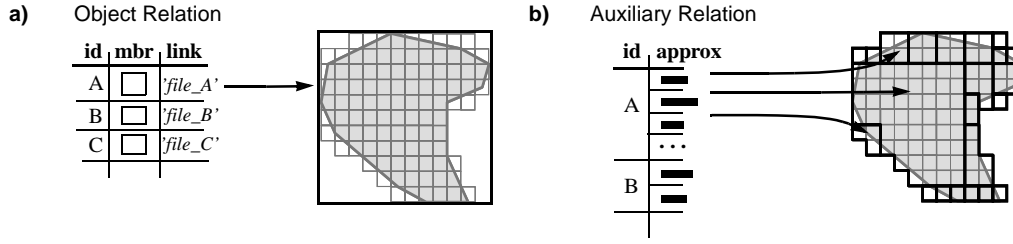


Figure 75: Spatial join procedure.

a) Object relation, **b)** Auxiliary relation with decomposed object approximations

In this section, we demonstrate the benefits of our cost-based decompositioning approach of high-resolution spatial objects by integrating it into a simple nested-loop join procedure and a more efficient sort-merge join variant. Both methods do not assume the presence of pre-existing spatial indices on the relations.

We start with two relations R and S , both containing a set of tuples $(id, mbr, link)$, where id denotes a unique object identifier, mbr denotes the minimal bounding rectangle conservatively approximating the respective object and $link$ refers to an external file containing the complete voxel set of the object (cf. Figure 75a). In this section, we assume that the voxel representation of the objects is accurate enough to determine intersecting objects without any further refinement step. In order to carry out the intersection tests efficiently, we decompose the high-resolution voxelized objects. We store the generated approximations in an auxiliary temporary relation (cf. Figure 75b), which allows us to reload certain approximations on demand keeping the main-memory footprint small.

The remainder of the section is organized as follows: In Section 6.5.1, we shortly present the related work in the area of spatial joins discussing three different classes dependent on the availability of index structures. In Section 6.5.2 and Section 6.5.3, we introduce a cost-based grouping approach into gray intervals, quite similar to the one presented in Section 6.3. The main difference is that we do not assume a potential query distribution, but use the statistics of the join partner relation as input parameter for the grouping algorithm. In Section 6.5.4, we introduce two different join algorithms based on our cost-based decompositioning. The experimental evaluation is deferred to Section 6.6.5, where we will present the benefits of our approach.

6.5.1 Related Work

In this section, we will shortly discuss different aspects of efficient spatial join processing of complex spatial objects. Numerous spatial join algorithms have been proposed over the last decade. Most of them rely on the paradigm of multi-step query processing [BKSS 94]. 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 divided into three classes:

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

The common solutions for the spatial join methods of Class 1 are the algorithms based on matching two R-trees as presented in [BKS 93b]. 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. In [LR 94] seeded trees were introduced in order to process spatial joins efficiently when only one R-tree is available. The authors propose to create a second R-tree using the available tree as a skeleton and apply thereafter a Class 1 algorithm. For spatial join algorithms of Class 3, initially no indices are available which could be used to improve the query performance. Several techniques have been proposed which partition the tuples into buckets and then use hash based techniques, e.g. the *spatial-hash join* [LR 96] or the *partition based spatial merge join* [PD 96]. The *scalable sweeping-based spatial join* [APR+ 98] is w.r.t. worst-case efficiency the most promising algorithm for processing spatial joins. Let us note that we use a simplified version of this algorithm as starting point for our sort-merge join variant presented in Section 6.5.4. Furthermore, there exist different other approaches to improve Class 3 join algorithms, e.g. in [DS 00] the problem of redundancy and duplicate detection was investigated and in [DSTW 02] a generic technique called *progressive merge join* (PMJ) was introduced that eliminates the blocking behavior of sorted-based join algorithms.

Most of the approaches presented in the literature focus on the efficient computation of the filter step where MBRs are used for approximating spatial objects. Our approach deals with very complex 3D objects, where the MBRs form rather poor approximations, leading to low filter selectivities. In the following sections, we will present a cost-based decompositioning algorithm for spatial join processing, which aims at finding an optimal trade-off between low redundancy and high approximation quality.

6.5.2 Cost Model

For our decompositioning algorithm we take the estimated join cost between a gray interval I_{gray} and a join-partner relation T into account. Let us note that T can be either of the tables R or S (cf. Figure 75a), or any temporary table containing derived information from the original tables R and S (cf. Figure 75b). The overall join cost $cost_{join}$ for a gray interval I_{gray} and a join-partner relation T are composed of two parts, the filter cost $cost_{filter}$ and the refinement cost $cost_{refine}$:

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

Filter Cost. The $cost_{filter}(I_{gray}, T)$ can be computed by the expected number of gray intervals $I_{gray, T}$ 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 gray interval $I_{gray, T}$ and the evaluation of the join predicate for the pair $(H(I_{gray}), H(I_{gray, T}))$:

$$cost_{filter}(I_{gray}, T) = N_{gray}(T) \cdot c_f$$

where $N_{voxel}(T)$ (number of voxels) $\geq N_{gray}(T)$ (number of gray intervals) $\geq N_{object}(T)$ (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 $B(I_{gray})$ in order to verify the intersection predicate. Consequently, our cost-based decompositioning 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 a gray interval I_{gray} can be computed as follows:

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

In Section 6.3.4, it was shown how to calculate the evaluation cost $cost_{eval}$. 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 $\sigma_{filter}(I_{gray}, T)$. In order to cope with arbitrary interval distributions, histograms can be employed to capture the data characteristics at any desired resolution (cf. Section 4.3.1). The selectivity $\sigma_{filter}(I_{gray}, T)$ related to a gray interval I_{gray} can be determined by using an appropriate interval histogram $H(T, v)$ of the join partner relation T . Based on $H(T, v)$, we compute a selectivity estimate by evaluating the intersection of I_{gray} with each bucket span $b_{i,v}$ (cf. Definition 7 and Figure 38).

$$\sigma_{filter}(I_{gray}, T) = \frac{\left[\sum_{i=1}^v \frac{overlap(H(I_{gray}), b_{i,v})}{\beta} \cdot n_i \right]}{\left[\sum_{i=1}^v n_i \right]},$$

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

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

6.5.3 Decompositioning Algorithm

For each object, there exist a lot of different possibilities to decompose the voxelized object into a gray interval sequence.

Lemma 5. Let $O_{voxel} = (id, \{v_1, \dots, v_n\})$ be a voxelized object and $\rho: IN^d \rightarrow IN$ be a space filling curve. Furthermore, let $W = \{(l, u) \in IN^2, l \leq u\}$ be the domain of intervals and let $b_1 = (l_1, u_1), \dots, b_n = (l_n, u_n) \in W$ be a sequence of intervals where $u_i + 1 < l_{i+1}$ representing the set $\{\rho(v_1), \dots, \rho(v_n)\}$. Then, there exist $O(2^n)$ different gray interval sequences $O_{gray} = (id, \langle \langle b_{i_0+1}, \dots, b_{i_1} \rangle, \langle b_{i_1+1}, \dots, b_{i_2} \rangle, \dots, \langle b_{i_{m-1}+1}, \dots, b_{i_m} \rangle \rangle)$.

Proof. The interval sequence representing O_{voxel} consists of $n-1$ “gaps”. For each of these gaps we can decide whether it is included in a gray interval, or whether it separates two gray intervals. Thus we have 2^{n-1} different possible gray interval sequences. ■

```

ALGORITHM JoinGroupCon ( $I_{gray}, H(T,v), T$ )
BEGIN
     $interval\_pair := split\_at\_maximum\_gap(I_{gray});$ 
     $left := interval\_pair.left;$ 
     $right := interval\_pair.right;$ 
     $cost_{gray} := cost_{join}(I_{gray}, T, H(T,v));$ 
     $cost_{dec} := cost_{join}(left, T, H(T,v)) + cost_{join}(right, T, H(T,v));$ 
    IF  $cost_{gray} > cost_{dec}$  THEN
        RETURN  $JoinGroupCon(left, H(T,v), T) \cup JoinGroupCon(right, H(T,v), T);$ 
    ELSE
        RETURN  $I_{gray};$ 
    END IF;
END.

```

Figure 76: Decompositioning algorithm *JoinGroupCon*.

Based on the formulas for join cost related to a gray interval I_{gray} and a join-partner relation T , we can find a cost optimum decompositioning algorithm. Unfortunately, as shown in the above lemma, there exist exponentially many decompositioning possibilities, which results in a high runtime of an optimum cost-based decompositioning algorithm. In this section, we will present a greedy algorithm with a guaranteed worst-case runtime complexity of $O(n)$ which produces decompositions helping to accelerate the query process considerably (cf. Section 6.6.5).

For fulfilling the grouping rules presented in Section 6.3, we introduce the following top-down decompositioning algorithm for gray intervals, called *JoinGroupCon* (cf. Figure 76). *JoinGroupCon* is a recursive algorithm which starts with a gray interval I_{gray} 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 I_{gray} . The expected join cost $cost_{join}(I_{gray}, T)$ 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 *JoinGroupCon* algorithm generating a small number of gray intervals. Let us note that the inequality “ $cost_{gray} > cost_{dec}$ ” in Figure 76 is independent of $N_{gray}(T)$, and thus $N_{gray}(T)$ is not required during the decompositioning algorithm.

distribution of the relation R into account where each object is approximated by a gray interval $I_{gray} = (z\text{-val}(mbr.lower), z\text{-val}(mbr.upper))$ (cf. Figure 69). The gray object interval sequence $O_{gray,S'}$ of each voxelized object $O_{voxel,S}$ is materialized in a temporary relation S' following the NF² schema of the relation of Figure 75b.

In the following description of the nested-loop join, we assume that the objects $O_{voxel,R}$ of relation R are accessed only once. Thus, there is no need to materialize the corresponding gray intervals in the database in contrast to the objects of relation S . Assuming that one object completely fits into memory, its gray intervals can be built on-the-fly during the join phase.

Join Phase. The join phase is performed in a *nested-loop* fashion. For each object, we execute the function $JoinGroupCon(I_{gray,R}, H(S',v), S')$ in the outer loop in order to build the gray object interval sequence $O_{gray,R'}$. This time, we apply the data distribution statistics of relation S' where each object $O_{gray,S'}$ might be represented by several gray intervals. In the inner loop, we test $O_{gray,S'}$ for intersection with $O_{gray,R'}$ calling the boolean function $intersect()$.

The function $intersect(O_{gray,R'}, O_{gray,S'})$ checks whether two gray object interval sequences $O_{gray,R'}$ and $O_{gray,S'}$ intersect. They intersect, iff there is at least one gray interval pair $(O_{gray,R'} \cdot I_{gray}, O_{gray,S'} \cdot I_{gray})$ which intersects. If we assume that all gray intervals of $O_{gray,R'}$ and $O_{gray,S'}$ fit in main memory and are sorted in ascending order by their starting point, we can efficiently perform the intersection tests by processing both interval sequences in parallel. As soon as an intersection is detected, the remaining tests for this object pair can be skipped and the value “true” is issued. The actual intersection test of a gray-interval pair is performed in two steps: in the first step (*filter step*) the intervals are 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. In order to execute the expensive exact intersection test, we have to load the byte sequence $B(O_{gray,S'} \cdot I_{gray})$ from disk, decompress it and test it for an intersection with $B(O_{gray,R'} \cdot I_{gray})$ which is still in main memory. Furthermore, we suggest to use the optimizations presented in Section 6.4.2 in order to avoid some of the costly refinement steps.

Sort-Merge Join. In this section, we introduce a variant of the sort-merge join. It is based on the worst-case optimal interval join algorithm described in [APR+ 98]. In the following we consider R and S as input relations. Initially each object O_{voxel} of

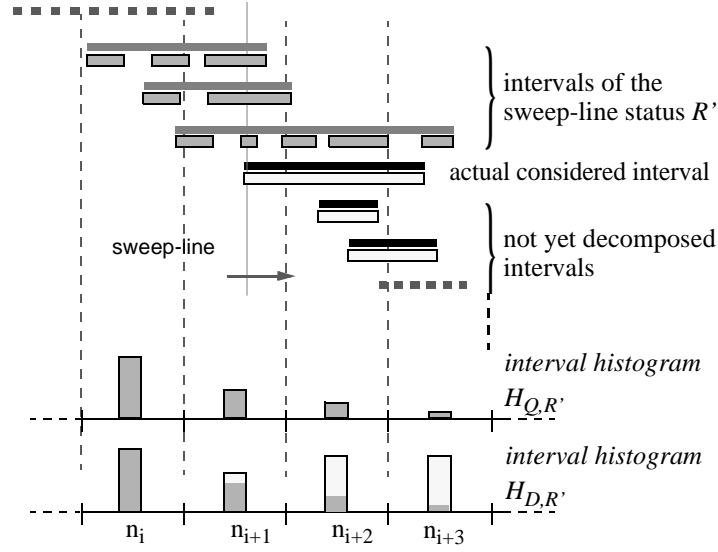


Figure 78: Intervals stemming from R and the corresponding histograms.

both relations is approximated by one gray interval $I_{gray} := (\min(z\text{-val}(O_{voxel})), \max(z\text{-val}(O_{voxel})))$. The join algorithm is performed in plane-sweep fashion. 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. Both relations R' and S' follow the schema of S' of the nested-loop join.

In analogy to the nested-loop join, we apply the algorithm *JoinGroupCon* (cf. Figure 76) 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 interval histograms for each data set. The interval histograms $H_{Q,R'}$ and $H_{Q,S'}$ represent the data distribution within the actual sweep-line status. The other interval histograms $H_{D,R'}$ and $H_{D,S'}$ 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 v , so that their bucket borders are congruent. An example is shown in Figure 78.

During the join processing, we try to estimate the filter selectivity for each actual considered gray interval 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 contri-



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. The complete join algorithm described in the following is depicted in Figure 79:

Preprocessing Phase. Initially, we gather the statistics about the data distribution of R and S and store them in the two interval histograms $H_{D,R}$ and $H_{D,S}$. Thereby, $H_{D,R}$ is generated from the data of R and $H_{D,S}$ from the data of S . Next, we order all bounding boxes of both relations R and S according to their starting point, i.e. $z\text{-val}(mbr.lower)$. We suggest to use the Z-order as space filling curve so that the lower left vertex of the bounding box coincides with the starting point of the gray interval I_{gray} covering the complete object and the upper right vertex coincides with

the end point of I_{gray} . Let us note that sorting the objects is rather cheap if we assume that we have comparatively few objects but rather complex ones.

Join Phase. We apply a plane sweep algorithm to walk through the sorted lists of gray intervals of both relations R and S . The event points of this algorithm are the starting points of the gray intervals of both relations. Each encountered interval $I_{gray} = (l, u)$ from relation S (R) is now processed according to the following three steps:

- **Step 1:** I_{gray} is decomposed based on the data distribution of the actual sweep-line status by applying $H_{Q,R}$ ($H_{Q,S'}$) and stored in a temporary list Q . In the same step we decompose I_{gray} applying the statistics $H_{D,R}$ ($H_{D,S'}$) and buffer the result in another temporary list D .
- **Step 2:** The resulting decomposed intervals of Q are used as query objects for the relation R' (S'). We report all objects having a gray interval I'_{gray} stored in R' (S') which intersects at least one of the decompositions of I_{gray} . These intersection queries can be efficiently carried out as outlined in Section 6.4.
- **Step 3:** The decomposed intervals of D are stored in the temporary relation S' (R'). In order to keep the relations R' and S' small, we delete all gray intervals $I'_{gray} = (l', u')$ from R' and S' where u' is smaller than l , i.e. all intervals which are certainly not accessed anymore. Finally, we have to update the interval histograms $H_{D,S'}$ and $H_{Q,S'}$ ($H_{D,R'}$ and $H_{Q,R'}$).

Let us note that our algorithm *JoinGroupCon* (cf. Figure 76) 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 (cf. Section 6.6.5).

6.6 Experimental Evaluation

In this section, we evaluate the performance of our approach for accelerating relational spatial index structures and spatial join processing, with a special emphasis on the various data compression techniques introduced in Section 6.3. We evaluate different grouping algorithms *GRP* in combination with various data compression techniques *DC*. We used the following data compressors *DC*:

NOOPT	The BLOB is unpacked
BZIP2	The BLOB is packed according to the <i>BZIP2</i> approach
ZLIB	The BLOB is packed according to the <i>ZLIB</i> approach
GEOM	The BLOB is packed according to the approach for simply structured objects
OPTRLE	The BLOB is packed according to the approach in [KPPS 03a]
QSDC	The BLOB is packed according to the <i>QSDC</i> approach

Furthermore, we grouped voxels into gray containers depending on two grouping algorithms *GRP*:

MaxGap. This grouping algorithm tries to minimize the number of gray containers while not allowing that a maximum gap $G(C_{gray})$ of any gray container C_{gray} 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 6.3, namely a small number of gray containers and a small number of white cells included in each of these containers.

GroupCon. We grouped the containers according to our cost-based grouping algorithm *GroupCon* (cf. Section 6.3 and Section 6.5), where the resolution of the used histograms was set to 100 buckets. Furthermore, we used the query distribution function from Figure 71b with $k^* = 1/100,000$. Note, that the grouping based on *MaxGap(DC)* does not depend on *DC*, whereas *GroupCon(DC)* takes the actual data compressor *DC* into account for performing the grouping.

In order to support the first filter step of *GRP(DC)*, we can take any arbitrary access method for spatial data, e.g. the RI-tree, the RR-tree or the RQ-tree. We have implemented the RI-tree and the RQ-tree on top of the Oracle9i Server using PL/SQL for most of the computational main memory based programming. Furthermore, we applied *GRP(DC)* to the RR-tree provided by Oracle [RRSB 99]. The evaluation of the blobintersection routines was delegated to a DLL written in C. All experiments were performed on a Pentium 4/2600 machine with IDE hard drives. The database

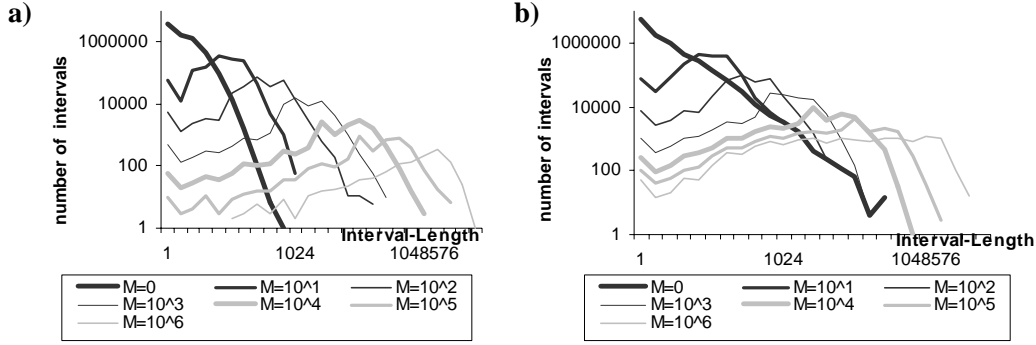


Figure 80: Interval length depending on the *MAXGAP* parameter.

a) CAR, b) PLANE

block cache was set to 500 disk blocks with a block size of 8 KB and was used exclusively by one active session.

Test Data Sets. All tests were carried out on our two test data sets CAR and PLANE (cf. Section 5.4). The gap distributions of these two data sets was already discussed in Section 5.4. Figure 80 depicts the interval distribution. It can be seen that the bucket which includes most intervals is regularly increasing with increasing *MAXGAP*. The gap histograms depending on the *MAXGAP* parameters are obvious because all gaps smaller than *MAXGAP* are used to form the gray intervals. On the other hand, the gaps larger than *MAXGAP* are unused.

6.6.1 Storage Requirements

First we look at the storage requirements of the RI-tree on the *PLANE* data set. In Figure 81a, the storage requirements for the index, i.e. the two B^+ -trees underlying the RI-tree, as well as for the complete *GrayContainers* table are depicted for the *MaxGap(QSDC)* approach. In the case of small *MAXGAP* parameters, the number of

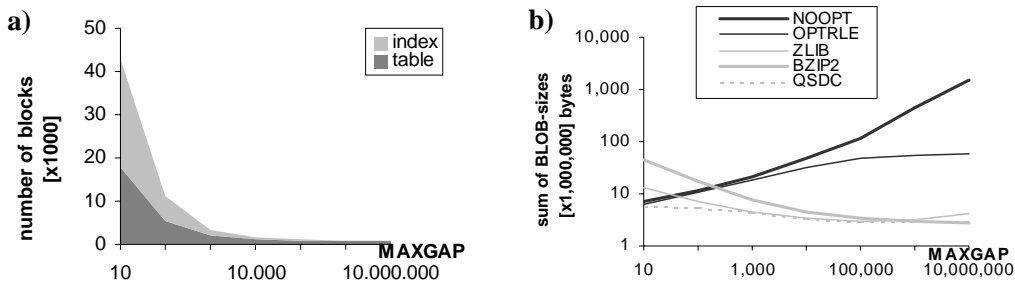


Figure 81: Storage requirements for the *RI-tree* (*PLANE*).

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

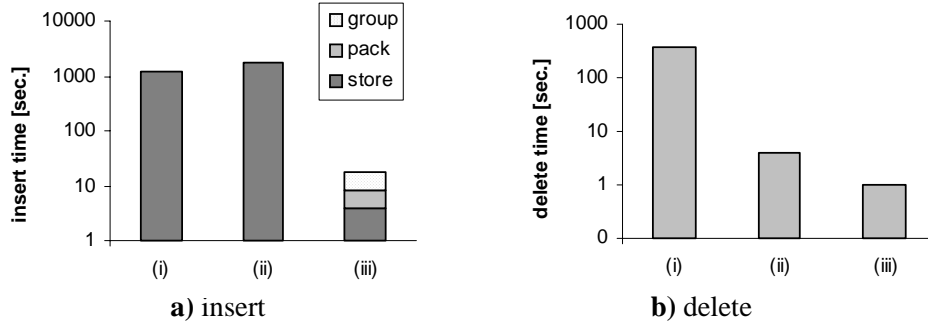


Figure 82: Update operations for the *RI-tree* (*CAR*).

- (i) Numerous black intervals, (ii) One gray interval with unpacked BLOB,
 (iii) Gray intervals grouped by *GroupCon(QSDC)*

disk blocks used by the index dominates the number of disk blocks for the *GrayContainers* table. With increasing *MAXGAP* parameters, the number of disk blocks used by the index dramatically decreases hand in hand with the number of gray container objects, and at high parameter values they yield no significant contributions any more to the overall sum of used disk blocks.

Figure 81b shows the different storage requirements for the BLOB with respect to the different data compression techniques. Due to an enormous overhead, the *ZLIB* and *BZIP2* approaches occupy a lot of secondary storage space for small *MAXGAP* values. On the other hand, for high *MAXGAP* values they yield very high compression rates. For the *PLANE* data set 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 [KPPS 03a]. The *QSDC* approach yields good results over the full range of the *MAXGAP* parameter. For high *MAXGAP* values, the number of disk blocks used for the BLOBs corresponds to the number of disk blocks used overall. For these high *MAXGAP* parameters, the *MaxGap(QSDC)*, *MaxGap(ZLIB)* and *MaxGap(BZIP2)* approach lead to a much better storage utilization than the *MaxGap(NOOPT)* and the *MaxGap(OPTRLE)* approach.

6.6.2 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. Fig-

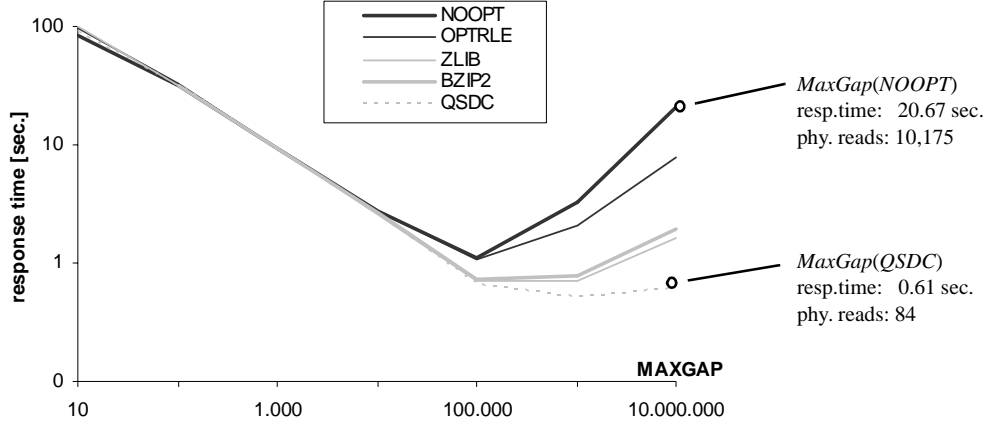


Figure 83: Boolean intersection queries for MaxGap(DC) (RI-tree (PLANE)).

ure 82a shows that inserting all objects into the database takes very long if we store the numerous black 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 *Group-Con(QSDC)* approach (*iii*) accelerates the insert operations by almost two orders of magnitude. The time spent for grouping and packing pays off, if we take into consideration that we save a lot of time for storing grouped and packed objects in the database. Obviously, the delete operations are also carried out much faster for our *Group-Con(QSDC)* approach as we have to delete much less disk blocks (cf. Figure 82b).

6.6.3 Collision Queries based on the MaxGap-Grouping

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

Packers. In Figure 83 it is shown in which way the overall response time for *boolean intersection* queries based on the RI-tree depends on the *MAXGAP* parameter. If we use small *MAXGAP* parameters, we need a lot of time for the first filter step whereas the blobintersection test is relatively cheap. Therefore, the different *MaxGap(DC)* approaches do not differ very much for small *MAXGAP* values. For high *MAXGAP* values we can see that the *MaxGap(QSDC)* approach performs best with respect to the overall runtime. The *MaxGap(QSDC)* approach is rather insensitive

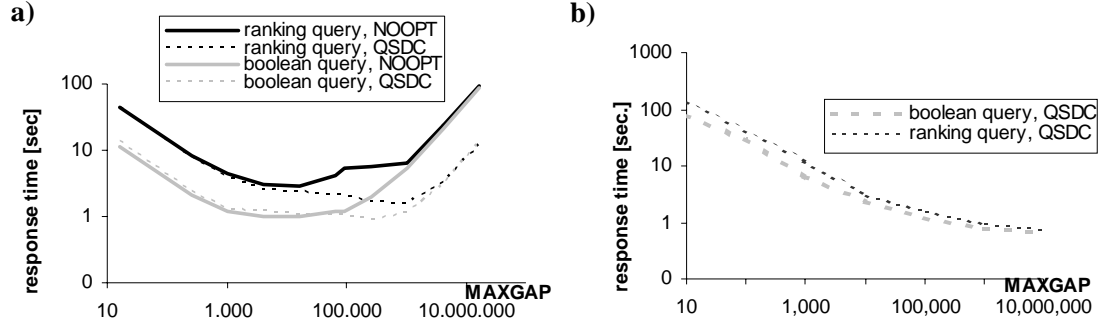


Figure 84: Intersection queries (CAR).

a) RQ-tree, b) RR-tree

against too large *MAXGAP* parameters. Even for values where the first filter step is almost irrelevant, e.g. $MAXGAP = 10^7$, 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.

Figure 84 depicts the results of the *MaxGap(QSDC)* approach for the RQ-tree and the RR-tree. Basically, we made the same observations as in the case of the RI-tree. In the case of the RR-tree we only implemented the packed version, because the byte sequence $B(C_{gray})$ might contain unreasonably long gaps, which are not taken into consideration during the grouping process.

In Figure 85 it is shown in what way the different data space resolutions influence the query response time. Generally, the higher the resolution, the slower is the query

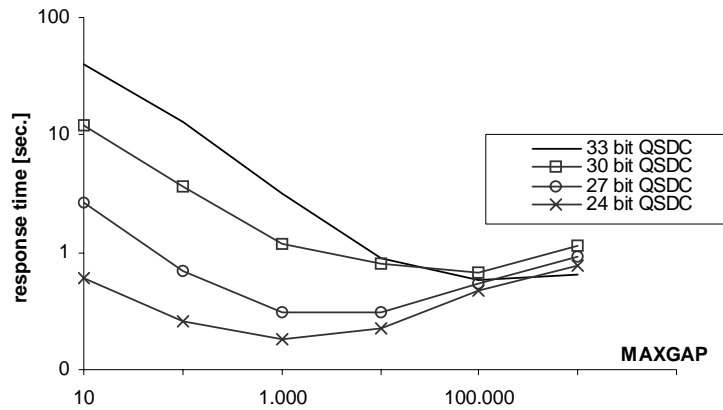


Figure 85: Boolean intersection queries for *MaxGap(QSDC)*.
(RI-tree using different resolutions (CAR))

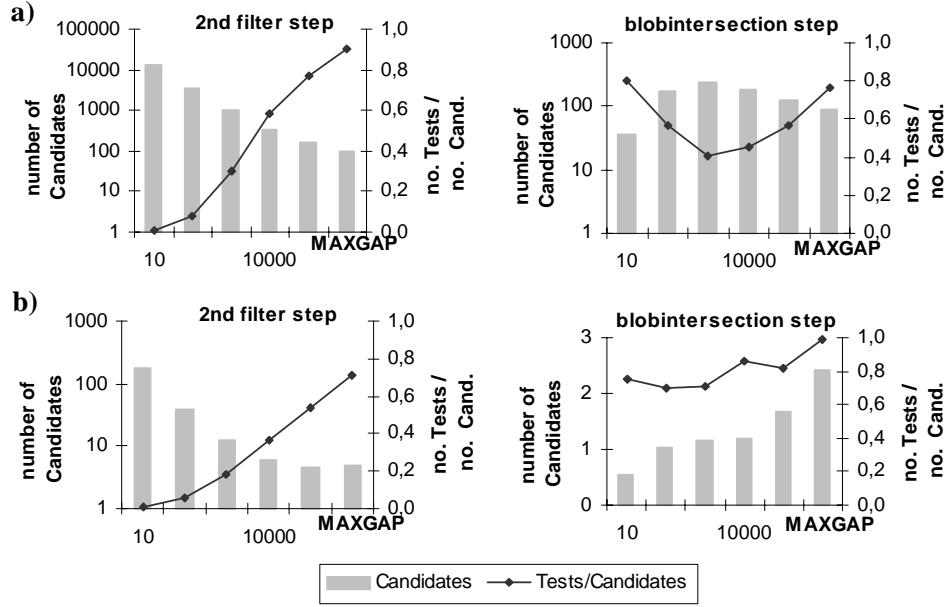


Figure 86: Tested candidate pairs of gray query and database intervals.
(Boolean intersection queries on the RI-tree)
a) CAR, b) PLANE

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.

Number of Tested Candidates. Figure 86 illustrates the number of interval candidate pairs and the number of the corresponding tests which are carried out for boolean intersection queries on the RI-tree. In the *second filter step*, the number of the candidate pairs rapidly decreases with increasing *MAXGAP* value although the number of candidate object IDs increases (cf. Figure 87). At low *MAXGAP* values, we have to test only a fractional amount of candidate pairs as the *fast second filter step* works very successfully with this parametrization (cf. Figure 86). Consequently, there is only a relative small number of candidate pairs left for the blobintersection tests.

In the *blobintersection step*, the number of both candidate pairs and corresponding tests do not vary as much as in the second step. Nevertheless, Figure 86 shows that this step benefits as well from the *skipping principle* introduced in Section 6.4.

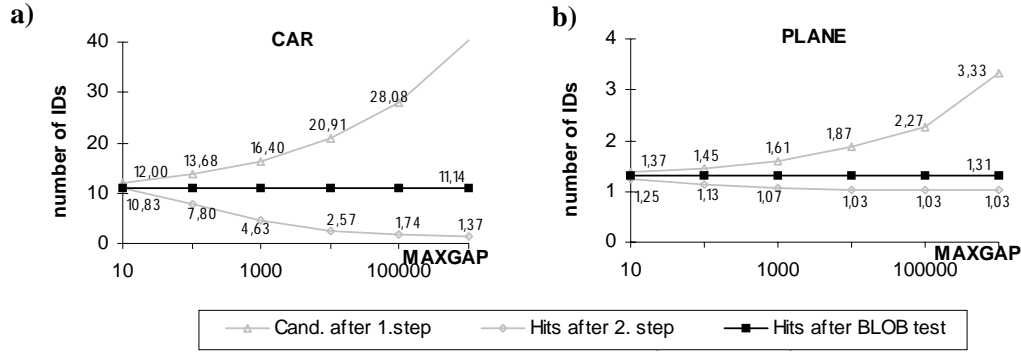


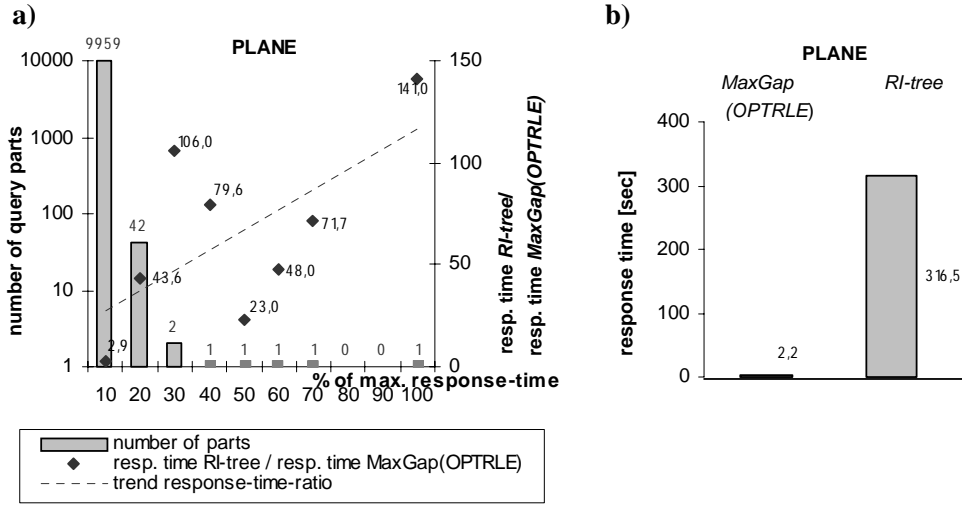
Figure 87: Candidate and result sets.
(Boolean intersection queries on the RI-tree)
a) CAR, b) PLANE

In Figure 87, it is illustrated that at small *MAXGAP* values the number of the different object IDs resulting from the first filter step is only marginally higher than the number of different IDs in the final result set. Likewise, the number of detected hits in the second filter step is only marginally smaller. With increasing *MAXGAP* values the two curves diverge.

Miscellaneous. The size of the parts in the *PLANE* data set varies considerably. We have a lot of small parts and only a few very large ones. In the case of the *CAR* data, this peculiarity is far less distinctive. As large query parts produce a large number of query intervals, it is obvious that the size of a part correlates with the response time. In Figure 88a, it is shown that for most parts from the *PLANE* data set the *MaxGap(OPTRLE)* method (*MAXGAP* = 10,000, *resolution* 33 bit) outperforms the RI-tree ‘only’ by a factor of 2.9 whereas there are some parts for which this factor is higher than 100. In Figure 88b, it is illustrated that we have to wait for more than five minutes for some collision queries when using the RI-tree. On the other hand, using the *MaxGap(OPTRLE)* method yields almost interactive response times for all collision queries.

6.6.4 Collision Queries based on the GroupCon-Grouping

Figures 83 and 84a show that for packed data, the optimum *MAXGAP* value is higher than for unpacked data. Furthermore, Figure 85 shows that for increasing resolutions the optimum *MAXGAP* also increases. We will now experimentally show that the *GroupCon* algorithm produces object decompositions which yield almost

**Figure 88:** Response time.

a) $|\text{resp. time RI-tree}| / |\text{resp. time MaxGap(OPTRLE)}|$ method, **b)** Maximum response times

optimum query response times for varying index structures, compression techniques and data space resolutions.

Table 6 depicts the overall query response time for boolean and ranking intersection queries for the RI-tree based on the *GroupCon* algorithm.

	NOOPT	BZIP2	QSDC	RI-tree [KPS 00][KPS 01]
number of containers	24,453	16,063	15,468	9,289,569
overall runtime for boolean queries [sec.]	1.35	0.71	0.55	135.01
overall runtime for ranking queries [sec.]	2.42	1.23	0.92	∞ (not applicable)

Table 6: *GroupCon* (DC) evaluated for *boolean* and *ranking* intersection 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 83. Furthermore, we notice that the *GroupCon*(QSDC) approach outperforms the RI-tree [KPS 01] by a factor of 180 for boolean intersection queries on the *PLANE* data set. For ranking intersection queries the RI-tree [KPS 00] is not applicable due to the enormous amount of generated join partners. On the other hand, the *GroupCon*(QSDC) approach yields interactive response times even for such queries. The *GroupCon* algorithm adapts to the optimum *MAXGAP* parameter for varying compression techniques, by allowing

greater gaps for packed data, i.e the number of generated container objects is smaller in the case of packed data.

In Table 7 it is shown that the query response times resulting from the *GroupCon* algorithm for different index structures, are almost identical to the ones resulting from a grouping based on an optimum *MAXGAP* parameter (cf. Figure 84a (*RQ-tree*) 84b (*RR-tree*) and 85 (*RI-tree* with a resolution = 33)).

	RR-tree	RQ-tree	RI-tree
overall runtime for <i>boolean</i> queries [s]	1.21	0.91	0.61
overall runtime for <i>ranking</i> queries [s]	2.34	2.12	1.23

Table 7: *GroupCon* (*QSDC*) evaluated for boolean and ranking intersection queries (*CAR*).

In Table 8 it is shown that the query response times resulting from the *GroupCon* algorithm for varying resolutions, are almost identical to the ones resulting from a grouping based on an optimum *MAXGAP* parameter (cf. Figure 85).

	33 bit	30 bit	27 bit	24 bit
overall runtime[s]	0.64	0.7	0.29	0.22

Table 8: *GroupCon* (*QSDC*) evaluated for boolean intersection queries for the *RI-tree* with different resolutions (*CAR*).

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

6.6.5 Spatial Join Processing

In this section, we want to show how our decompositioning algorithm accelerates the presented join algorithms of Section 6.5.4. We have performed intersection joins over two relations, each containing approximately a half of the parts from the *CAR* data set, i.e. 14×10^6 voxel and approximately 200 objects. We took care that the data of both relations have similar characterizations with respect to the object size and distribution.

Nested Loop Join. In Figure 89 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 89a) and no compression (cf.

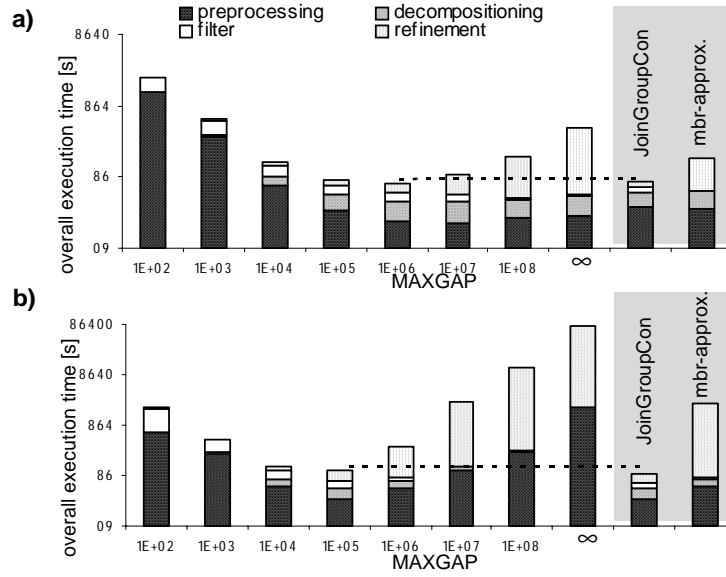


Figure 89: *GRP(DC)* evaluated for the nested loop join (*CAR* data set).

a) *QSDC* compression, **b)** *NOOPT* (no compression)

Figure 89b). The figures depict the overall contributions of the *preprocessing phase*, of the *on-the-fly decomposition* 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 influenced by the BLOB sizes, 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 is very cheap then. The *preprocessing step* itself consists of three parts, i.e. loading of the original voxel sets of relation *S*, decomposition of the voxel sets and storing the resulting gray intervals in relation *S'*. Note, that the loading part is independent of the chosen *MAXGAP* parameter and the decomposition according to a *MAXGAP* parameter is a straightforward task which does not consume much CPU time. Thus, the differences in the preprocessing step are caused by storing the gray intervals to disk. Figure 89b shows that for extremely high and small *MAXGAP* values these cost are very high. For small values this is due to the fact that we have to store an enormous amount of gray intervals each having a constant overhead. For high *MAXGAP* values, we have to store only a few very large gray intervals resulting as well in high I/O cost. Note that if we use the *QSDC* approach for high *MAXGAP* values, the I/O cost of the preprocessing step still remain low (cf. Figure 89a).

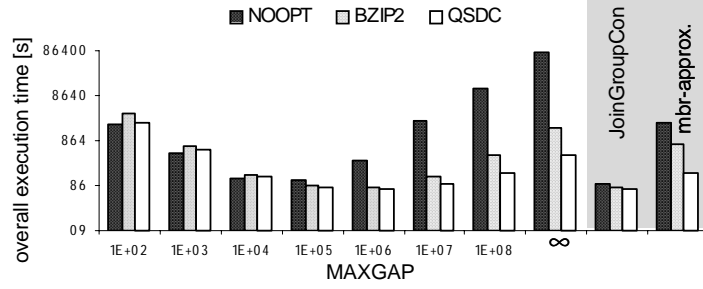


Figure 90: Overall nested-loop join performance for different packers.
(CAR data set)

Taking all steps together, Figures 89 and 90 show that 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*. Both figures show that the *JoinGroupCon* algorithm produces object decompositions which yield the optimal trade-off between the *filter* and *refinement cost* for varying compression techniques. Let us note that our cost-based decompositioning algorithm *JoinGroupCon* together with our *QSDC* approach accelerates the nested-loop join in the best possible way. Furthermore, we implemented the nested-loop join approach where the mbr approximations of the objects are used as geometric filter. Note that we also conducted a preprocessing step in order to generate a more suitable object representation. Without this additional preprocessing step, the time for the nested loop variant would be very high due to a much more expensive refinement step. Our approach outperforms the uncompressed variant of the nested loop join based on the mbr approximations by more than one order of magnitude.

Sort-merge join. In this paragraph, we demonstrate how our cost-based decompositioning algorithm based on the *QSDC* approach accelerates the sort-merge join while keeping the main memory-footprint small. Figure 91 shows 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 79) are very expensive if the available main memory is limited. If we use our *JoinGroupCon* algorithm without any compression, we need 50 MB or more to get the best possible runtime. If we use *JoinGroupCon* 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

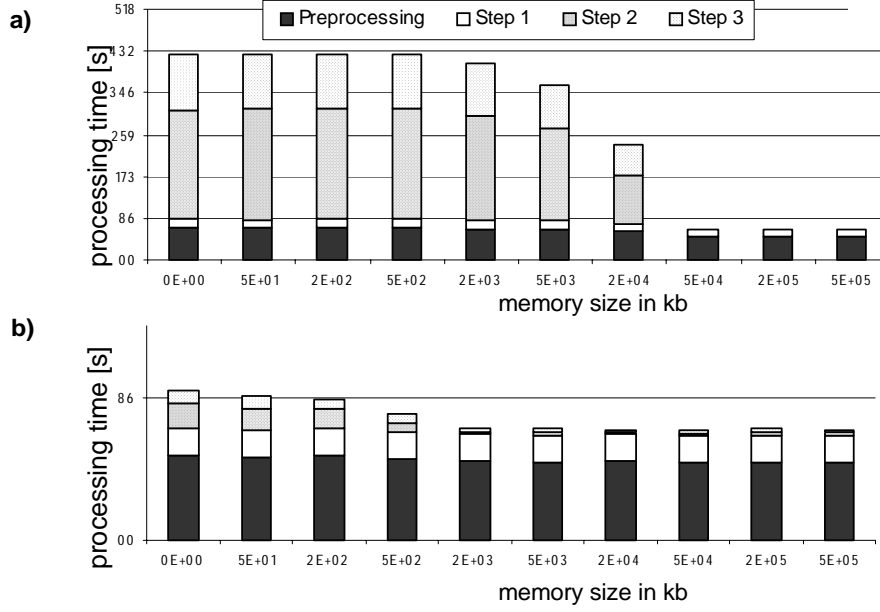


Figure 91: Sort-merge join performance.
 (CAR dataset; different cache sizes of the sweep-line status)
 a) *JoinGroupCon (NOOPT)*, b) *JoinGroupCon (QSDC)*

unpack speed. Note that already by 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 92 shows the influence of the available main memory for one-value interval approximations, i.e. $O_{gray} = (id, I_{gray})$, and gray approximations formed by our *JoinGroupCon* algorithm. The one-value interval approximations produce more false hits resulting in higher refinement cost. Note, that 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 figure demonstrates 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 *JoinGroupCon* together with our *QSDC* approach accelerates the sort-merge join in the best possible way while keeping the required main memory small. For reasonable main memory sizes we achieve an acceleration by more than two orders of magnitude for the *CAR* data set.

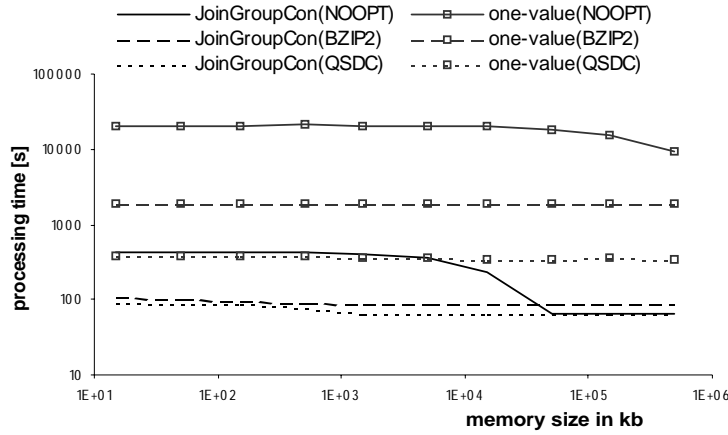


Figure 92: Overall sort-merge join performance.
(CAR data set; different cache sizes of the sweep-line status)

6.7 Conclusion

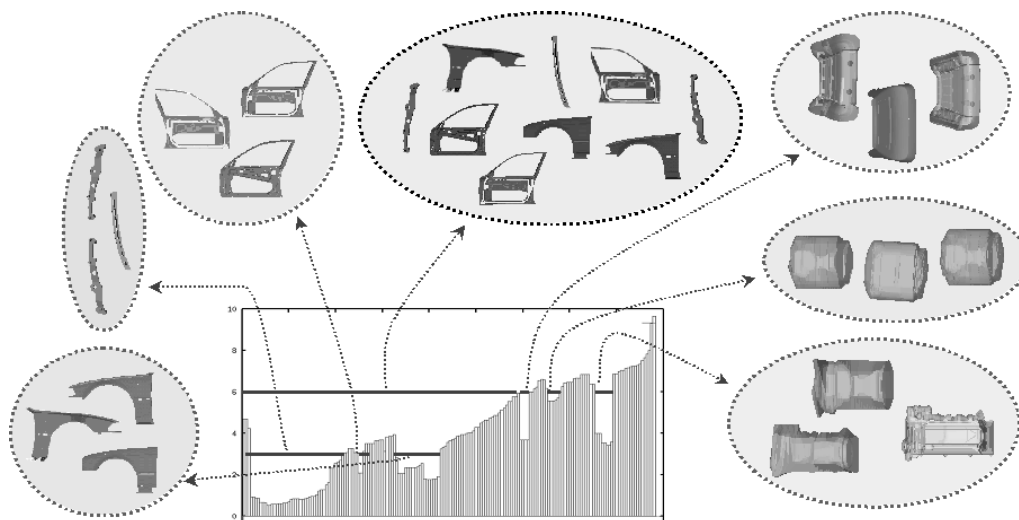
In this chapter, we introduced a new approach for accelerating spatial query processing for relational index structures. We presented gray containers as a new and general concept and showed how we can efficiently store them by means of data compression techniques within ORDBMSs. In particular, we introduced 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 introduced a cost-based decompositioning algorithm for complex spatial objects, called *GroupCon*. *GroupCon* takes decomposition cost and access probabilities of gray containers into account. This decompositioning algorithm is applicable for different spatial index structures, data space resolutions and compression algorithms. We showed in a broad experimental evaluation that the combination of *GroupCon* and *QSDC* accelerates the RI-tree, the RQ-tree and the RR-tree by up to two orders of magnitude. Furthermore, we showed that the combination of a slightly altered *GroupCon* algorithm, called *JoinGroupCon*, and *QSDC* accelerates spatial join processing of complex objects by more than one order of magnitude compared to the use of uncompressed one-value approximations. The main difference between *GroupCon* and *JoinGroupCon* is that the latter does not

assume a potential query distribution, but exploits available statistics of the join partner relation as input parameter for the grouping process.

Note that for spatial indexing a similar approach is conceivable by combining the results of this chapter with the results of the foregoing chapter. In Chapter 5, we concentrated on the acceleration of relational indexing by means of statistics, whereas in this chapter we looked at the decomposition of complex spatial objects based on an assumed query distribution. Combining these two techniques allows to accelerate relational index structures in such a way that interactive response times for digital mockup and other application ranges of virtual engineering are possible.

Part III

Database Support for Similarity Search



Chapter 7

Foundations of Similarity Search

Similarity search has gained increasing importance in many different applications, including medical imaging [KSF+ 96], molecular biology [AKKS 99], multimedia [Gud 95], and computer aided design [BKK 97a] [BKK 97b]. The search of similar database objects for a given query object is typically performed by following a feature-based approach. The basic idea is to extract important properties from the original data objects and to map these features into high-dimensional *feature vectors*, i.e. points in the feature space. Since the choice which features to extract mainly depends on the considered application, numerous *feature transformations* have been proposed. The result of such a transformation is a feature vector which is stored in a *feature database*, e.g. spatial databases storing feature transformed landuse maps, multimedia databases storing feature transformed audio sequences, and CAD databases storing feature transformed industrial parts.

This chapter is dedicated to the foundations of similarity search, with a strong emphasis on related work. It is organized as follows. In Section 7.1, we formally introduce the basic similarity query types, and discuss, in Section 7.2, how we can integrate them into an ORDBMS. In the Sections 7.3 to 7.5, we present different access methods and algorithms from the literature which are used for *efficient similarity search*. In Section 7.6, we discuss existing approaches for *effective similarity search*.

7.1 Similarity Query Types

There are some specific query types that occur in the context of similarity search in CAD databases. The most important ones are: *range queries*, *k-nearest neighbor queries*, and *incremental ranking queries*. Whereas for range queries, the number of results is typically unknown in advance, the *k*-nearest neighbor queries specify the retrieval of those *k* objects from the database that have the smallest distances to *q*. Finally, similarity ranking queries support incremental fetching of the database objects.

In this section, we provide formal definitions for these fundamental similarity query types. Let O be the domain of all objects that may occur as database objects or query objects. For every type of similarity search, a distance function $\text{simdist}: O \times O \rightarrow IR_0^+$ has to be provided that measures the (dis-)similarity of two objects o_1 and o_2 by $\text{simdist}(o_1, o_2)$. Often we abbreviate simdist by d . By $DB \subseteq O$, let us denote a database containing $N = |DB|$ objects.

7.1.1 Similarity Range Queries

Range queries are specified by a query object q and a range value ϵ by which the answer set is defined to contain all the objects o from the database that have a distance to the query object q of less than or equal to ϵ :

Definition 19 (Similarity Range Query).

For a query object $q \in O$ and a query range $\epsilon \in IR_0^+$, the similarity range query $\text{sim}_{\text{range}}: O \times IR_0^+ \rightarrow 2^{DB}$ returns the set

$$\text{sim}_{\text{range}}(q, \epsilon) = \{o \in DB \mid \text{simdist}(o, q) \leq \epsilon\}.$$

Note that for the similarity range query, the distance values of the resulting objects is bounded by the query range ϵ , but the number of answers is previously unknown. The result may be empty if no object has a similarity distance to the query object that is less or equal to the query range, and it may enclose the overall database if no object has a distance to the query object that is greater than the query range (cf. Figure 93). A user may thus be forced to iteratively start several queries before getting a feeling for an appropriate value of ϵ . For the query range $\epsilon = 0$, the similarity range query is equivalent to a *point query* (i.e. searching for identical database objects). However, the point query is a seldom used query type in the context of similarity search.

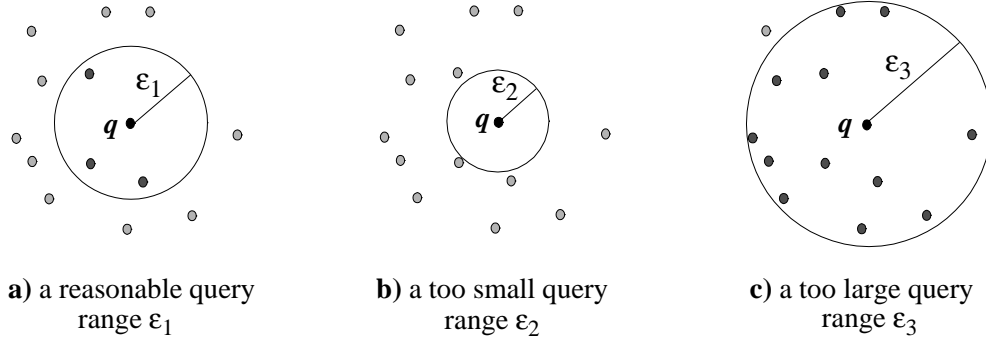


Figure 93: Similarity range query.

7.1.2 Similarity k-nn Queries

The k -nearest neighbor query overcomes the problem of the similarity range query by giving the user the possibility to specify the size k of the answer set. This query type does not require a user to provide a query range and is therefore far easier to use than the similarity range query. The k -nearest neighbor query returns the k most similar feature vectors from the database and is defined as follows:

Definition 20 (Similarity k -Nearest Neighbor Query).

For a query object $q \in O$ and a query parameter $k \in \mathbb{N}$, the k -nearest neighbor query $sim_{knn}: O \times \mathbb{N} \rightarrow 2^{DB}$ returns the set $NN_q(k) \subseteq DB$ that contains k objects from the database, and for which the following condition holds:

$$\forall o_1 \in NN_q(k), \forall o_2 \in DB \setminus NN_q(k): (simdist(o_1, q) \leq simdist(o_2, q))$$

If there exist several database objects with the same distance as the k -th object in the answer set, denoted as $simdist_{q,k}$, this k -th object is a non-deterministic selection of one of those equally distanced objects. If the query parameter k is equal to 1, we have the special case of a *nearest neighbor query*, i.e. finding the most similar object in the database. Obviously, the value of k depends on the performed task, but in general, the value for this query parameter is small ($k < 100$). Examples for k -nearest neighbor queries with several values of k are given in Figure 94. As depicted, $simdist_{q,k}$ grows monotonically for an increasing value of k .

When considering the k -nearest neighbor query as defined above, we find three aspects which may be considered as a disadvantage for CAD applications. First, al-

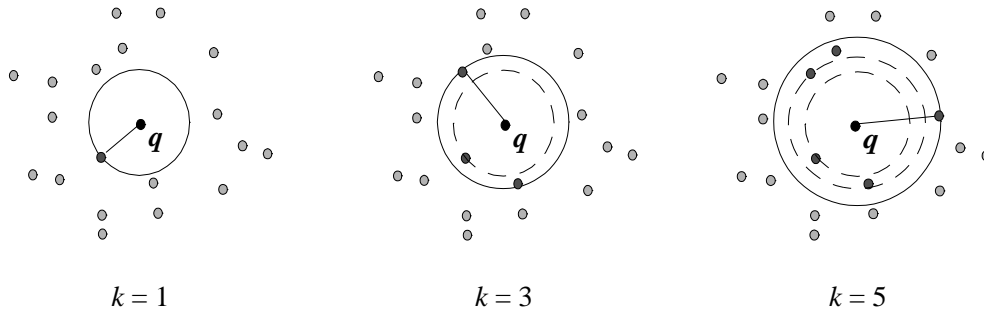


Figure 94: Similarity k -nearest neighbor query.

though the query parameter k is comparatively easy to select, it may be still difficult to provide one single value. Rather, the user may be interested in starting with a very small value, e.g. $k = 3$, and if the answer set does not meet his expectation, the similarity system should be able to generate further similar objects in an incremental “give-me-more” manner. Using the k -nearest neighbor query type for this purpose, the user is forced to increase the value of k and to start another query. This is obviously inefficient since the already generated similar objects are computed once again. Secondly, a user may not accept to see answer objects which he already knows from previous queries. Third, even if a user chooses a rather high value of k , he would like to get the first results soon, i.e. we need a pipelined query processing.

7.1.3 Similarity Ranking Queries

An incremental similarity search is achieved by the so-called *similarity ranking query*. The basic idea of this query type is to rank the database objects in order of their similarity distance.

For reasons of efficiency, the ranking procedure should not be performed and completed in advance at query initialization time. In view of very large databases and of expensive similarity distance functions for complex objects, this course will take too much time until the user will receive the first answer. While incrementally proceeding in the ranking procedure, the next object should be reported shortly after the corresponding user request, as soon as its correct ranking is ensured. Another reason for deferring as much as possible of the ranking procedure is that the user often may be satisfied with only a few answers. In this case, the system has spent too much effort in ranking all the remaining objects in vain.

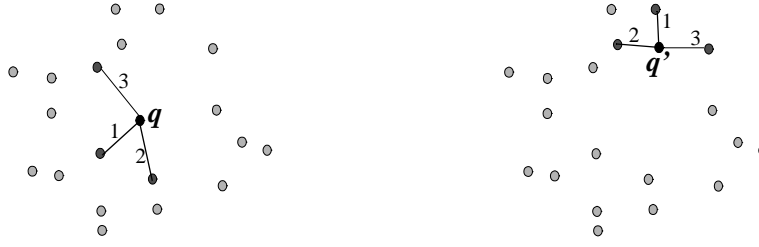


Figure 95: Examples of a q -ranking for two query points q' and q'' .

Definition 21 (Similarity Ranking Query).

Let $q \in O$ be a query object and $DB \subseteq O$ a database. Let $ranked_q: \{1 \dots |DB|\} \rightarrow DB$ be a bijection which ranks our database DB w.r.t. the query object q as follows: $\forall i, j \in \{1 \dots |DB|\}: i < j \Rightarrow simdist(rank_q(i), q) \leq simdist(rank_q(j), q)$. Then, the similarity ranking function $sim_{rank}: O \rightarrow (\{1 \dots |DB|\} \rightarrow DB)$ is defined as:

$$sim_{rank}(q) = ranked_q$$

We write $rank_q(i) = o_i$ for the object o_i that is ranked at position i . Note that in most cases, the ranking is uniquely characterized by this definition. However, if there are several objects in the database that have the same distance to the query object, i.e. $simdist(o_i, q) = simdist(o_j, q)$ for some $i, j \in \{1 \dots |DB|\}$, the order of o_i and o_j is not determined, and there is not a single $rank_q$ -function but a family of ranking functions which we denote by $RANK_q$. Figure 95 provides two examples of similarity ranking queries. In the examples, the top k objects are marked for $k = 3$.

7.1.4 Further Similarity Queries

Besides the three mentioned query types, there exist other similarity queries, as for instance *approximate nearest neighbor queries* and *inverse nearest neighbor queries*.

In *approximate k -nearest neighbor queries* the user also specifies a query point and a number k of answers to be reported. In contrast to exact nearest neighbor queries, the user is not interested exactly in the closest points, but is satisfied with points which are not much further away from the query point than the exact nearest neighbors. The degree of inexactness is a parameter which is also decisive for the efficiency improvement of the query processing.

In *inverse nearest neighbor queries* the user only specifies a query point. Given this query point within a data set, an inverse nearest neighbor query finds all points for which the query point is a nearest neighbor.

7.2 Similarity Queries within Object-Relational Database Systems

All major database vendors have already added object-relational functionality to their relational database servers. In order to achieve a seamless integration of custom object types and predicates within the declarative DDL and DML, ORDBMSs provide the database developer with extensibility interfaces (cf. Chapter 3). In this section we discuss the integration of similarity queries into an ORDBMS. The examples are based on the table schema *CADOBJECTS* (*id*, *geom*) introduced in Chapter 3.

7.2.1 Integration of Range Queries

The declarative integration of similarity range queries can be achieved in a straightforward way. We only have to implement a method *ObjectIsInRange* in order to allow the user to issue the following SQL-statement.

```
SELECT id
FROM CADOBJECTS db
WHERE ObjectIsInRange(:query_obj, db.geom, :eps)
```

Figure 96: SQL-Statement for a range query.

Both a functional and an index based implementation of this method is possible.

7.2.2 Integration of k-nn Queries

In order to detect the *k* nearest neighbors of a given *query_obj*, it would be desirable to post an SQL-statement similar in design to the following one:

```
SELECT id
FROM CADOBJECTS db
WHERE ObjectIsInKNNSet(:query_obj, db.geom, :k)
```

Figure 97: SQL-Statement for a *k*-nn query.

Unfortunately, the integration of *k*-nn queries cannot be done in such a straightforward way. Although it is possible to provide an index based implementation of the predicate *ObjectIsInKNNSet*, it is inherently impossible to provide a functional implementation. If we have only access to the two input objects *query_obj* and *db.geom*, we can not decide whether *db.geom* is one of the *k* nearest neighbors of our query object *query_obj*. On the other hand, an index based implementation, has access to all

tuples of the table *CADOBJECTS* and can therefore produce the k nearest neighbors of a given query object.

Therefore, the database vendors suggest to use hints in order to force the query optimizer to use the index based implementation. If the optimizer hint is not used, an internal database error might occur [Ora 99b].

Note, that the k returned database objects do not have to be ordered according to their distance to the query object. Often, users are also interested in this ordering and the actual distance to the query object. This can be achieved by using *ancillary operators* (cf. Section 3.2.1) [Ora 99b]. By using a common tag (e.g. 1), an ancillary operator, e.g. *kNNDistance*, has a functional implementation that has access to state generated by the index based implementation of the primary operator, e.g. *ObjectIsInKNNSet*. By means of these ancillary operators, we can order our k nearest neighbors according to their distance to the query object, as shown in the following SQL-statement:

```
SELECT id, kNNDistance(1) as distance
FROM CADOBJECTS db
WHERE ObjectIsInKNNSet(:query_obj, db.geom, :k,1)
ORDER BY distance
```

Figure 98: SQL-Statement for a ranked k-nn query.

Frequently, additional constraints have to be taken into consideration for a k -nearest neighbor search. If we want to find those k nearest neighbors of a query object, which consist of at least 1.000.000 voxels, we can use the pseudo column *rownum* in order to detect the desired k tuples.

```
SELECT id, kNNDistance(1) as distance
FROM CADOBJECTS db
WHERE ObjectIsKNN(:query_obj, db.geom, ∞,1) AND
CADOBJECT(db.geom).voxels.count >=1.000.000 AND rownum <= :k
ORDER BY distance
```

Figure 99: SQL-Statement for a ranked k -nn query with additional constraints.

Of course, this statement is rather inefficient, if the implementation of the *ObjectIsInKNNSet* method is not based on an algorithm which supports ranking queries.

If the implementation is not based on an incremental ranking algorithm, we can also invoke the *ObjectIsInKNNSet* method with a last parameter of k instead of ∞ . In this

case the *ObjectIsInKNNSet* predicate might be evaluated multiple times in order to return the desired number of results that satisfy the WHERE clause [Ora 99b]. If the predicate is invoked for the n th time, it returns the $(n - 1) \cdot k + 1$ to $n \cdot k$ nearest neighbors to the query object. This approach is quite similar to the *aggressive strategy* of the *stop after* operator introduced by Carey and Kossmann [CS 97], which might need a *restart* operator to produce the desired number of results¹.

7.2.3 Integration of Ranking Queries

If the *ObjectIsInKNNSet* predicate is based on an incremental ranking algorithm, as for instance the one presented in [HS 95], we can use the SQL-statements of the Figures 97 to 99 without any efficiency problems. Only the necessary number of nearest neighbors are produced, until all constraints within the WHERE clause are fulfilled.

7.3 Access Methods for Similarity Search

In this section, we outline some of the index structures suitable for similarity search on spatial objects. Together with the algorithms of Section 7.4 and 7.5, these index structures can be used to answer similarity queries efficiently. For a more detailed elaboration on spatial access methods suitable for similarity search and on the corresponding query processing techniques, we refer the interested reader to [GG 98] and [BBK 01].

7.3.1 Multi-Dimensional Access Methods

Typically, the similarity search process is a CPU and I/O intensive task and the conventional approach to address this problem is to use a *multidimensional index structure* [GG 98]. The R*-tree [BKSS 90], for example, is an index structure for multidimensional data objects which hierarchically partitions the data space into subpartitions. The concept of minimum bounding rectangles (MBRs) is used to conservatively approximate objects that lie within a subpartition. Since the R*-tree is mainly efficient for low-dimensional feature spaces ($d < 6$, where d denotes the

¹ The implementation of the k -nn operator in Oracle 9i is based on the RKV algorithm [RKV 95], which does not support incremental ranking queries. Therefore, the system might evaluate the k -nn predicate multiple times in order to produce the desired number of results [Ora 99b].

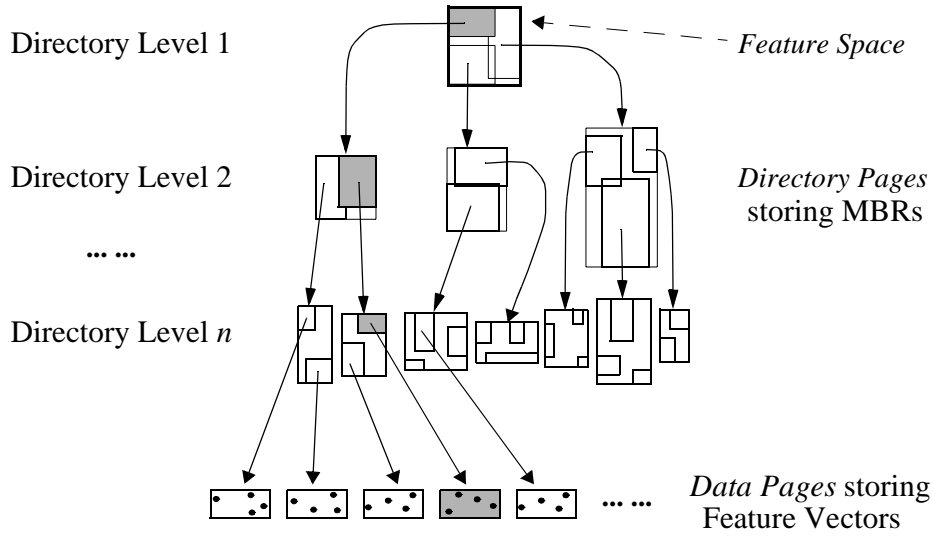


Figure 100: The R*-tree architecture.

dimension), specialized index structures have been proposed which are also efficient for medium-dimensional feature spaces ($d < 20$). Examples are the TV-tree [LJF 94], SS-tree [WJ 96], and X-tree [BKK 96]. However, when the dimension of the feature space is very high (e.g. $d = 50$), even these specialized index structures mostly fail to efficiently process similarity queries. This effect is usually termed as *curse of dimensionality*. In the following, we shortly discuss some of these index structures.

The R-Tree. [Gut 84] is the fundament of a whole family (R-tree, R^+ -tree, and R^* -tree) of height-balanced, multidimensional index structures and was originally proposed for 2-dimensional, spatially extended data objects (e.g. polygons). The R-tree is composed of directory pages and data pages, and follows the paradigm of partitioning the data space hierarchically. Minimum bounding rectangles (MBRs) are used as page regions. MBRs are multidimensional intervals, which minimally enclose a point set. Overlapping regions are allowed, although overlaps are bad for the search performance. Figure 100 shows an R^* -tree where the data pages contain feature vectors.

The TV-Tree. In [LJF 94] Lin, Jagadish and Faloutsos presented the TV-tree which is an R-tree-like index structure. It is designed especially for real data that are amenable to the Karhunen-Loeve-Transform, i.e. the principal component analysis. Such data yield a high variance and therefore a good selectivity in the first few di-

mensions, which are used for cutting branches in query processing. The benefit of this tree lies in its ability to adapt dynamically and use a variable number of dimensions to distinguish between objects or groups of objects. Since this number of required dimensions is usually small, the method saves space and leads to a larger fan-out. As a result, the tree is more compact and shallower. The authors compared the TV-tree with the R*-tree and showed that their method saves up to 80% in disk accesses.

The SS-Tree. White and Jain presented the SS-tree [WJ 96], an R-tree-like index structure that uses minimum bounding spheres (MBSs) instead of minimum bounding rectangles (MBRs). The region description comprises therefore the centroid and the radius. While spheres generally lead to smaller access probability of pages compared to volume-equivalent MBRs, they have the disadvantage that overlap-free splits are often not possible. In an experimental study, it was shown that the SS-tree outperforms the R*-tree by a factor of 2.

The X-tree. In [BKK 96], the X-tree was proposed which is an index structure adapting the algorithms of R*-trees to high-dimensional data using two techniques: First, the X-tree uses an overlap-free split algorithm which is based on the split history of the tree. Secondly, the X-tree is based on the concept of supernodes. Supernodes are directory nodes which are enlarged by a multiple of the block size. In an experimental study with artificial and real-world data sets, the authors showed that for high-dimensional data, the X-tree clearly outperforms the R*-tree and the TV-tree. Furthermore, in [BBKM 99] it was shown how we can integrate the X-tree in a relational database management system.

7.3.2 One-Dimensional Access Methods

Another approach is based on the mapping of a d -dimensional data point into a one-dimensional value and then make use of an existing one-dimensional index such as a B⁺-tree for instance. We call this approach *one-dimensional access methods*. Usually, the performance of the *multi-dimensional indexing* approach is slightly better, however, as the second category makes use of existing and proven technology, there exist some advantages, too. The mapping techniques can be implemented much easier and important issues such as recovery or concurrency control can be considered solved problems, as the techniques make use of existing B⁺-tree indexes. The decision which index structure is appropriate for a given application is very complex

and depends on the data distribution, the query mix, the size and the dimensionality of the database [BBKM 99].

The Pyramid-Tree. Berchtold, Böhm and Kriegel introduced the Pyramid-Tree [BBK 98] which is an index structure that maps a d -dimensional point into a one-dimensional point and uses a B^+ -tree to index the one-dimensional points. In the data pages of the B^+ -tree, the Pyramid-tree stores both the d -dimensional points and the one-dimensional key. Thus, no inverse transformation is required and the refinement step can be done without any further look-ups. The specific mapping used by the Pyramid-tree is called Pyramid-mapping. It is based on a special partitioning strategy that is optimized for range queries on high-dimensional data. The Pyramid-tree achieves the partitioning by first dividing the d -dimensional space into $2d$ pyramids having the center point of the space as their top. In a second step, the single pyramids are cut into slices parallel to the basis of the pyramid forming the data pages.

In an extensive performance analysis the authors show that for almost hypercube shaped queries the Pyramid-technique clearly outperforms the X-tree and the sequential scan on synthetic and real-world data. Most important, for uniformly distributed feature sets, the performance of the Pyramid-technique does not degenerate when the dimension of the feature space increases. However, for queries yielding a low selectivity (i.e. a large answer set) or extremely skewed queries, the sequential scan outperforms the Pyramid-technique.

iMinMax. Recently, a new index scheme, called iMinMax was introduced [OTYB 00]. iMinMax maps high-dimensional points to single dimension values, dependant on their minimum and maximum coordinate values. As other dimension reduction methods, this scheme is also based on the B^+ -tree.

Experiments carried out by the authors showed that this method is significantly more efficient than the Pyramid technique. The authors state that performance difference is expected to increase as the data volume and dimensionality increase, and for skewed data distribution.

iDistance. In [YOTJ 01], an efficient method called iDistance for k -nearest neighbor search in high-dimensional spaces was presented. iDistance partitions the data and selects a reference point for each partition. The data in each cluster are transformed into a single dimensional space based on their similarity with respect to a reference point. This allows the use of a B^+ -tree for k -nn queries.

The authors carried out extensive experiments which reveal that their approach is superior to a linear scan and the iMinMax approach.

The NB-Tree. In [FJ 03] multidimensional points are mapped to a 1D line by computing their Euclidean Norm. The one dimensional values are stored in a B^+ -tree which forms the foundation for the efficient processing of similarity queries.

The authors carried out experiments on both real world and synthetic data. The experiments show that this simple approach outperforms the Pyramid Technique, and the SR-tree (a combination of the R^* -tree and the SS-tree)[KS 97] for many data distributions.

None of the above mentioned mappings from a d -dimensional point to a one-dimensional key is bijective. Furthermore, other examples of one-dimensional access methods which are based on space filling curves such as the *Hilbert-curve* [FR 89][Jag 90] are only theoretically bijective. In practice, this difference between bijective and non-bijective does not exist because bijective mappings are only bijective if infinite precision is applied [BBKM 00].

As an implication, we cannot process a given query by only using the one-dimensional keys. But fortunately, one can at least guarantee that there exist no false drops. On the other hand, the answer set might contain false hits. Therefore, we have to refine the candidate set by taking the d -dimensional feature vectors into account.

7.3.3 Scan-Based Access Methods

Techniques like the VA-file [WSB 98] and the IQ-tree [BBJ+ 00] exploit the fact that the simple sequential scan often provides better query performance than index approaches due to the lack of random disk seeks. The VA-file uses a special compression technique in order to reduce the total amount of data that has to be scanned. The IQ-tree is a sophisticated technique that combines the paradigm of index selectivity and the compression concept of the VA-file. Both techniques are well suited for efficient similarity search in high-dimensional feature spaces.

The VA-File. Weber *et al.* [WSB 98] developed, an index structure that actually is not an index structure. The authors prove in the paper that under certain assumptions, above a certain dimensionality no index structure can process a nearest neighbor query efficiently. Thus, they suggest to speed-up the sequential scan rather than trying to fight a war that already is lost. The basic idea of the VA-file is to keep two files: a bit-compressed (quantized) version of the points and the exact representation of the

points. Both files are unsorted, however, the ordering of the points in the two files is identical. Query processing is equivalent to a sequential scan of the compressed file with some look-ups to the second file whenever this is necessary.

The VA-File outperforms both the R*-Tree and the X-Tree when the dimension is higher than six, but its performance is very sensitive to the actual data distribution.

The IQ-Tree. The idea of quantization based compression has also been integrated to index based query processing. The IQ-tree [BBJ+ 00] combines the ideas of a tree, a scan, and the quantization. The technique performs an I/O optimizing scan through the data pages if the index selectivity is not high enough to compensate for the cost of seek operations. In contrast to the VA-file, the quantization grid of the IQ tree is related to the data page regions and its resolution is automatically optimized during index construction and maintenance.

In the experiments it was shown that the IQ-tree yields a performance that is the “best of two worlds”. In low- and medium-dimensional feature spaces, the IQ-tree performs comparable to the X-tree and clearly outperforms scan-based approaches like the VA-file. On the other hand, when indexing high-dimensional feature sets, the IQ-tree performs comparable to the VA-file and clearly outperforms the X-tree. Thus, the IQ-tree shows a better overall performance than competing techniques for low-, medium-, and high-dimensional feature spaces.

7.3.4 Metric Access Methods

In some application domains, it is not possible to apply a feature transformation to the original data set. However, in most cases it is still possible to provide a similarity distance function to measure similarity. Since the similarity distance function is required to be a metric, the triangle inequality can be used to prune the search space.

In this section, we shortly sketch two metric index structures. For more details about similarity search in metric spaces, we refer the interested reader to [CNBM 01].

M-tree. In [CPZ 97] a new access method, called M-tree, is proposed to organize and search large data sets from a generic “metric space”, i.e. where object proximity is only defined by a distance function satisfying the positivity, symmetry, and triangle inequality postulates. The M-tree extends the domain of applicability beyond the traditional vector spaces. It is interesting to remark that Ciaccia et al. [CPZ 97]

present preliminary results showing that the M-tree can even outperform a well-known vector space data structure (the R*-tree) when applied to a vector space.

An M-tree node consists of a set of database objects, which represent all database objects stored in the corresponding subtrees. Furthermore, each representative stores its covering radius. At query time, the query is compared to all the representatives of the node and the search algorithm enters recursively into all those that cannot be discarded. In Section 10.3.1, an optimized relational implementation of the M-tree is discussed in full detail.

Slim-Tree. The Slim-tree [TTSF 00] is a dynamic tree for organizing metric data sets in pages of fixed size. As the degree of overlap directly affects the query performance of index structures, the Slim-tree tries to reduce the degree of overlap. It is based on new algorithms for inserting objects and splitting nodes.

Results obtained from experiments with real-world data sets show that the Slim-tree outperforms the M-tree by a factor of 35%.

7.3.5 Miscellaneous Access Methods

In this subsection we list some other important access methods which cannot be assigned to any of the foregoing subsections.

Tree-Striping. In [BBK+ 00] a new technique called tree striping was introduced. It generalizes the well-known inverted lists and multidimensional indexing approaches. A theoretical analysis shows that both, inverted lists and multidimensional indexing approaches, are far from being optimal. Consequently, the tree-striping approach proposes the use of a set of multidimensional indexes. The basic idea of tree striping is to use a number k of lower-dimensional indexes.

The experimental evaluation on synthetic and real world data showed that their approach clearly outperforms both multi-dimensional index structures and the inverted list approach.

Voronoi-Approximations. In [BEK+ 98] [BKKS 00] an approach was introduced to overcome the curse of dimensionality problem, by pre-computing the results of any nearest neighbor query. This pre-computing corresponds to a computation of the Voronoi cell of each data point. In a second step, the conservative approximations of the Voronoi cells are stored in a high-dimensional index structure. As a result, nearest neighbor search corresponds to a simple point query on the index

structure. Although this technique is based on a precomputation of the solution space, it is dynamic, i.e. it supports insertions of new data points.

In the experiments the authors compared this approach to the X-tree and showed that their new approach outperforms the X-tree up to a factor of 4.

Parallel Nearest-Neighbor Search. In [BBB+ 97] the authors present a new parallel method for fast nearest-neighbor search in high-dimensional feature spaces. The core problem of designing a parallel nearest-neighbor algorithm is to find an adequate distribution of the data onto the disks. The basic idea of their data declustering technique is to assign the buckets corresponding to different quadrants of the data space to different disks.

The approach was evaluated by using large amounts of real data and by comparing the approach to other declustering methods. It was shown that their new approach outperforms other declustering methods, e.g. the Hilbert approach, by a factor of up to 5.

7.4 One-Step Similarity Query Processing

In this section we look at one-step similarity query processing. All objects are modelled by simple objects, e.g. numerical values or feature vectors, which can efficiently be managed within one- or multi-dimensional relational index structures.

All query types introduced in section 7.1 can be evaluated by a single sequential scan on the complete index data. In this case the database is read in very large blocks, determined by the amount of main memory available to the query processing. After reading a block from disk, the CPU processes each object. Thereafter, the next block is read.

Obviously, this might not be the most efficient way to process similarity queries. In the following, we will introduce algorithms for the query types discussed in Section 7.1, which exploit the pruning power of the underlying index structures. The algorithms have originally been introduced for different multi-dimensional index structures. We will exemplarily introduce them for the family of the R-trees, which might contain the most important multi-dimensional index structures. Furthermore, we show how the different query types can be processed on the B^+ -tree.

```

ALGORITHM RangeQuery (Object query, float  $\epsilon$ )
BEGIN
  PriorityQueue queue;
  queue.insert (root);
  WHILE NOT queue.isEmpty() DO
    Element first = queue.pop();
    CASE first OF
      DirNode:
        FOR EACH child in first DO
          IF RangeIntersectsRegion (child.Region, query,  $\epsilon$ ) THEN
            queue.insert (child);
      DataNode:
        FOR EACH object in first DO
          IF IsPointInRange (object, query,  $\epsilon$ ) THEN
            queue.insert (object);
      Object:
        report (first);
    END CASE;
  END DO;
END.

```

Figure 101: Range query processing on R-trees.

7.4.1 Index based Range Queries

The algorithm for a range query returns a set of points contained in the query range, i.e. $\text{sim}_{\text{range}}(q, \epsilon) = \{o \in DB \mid \text{simdist}(o, q) \leq \epsilon\}$. The size of the result set is previously unknown and may reach the size of the entire database. The algorithm can be formulated independently from the applied distance function.

R-tree. In Figure 101 an algorithm for range queries on R-trees is depicted. It is important to provide effective and efficient tests for the predicates *IsPointInRange* and *RangeIntersectsRegion*, which in the case of the R-tree family can easily be achieved.

B⁺-tree. In the case our objects are modelled by simple numerical values, and are stored within a B⁺-tree, a range query $\text{sim}_{\text{range}}(q, \epsilon)$ can be processed by an index range scan on the leaf level of the B⁺-tree. The scan starts at $q - \epsilon$ and ends at $q + \epsilon$.

7.4.2 Index based k-nn Queries

The algorithm for a k-nn query $\text{sim}_{\text{kn}}(q, k) = NN_q(k)$ returns the k points closest to q with respect to a given distance function *simdist*, so that the following expression is true: $\forall o_1 \in NN_q(k), \forall o_2 \in DB \setminus NN_q(k): (\text{simdist}(o_1, q) \leq \text{simdist}(o_2, q))$.

R-tree. For nearest neighbor search on multidimensional index structures, we present the algorithm of [RKV 95] which is designed for R-trees. Though developed for retrieving the single nearest neighbor, the method can easily be extended to k -nearest neighbor search.

The algorithm follows the classic *branch-and-bound* paradigm, and as heuristics to control the search in the underlying index, the functions *mindist* and *minmaxdist* are used. While *mindist* returns the minimum distance of any point in the MBR and the query point, *minmaxdist* determines the minimum of all distances that maximally can occur for objects in the MBR and the query point. While *minmaxdist* has a strong heuristic power and is used to prune paths of the search tree, only the use of *mindist* guarantees the minimum number of page accesses as shown in theoretical analyses [BBKK 97].

B⁺-tree. In the case our objects are modelled by simple numerical values, and are stored within a B⁺-tree, a k -nn query $NN_q(k)$ can be processed by two index range scans on the leaf level of the B⁺-tree. Both range scans start at q and proceed in opposite directions. At the beginning, we fetch a tuple from both cursors. Thereafter, we always fetch data from that cursor, where the last fetched data object is closer to q . After having fetched k values from our two open cursors, we close both cursors, and return the k tuples to the caller. Obviously, this algorithm inherently supports ranking queries as well.

7.4.3 Index based Ranking Query

In [HS 95], an algorithm for ranking in spatial databases in the context of 2D geographic information systems (GIS) is presented. The algorithm requires a multi-dimensional access method that is hierarchically managed in terms of container blocks. This paradigm demands that a container at least encloses the space which is enclosed by subordinate container blocks. While several access methods follow this structure, e.g. R-trees, the authors focus on the PMR quadtree [NS 87]. Every quadtree structure hierarchically decomposes the space into 2^d partitions for d dimensions which results in exorbitant numbers of childs for each node for high-dimensional spaces ($10 \leq d$).

R-tree. In Figure 102, we present a version of the algorithm that has been adapted to the family of R-trees. This version of the algorithm is simpler than the original version of [HS 95] which has to manage the characteristics of the PMR quadtree. Whereas the R-tree follows the paradigm of overlapping regions, the PMR quadtree manages

```

ALGORITHM RankingQuery (Object query)
BEGIN
  PriorityQueue queue;
  queue.insert (0, root);
  wait (getnext_is_called);
  WHILE NOT queue.isempty() DO
    Element first = queue.pop();
    CASE first OF
      DirNode:
        FOR EACH child in first DO
          queue.insert (mindist (query, child.region), child);
      DataNode:
        FOR EACH object in first DO
          queue.insert (distance (query, object), object);
      Object:
        report (first);
        wait (getnext_is_called);
    END CASE;
  END DO;
END.

```

Figure 102: Incremental ranking query processing on R-trees.

spatial objects by clipping. Thus, a single object may be stored in more than one leaf of the tree, and the ranking algorithm has to manage duplicate results which are eliminated by carefully controlling the order by which the results are reported.

7.5 Multi-Step Similarity Query Processing

Due to the immense and even increasing size of current CAD databases, strong efficiency requirements have to be met. Thus, for the evaluation of complex similarity queries, fast processing is important. Following the paradigm of multi-step query processing [OM 88] [BKSS 94], a filter and refinement architecture produces and reduces candidate sets from the database, yielding an overall result that contains the correct answer, i.e. producing neither false hits nor false drops (cf. Section 2.2.2).

Filter steps are based on approximated objects. For efficiency reasons, these filter steps might be supported by suitable index structures (cf. Section 7.3). Approximations might be feature vectors stored in a spatial access method (SAM), or simply numerical values which can be managed by a B^+ -tree. Refinement steps discard false positive candidates, i.e. false hits, but are not able to reconstruct false negatives, i.e. false drops, that have been dismissed by a previous filter step. Thus, the strong requirement for filter steps is to prevent from false drops, but the quality of a filter step

depends on its selectivity. In general, the evaluation of a single object within a refinement step is expensive, and the number of candidates should be as small as possible. Thus, the less candidates a filter step passes to a subsequent refinement step, the better is the performance of the overall query processing.

Based on this multi-step query paradigm, we will look at some approved approaches for similarity query processing as presented in the literature. After introducing the lower bounding criterion, which forms the foundation of multi-step similarity query processing, we discuss algorithms for range queries [FRM 94], for k -nearest neighbor queries [KSF+ 96], and for incremental k -nearest neighbor queries [SK 98].

7.5.1 The Lower-Bounding Property

As outlined in Section 2.2.2 conservative approximations of the objects guarantee that a multi-step query processor does not produce any false drops. In Part II of this thesis it was shown that this concept is decisive for efficiently detecting intersecting CAD objects. We can also apply this concept to similarity queries. If a *filter distance* d_f lower bounds the *object distance* d_o no false drops are produced by a multi-step query processor.

Definition 22 (*Lower-Bounding Property*).

Let O be a set of objects. A filter distance function d_f and an object distance d_o fulfill the lower-bounding property, if d_f underestimates d_o in any case, i.e. for all objects $o_1, o_2 \in O$: $d_f(o_1, o_2) \leq d_o(o_1, o_2)$ holds.

The *lower-bounding criterion* is closely related to the concept of *conservative approximations*. If the lower-bounding property holds, then the following expression is true as well:.

$$\forall q \in O, \forall \epsilon \in IR: \{o \in DB: d_o(o, q) < \epsilon\} \subseteq \{o \in DB: d_f(o, q) < \epsilon\}$$

Consequently, the lower-bounding property ensures that no false drops occur.

7.5.2 Multi-Step Range Queries

In [FRM 94], a multi-step algorithm is presented that performs similarity search for complex distance functions by using appropriate filter distance functions d_f on the approximation $Appr(o)$ of the objects o . An approximation $Appr(o)$ might be an n -dimensional feature vector $F_n(o)$ managed in an R-tree, or a one-dimensional value

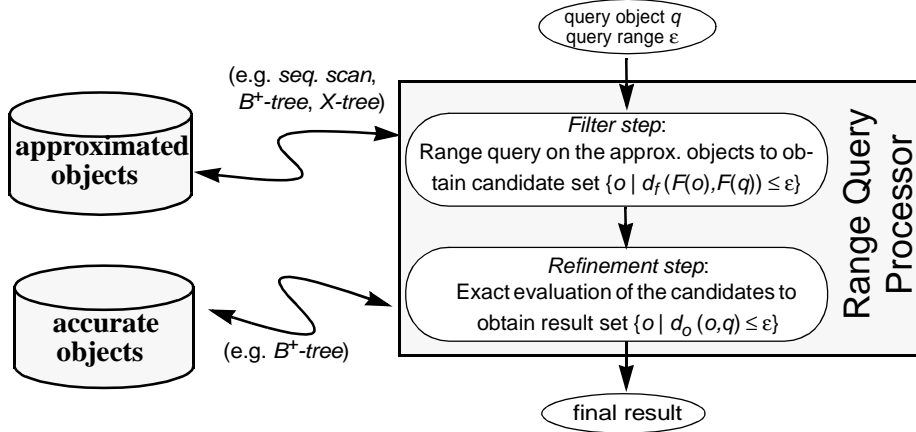


Figure 103: Similarity range query processor.

$F_1(o)$ managed within a B^+ -tree. The corresponding lower-bounding property guarantees that no false drops occur when applying the filter step. The efficiency of the method depends on the selectivity of the filter distance function and on the underlying access method. Figure 103 provides a schematic illustration of the algorithm.

7.5.3 Multi-Step k-nn Queries

In the preceding, we investigated algorithms for similarity search that are restricted to similarity models which are completely defined by a distance function of low- or medium-dimensional feature vectors. Note that the numerical values managed in the B^+ -tree can also be regarded as one-dimensional feature vectors. In practice, it often occurs that similarity distance functions have a higher complexity and may not be represented by a simple feature vector distance, or that they are too high in their dimensionality to be efficiently managed by a multidimensional index structure.

In the context of fast nearest neighbor search in medical image databases, Korn et al. [KSF+ 96] suggest an algorithm for k -nearest neighbor search that follows the multi-step query processing paradigm. In Figure 104, we illustrate the architecture of the algorithm and indicate the interplay with the underlying index structure, managing the approximated objects.

The k -nearest neighbor search $sim_{knn}(q, k) = NN_q(k)$ is performed in four steps. First, we determine the primary candidates by performing a k -nearest neighbor search on the approximated data around $Appr(q)$ w.r.t. d_f . In a second step, we deter-

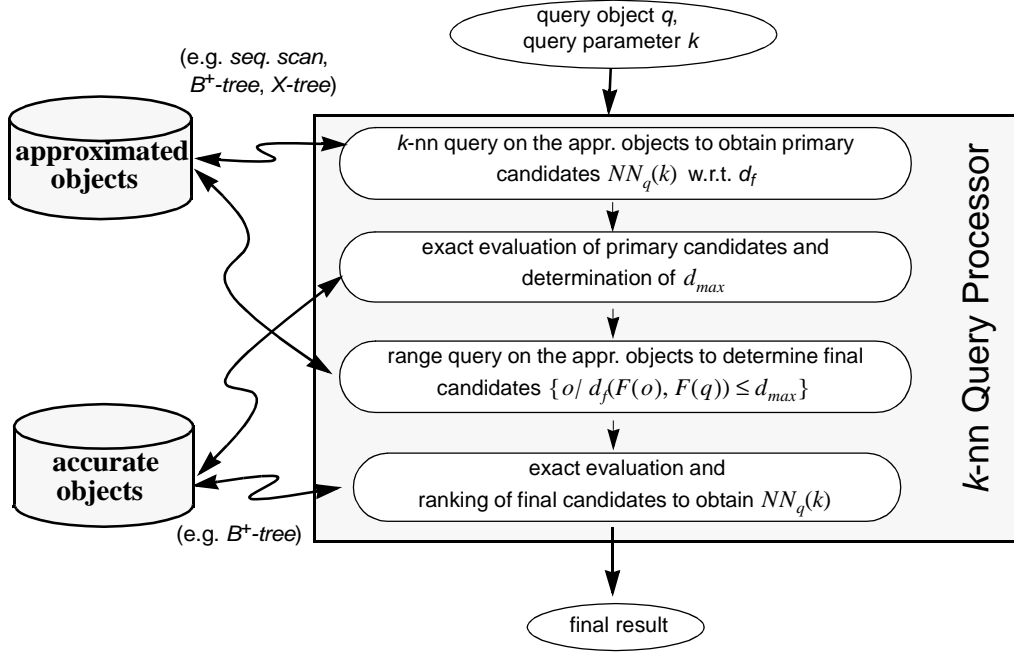


Figure 104: k -nearest neighbor query processor according to [KSF+ 96].

mine the range for the primary candidates o , i.e we determine $d_{max} = \max\{d_o(o, q) \mid o \in NN_q(k) \text{ wrt. } d_f\}$. Third, the final candidates are selected, by performing a range query on the approximations to obtain $\{o \mid d_f(F(o), F(q)) \leq d_{max}\}$. Finally, we rank the final candidates o according to $d_o(o, q)$, and report the top k objects.

7.5.4 Multi-Step Ranking Queries

In [SK 98] Seidl and Kriegel formally introduced a criterion for the optimality of multi-step k -nearest neighbor algorithms with respect to the number of candidates for which an exact evaluation of the object distance has to be performed. Furthermore, they presented an optimal multi-step k nearest neighbor algorithm. The algorithm is depicted schematically in Figure 105.

The algorithm has two basic components: By the incremental ranking query on the underlying access method, candidates are iteratively generated in ascending order according to their filter distance d_f to the query object. The second major component is the *result* list that manages the k nearest neighbors of the query object q within the current candidate set, keeping step with the candidate generation. The current k -th distance is held in d_{max} which is set to infinity until the first k candidates are retrieved

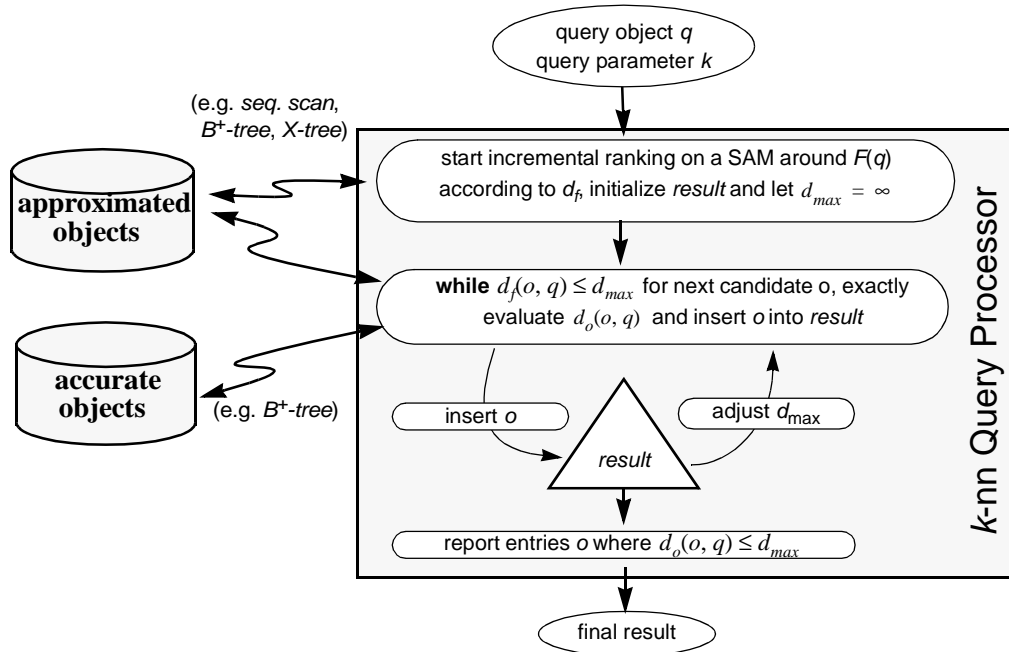


Figure 105: Multi-step query processor for optimal k -nearest neighbor search.

from the index and evaluated. During the algorithm d_{max} will be decreased exactly down to $\text{simdist}_{q,k}$. Based on this fact the termination of the algorithm is controlled.

Ranking Search. Although the algorithm presented in [SK 98] is based on an incremental ranking on a SAM, it does not support ranking queries itself. In order to support these queries for complex distance functions d_o we have to slightly alter the algorithm. Figure 106 shows that we have to add an additional while-loop in which we test whether the first object in our result list o_1 is closer to q than $d_f(o, q)$, whereby o denotes the last object retrieved from our ranking index structure. If this is the case, we can report o_1 and delete it from our result list.

7.6 Similarity Models

In the last ten years, an increasing number of database applications has emerged for which efficient and effective support for similarity search is substantial. The importance of similarity search grows in application areas such as multimedia, medical imaging, molecular biology, computer aided engineering, marketing and purchasing assistance, etc. [Jag 91] [AFS 93] [MG 93] [FEF+ 94] [FRM 94] [ALSS 95]

```

ALGORITHM RankingSearch (Object  $q$ )
BEGIN
  initialize  $ranking = \text{index.ranking}(F(q), d_p)$ ;
  initialize  $result = \text{new sorted list } \langle \text{key, object} \rangle$ ;
  initialize  $d_{max} = \infty$ ;
  wait (getnext_is_called);
  WHILE  $o = ranking.getnext$  DO
    BEGIN
       $result.insert(d_o(o, q), o)$ ;
       $d_{max} = result[1].key$ ;
      WHILE  $d_{max} \leq d_f(o, q)$  AND NOT  $result.isempty()$  DO
        BEGN
          report ( $result[1].object$ );
           $result.delete(1)$ ;
          IF NOT  $result.isempty()$  THEN
             $d_{max} = result[1].key$ ;
          END IF;
        END DO;
      END DO;
    END.

```

Figure 106: Multi-step ranking queries.

[BKK 97a] [BKK 97b] [BK 97] [Kei 99]. Particularly, the task of finding similar shapes in 2D and 3D becomes more and more important. Examples for new applications that require the retrieval of similar 3D objects include databases for molecular biology, medical imaging and computer aided design.

In this section, we discuss some of the approaches presented in the literature to establish similarity measures. We provide a classification of the techniques into feature-based models (cf. Section 7.6.1) and direct geometric models (cf. Section 7.6.2).

7.6.1 Feature-Based Similarity Search

Feature Transformation. As distance functions form the foundation of similarity search, we need object representations which allow efficient and meaningful distance computations. A common approach is to represent an object by a numerical vector, resulting in straightforward distance functions. In this case a feature transformation extracts distinguishable spatial characteristics which are represented by numerical values and grouped together in a feature vector.

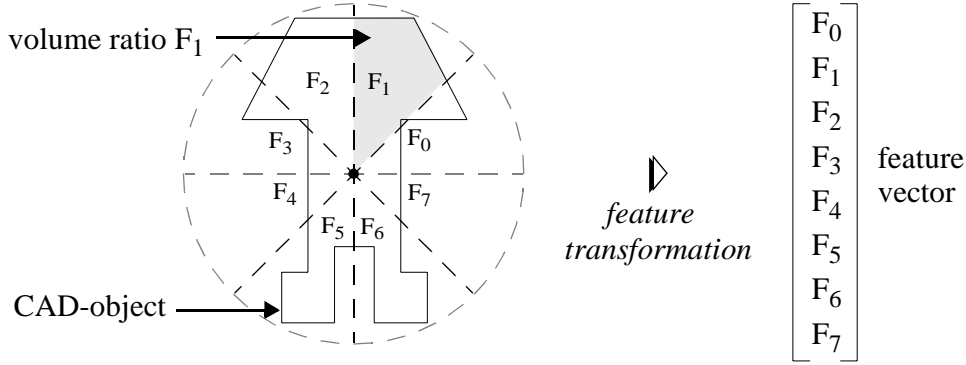


Figure 107: The Section Coding feature transformation.

A 2-dimensional CAD object, for instance, can be transformed into an 8-dimensional *feature vector* by specifying the perimeter around the object's center of gravity first. Then, the perimeter is partitioned into $d = 8$ sectors and the volume of the object lying in each sector is determined [BKK 97b]. Thus, the resulting feature vector consists of eight ratio values (cf. Figure 107).

Using a feature transform, the objects are mapped onto a feature vector in an appropriate multidimensional feature space. The similarity of two objects is then defined as the proximity of their feature vectors in the feature space: The closer their feature vectors are located, the more similar two objects are considered. Most applications use the Euclidean metric (L_2) to evaluate the feature distance, but there are several other metrics commonly used, e.g. the Manhattan metric (L_1), and the Maximum metric (L_∞).

Feature-Based Similarity Models. Several reasons lead to the wide use of feature-based similarity models: First, the more complex the objects are, the more difficult it may be to find an appropriate similarity distance function. A second reason wherefore feature-based similarity models are quite popular is that they may be easily tuned to fit to specific applications. In general, this task is performed in close cooperation with domain experts who specify appropriate features and adapt them to the specific requirements. Since the existing techniques for query processing are independent from the particular definition of the features, efficient support may be provided without an in-depth insight into the application domain.

Examples where the paradigm of feature-based similarity has been successfully applied to the retrieval of similar spatial objects include structural features of 2D contours [GM 93] [MG 93] [MG 95], angular profiles of polygons [BMH 92], rectangular covers of regions [Jag 91], algebraic moment invariants [TC 91][FEF+ 94], and 2D section coding [BKK 97a]. Non-geometric applications include similarity search on time series [FRM 94][AFS 93], and on color histograms in image databases [FEF+ 94][NBE+ 93][HSE+ 95], among several others.

Agrawal et al. present a method for similarity search in a sequence database of one-dimensional data [AFS 93]. The sequences are mapped onto points of a low-dimensional feature space using a Discrete Fourier Transform, and then a PAM is used for efficient retrieval. This technique was later generalized for subsequence matching [FRM 94], and searching in the presence of noise, scaling, and translation [ALSS 95]. However, it remains restricted to one-dimensional sequence data.

Mehrotra and Gary suggest the use of boundary features for the retrieval of shapes [MG 93] [GM 93]. Here, a 2D shape is represented by an ordered set of surface points, and fixed-sized subsets of this representation are extracted as shape features. All of these features are mapped to points in multidimensional space which are stored using a Point Access Method (PAM). This method is essentially limited to two dimensions.

Jagadish proposes a technique for the retrieval of similar shapes in two dimensions [Jag 91]. He derives an appropriate object description from a rectilinear cover of an object, i.e. a cover consisting of axis-parallel rectangles. The rectangles belonging to a single object are sorted by size, and the largest ones serve as retrieval key for the shape of the object. This method can be generalized to three dimensions by using covers of hyperrectangles, as we will see in Chapter 8.

Histograms as Feature Vectors. Histograms represent a quite general class of feature vectors which have been successfully applied to several applications. For any arbitrary distribution of objects, a histogram represents a more or less fine grained aggregation of the information. The general idea is to completely partition the space of interest into disjoint regions which are called cells, and to map every object onto a single bin or to distribute an object among a set of bins of the corresponding histogram. Then a histogram can be transformed directly into a feature vector by mapping each bin of the histogram onto one dimension (attribute) of the feature vector. The histogram approach applies to geometric spaces as well as to non-geometric spaces.

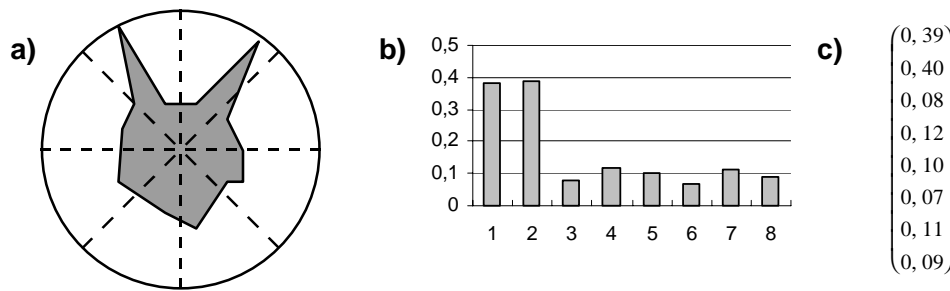


Figure 108: Section coding of 2D regions.

a) Original object, b) Corresponding histogram, c) Corresponding feature vector

A popular example for the use of histograms to define the similarity of complex objects is the color histogram approach which is a core component of the QBIC system [NBE+ 93][FEF+ 94]. Among other techniques, color histograms are used to encode the percentage of colors in an image [SH 94][HSE+ 95]. Our second example is taken from a spatial database application: The 2D section coding approach [BKK 97b] represents a particular histogram technique that is used in the S3 system [BK 97] for the retrieval of similar mechanical parts. For each object, the circumscribing circle is decomposed into a fixed number of sectors around the center point. For each sector, the fraction of the area is determined that is overlapped by the object. Altogether, the resulting feature vector is a histogram over the 2D, whose bins represent the corresponding 2D sectors. Figure 108 illustrates the technique by an example with 8 sectors.

In [KKS 98] [AKKS 99] the retrieval of similar 3D objects from a biomolecular database was investigated. The introduced models are based on 3D shape histograms, where three different approaches were used for space partitioning: shell bins, section bins and combined bins (cf. Figure 109). Unfortunately, these models are not inherently suitable for voxelized data which are axis-parallel.

7.6.2 Geometry-Based Similarity Search

A class of models that is to be distinguished from the feature-based techniques are the similarity models that are defined by directly using the geometry. Two objects are considered similar if they minimize a distance criterion that is purely defined by the geometry of the objects. Examples include the similarity retrieval of mechanical

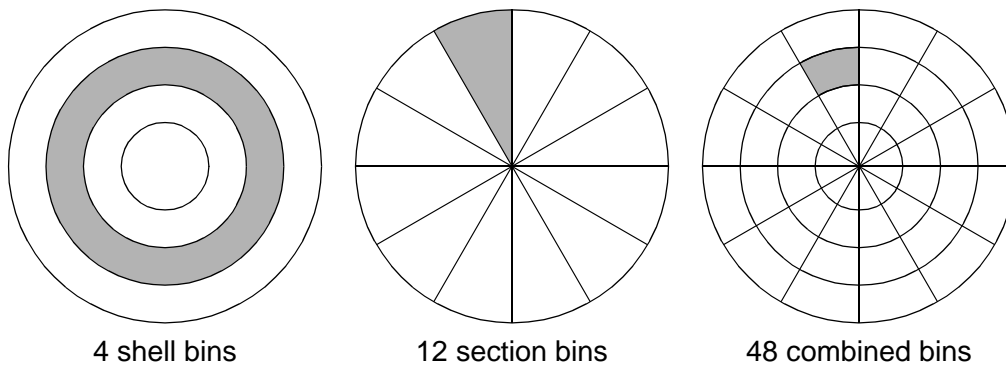


Figure 109: Shells and sections as basic models for shape histograms.
(In each of the 2D examples, a single bin is marked)

parts, the difference volume approach, and the approximation-based similarity model for 3D surface segments:

Rotational Symmetric Mechanical Parts. In [SKSH 89], a method is presented to retrieve similar mechanical parts from a database. The similarity criterion is defined in terms of tolerance areas which are specified around the query object. All objects that fit into the tolerance area count for being similar. Although the parts are 3D, only their 2D contour is taken into account for the retrieval technique.

Difference Volume Approach. The difference volume or error volume of spatial objects is a promising approach which has been already successfully applied to medical images, for instance [Hig 90][Vin 91]. Furthermore, extensions such as the combination with methods from mathematical morphology have been investigated on a tumor database [KSF+ 96]. However, they considered only 2D images. A competing approach is based on a new geometric index structure as suggested in [Kei 99]. The basic idea of this solution is to use the concept of hierarchical approximations of the 3D objects to speed up the search process.

Approximation-based Similarity of 3D Surface Segments. The retrieval of similar 3D surface segments is a task that supports the docking search for proteins in biomolecular databases. Following the approximation-based model, the similarity of 3D surface segments is measured by their mutual approximation error with respect to a given multi-parametric surface function which serves as the underlying approximation model. To state it simply, two segments are the more similar, the better they fit to the approximation of the partner segment [KSS 97].

Class	Definition of Similarity	Examples
feature-based similarity	similarity is proximity in the feature space	<ul style="list-style-type: none"> • rectangular cover of regions [Jag 91] • algebraic moment invariants [TC 91] • 2D contour features [GM 93][MG 95] • angular profiles of polygons [BMH 92] • section coding [BKK 97a] • time series [AFS 93][FRM 94] • color histograms [NBE+ 93][FEF+ 94] [HSE+ 95]
geometric similarity	similarity is directly defined by geometry	<ul style="list-style-type: none"> • symmetric mechanical parts [SKSH 89] • difference volume [Vin 91][KSF+ 96][Kei 99] • 3D surface segments [KSS 97]

Figure 110: Classification of complex similarity models.

7.6.3 Summary

In Figure 110, we summarize our classification of similarity models into feature-based approaches and direct geometry-based proposals. The list of examples is by no means complete but provides an impression of the potentials of both paradigms. In this work, we introduce effective similarity models for CAD objects, which rely on the feature based histogram approach.

Chapter 8

Similarity Models for Voxelized CAD Data

Although considerable work on similarity search in database systems has been published, many of the previous approaches deal only with one- or two-dimensional data, such as time series, digital images or polygonal data. Most of these approaches do not support three-dimensional objects.

In this chapter, we concentrate on similarity models for voxelized CAD data. In Section 8.1, we start with introducing object similarity functions, emphasizing invariance properties which are required for effective similarity search in the area of virtual engineering. In Section 8.2, three different *space partitioning* similarity models for voxelized CAD data are presented, namely the *volume model*, the *solid-angle model* and the *eigen-value model*. In Section 8.3, we turn our attention to *data partitioning* similarity models. We first discuss the *cover-sequence model* which serves as a starting point for the *vector set model*. In contrast to the other four models, the vector set model uses sets of feature vectors for representing an object instead of single feature vectors. The discussion of the effectiveness and efficiency of the five presented models is deferred to the following two chapters.

8.1 Object Similarity

The degree of similarity between two objects heavily depends on the chosen distance function. Ideally a distance measure has the properties of a metric.

Definition 23 (Metric).

Let M be an arbitrary data space. A metric is a mapping $dist: M \times M \rightarrow IR$ such that for all $x, y, z \in M$ the following statements hold:

- $dist(x, y) = 0 \Leftrightarrow x = y$ (reflexivity)
- $dist(x, y) = dist(y, x)$ (symmetry)
- $dist(x, z) \leq dist(x, y) + dist(y, z)$ (triangle inequality)

Based on *metric distance* functions, we can define *metric object similarity*.

Definition 24 (Metric Object Similarity).

Let O be the domain of the objects and $F: O \rightarrow M$ be a mapping of the objects into a metric data space M . Furthermore, let $dist: M \times M \rightarrow IR$ be a metric distance function. Then a metric object similarity function $simdist: O \times O \rightarrow IR$ is defined as follows:

$$simdist(o_1, o_2) = dist(F(o_1), F(o_2)).$$

Often, the d -dimensional vector space IR^d is used. By means of a suitable feature transformation (cf. Section 7.6.1) distinguishable spatial characteristics are extracted and grouped together in a numerical feature vector. In this important special case, the similarity of two objects can be defined as follows.

Definition 25 (Feature-Based Object Similarity).

Let O be the domain of the objects and $F: O \rightarrow IR^d$ be a mapping of the objects into the d -dimensional feature space. Furthermore, let $dist: IR^d \times IR^d \rightarrow IR$ be a distance function between two d -dimensional feature vectors. Then a feature-based object similarity function $simdist: O \times O \rightarrow IR$ is defined as follows:

$$simdist(o_1, o_2) = dist(F(o_1), F(o_2)).$$

There exist a lot of distance functions which are suitable for similarity search. In the literature, often the L_p -distance is used, e.g. the Euclidian distance ($p = 2$).

$$d_{euclid}(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|_2 = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$$

8.1.1 Normalization of CAD Data

For effective similarity search it is often required to meet invariance properties with respect to a certain class of transformations, i.e. applying a transformation from this class to an object should have no influence on the result of the similarity function. This leads to the following definition.

Definition 26 (*Invariance*).

Let O be the domain of the objects and $simdist: O \times O \rightarrow IR$ be a metric object similarity function. $simdist$ is invariant with respect to a class of transformations C , iff for all objects $o_1, o_2 \in O$ and all transformations $T \in C$ holds:

$$simdist(o_1, o_2) = simdist(T(o_1), o_2) = simdist(o_1, T(o_2))$$

Invariance can be achieved by applying appropriate transformations to the objects in the database. This is called the normalization of data. Invariance properties relevant for similarity search in CAD databases are *scaling*, *translation*, *rotation* and *reflection* invariances. It depends on the user as well as on the chosen similarity model which invariances have to be considered for a particular application. Taking the desired normalization of the data into account, we get the following extended similarity definition.

Definition 27 (*Extended Metric Object Similarity*).

Let O be the domain of the objects and $F: O \rightarrow M$ be a mapping of the objects into a metric data space M . Furthermore, let $dist: M \times M \rightarrow IR$ be a metric distance function, and let C be the set of all user-dependent combinations of scaling, translation, rotation and reflection transformations. Then an extended metric object similarity function $simdist: O \times O \rightarrow IR$ is defined as follows:

$$simdist(o_1, o_2) = \min_{T \in C} \{dist(F(o_1), F(T(o_2)))\}$$

We achieve invariance by taking the minimum of the distances between object o_1 and all transformations out of C applied to object o_2 . In the next four sections, we discuss each of the mentioned invariance properties in more detail. In particular we describe the corresponding transformation matrices M_T using homogeneous coordinates [Gri 92][NS 86]. By applying such a transformation matrix M_T , each voxel $v = (x, y, z)^T$ is mapped onto $v' = (x', y', z')^T$ according to the following equation:

$$M_T \cdot (x, y, z, 1)^T = (x', y', z', 1)^T.$$

The transformation matrices M_T are different for each object o , depending on its minimal bounding box $MBB_o = (x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max})$. These minimal bounding boxes have to be updated after each transformation in order to obtain the correct transformation matrices M_T for the next step.

In the following, we will discuss the different invariances and present the corresponding transformation matrices.

Scaling Invariance. The actual size of the different objects in a CAD database can vary from a few millimeters to several meters. In order to compare the shape of the objects, we scale them to a uniform size. We fit each voxelized object into a cubic voxel space with a predefined extension r in each of the three dimensions. This can be achieved by applying the following transformation with appropriate parameters s_x , s_y and s_z .

$$M_{scal} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

We distinguish between two scaling methods.

- *Proportional scaling:* The objects are scaled proportionally with respect to the three coordinate axes. The shape of the objects is preserved. Using this method the scaling factors s_x , s_y and s_z are equal and can be determined like this: $s_x = s_y = s_z = \frac{r}{\Delta mbb_o}$ where Δmbb_o is the maximal extension of the minimal bounding box with respect to the three coordinate axes. Δmbb_o can be computed as follows: $\Delta mbb_o = \max \{ x_{max} - x_{min}, y_{max} - y_{min}, z_{max} - z_{min} \}$.
- *Non-proportional scaling:* Here the extensions of the objects are adjusted to the size of the voxel space independently for each of the coordinate axes. This way the voxel space is used optimally, but the shape of the objects is distorted. Small differences of the shape in dimensions with low extension are amplified to a high degree. The scaling factors s_x , s_y and s_z result from the ratio of the extension of the voxel space and the extension of the object for each dimension.

$$s_x = \frac{r}{x_{max} - x_{min}}, \quad s_y = \frac{r}{y_{max} - y_{min}}, \quad s_z = \frac{r}{z_{max} - z_{min}}$$

We store each object normalized with respect to proportional scaling in the database. Furthermore, we store the scaling factor, so that we can (de)activate scaling

invariance depending on the users needs at runtime, as the actual size of the parts may or may not exert influence on the similarity model.

Translation Invariance. CAD objects are designed and constructed in a standardized position, normalized to the center of the coordinate system. So similarity models for CAD data should recognize similar parts, independently of their spatial location. The four, respectively five, tires of a car are similar, although they are located differently.

Therefore we move each object, i.e. each voxel of the object, to a uniform position in the voxel space, so that the center of the minimal bounding boxes lies in the origin of the coordinate system. The corresponding matrix M_{trans} looks like this:

$$M_{trans} = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The parameters t_x , t_y and t_z are the coordinates of the center of the minimal bounding box.

$$t_x = \frac{x_{min} + x_{max}}{2}, \quad t_y = \frac{y_{min} + y_{max}}{2}, \quad t_z = \frac{z_{min} + z_{max}}{2}$$

We store each object normalized with respect to translation in the database.

Rotation Invariance. In general, we can apply principal axis transformation in order to achieve invariance with respect to rotation. Here, the idea is to map the position of each object in such a way that the principal axis of each object is parallel to the coordinate system.

In this thesis, we pursue another approach. In the case of CAD applications, not all possible rotations are considered, but only 90°-rotations. This yields up to 24 different possible positions for each object. The transformation matrices for 90°-rotation around the X, Y and Z axes are listed here:

$$M_{rot}^X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M_{rot}^Y = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M_{rot}^Z = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reflection Invariance. Reflected parts, e.g. the right and left front door of a car, should be recognized as similar as far as design is concerned. If we look at the production, reflected parts are no longer similar and have to be treated differently.

To reflect an object with respect to the X, Y or Z axis, one of the following transformation matrices can be used:

$$M_{ref}^X = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M_{ref}^Y = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad M_{ref}^Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Taking 90°-rotations as well as reflection into account, we may obtain up to $24 \times 2 = 48$ varying positions. We could achieve 90°-rotation and reflection invariance by storing 48 different feature vectors for each object in the database or by carrying out 48 different permutations of the query object at runtime.

To sum up, a similarity model for CAD data should take translation and rotation invariances into account whereas reflection and scaling invariances have to be tunable. Throughout our experiments, we considered invariance with respect to scaling, translation, reflection, and 90°-rotation by permuting the query object.

8.2 Space Partitioning Similarity Models

In this section, we discuss three different *space partitioning* similarity models suitable for voxelized CAD data, namely the *volume model*, the *solid-angle model* and the *eigen-value model*. Each of the models is based on shape histograms.

8.2.1 Shape Histograms for Voxelized CAD Data

Shape histograms are based on a complete partitioning of the data space into disjoint cells which correspond to the bins of the histograms. We divide the data space into axis parallel, equi-sized partitions (cf. Figure 111). This kind of space partitioning is especially suitable for voxelized data, as cells and voxels are of the same shape, i.e. cells can be regarded as coarse voxels. The data space is partitioned in each dimension into p grid cells. Thus, our histogram will consist of $k \times p^3$ bins where $k \in \mathbb{N}$ depends on the model specifying the kind and number of features extracted from each cell. For a given object o , let $V^o = \{v \in V_i^o \mid 1 \leq i \leq p^3\}$ be the set of

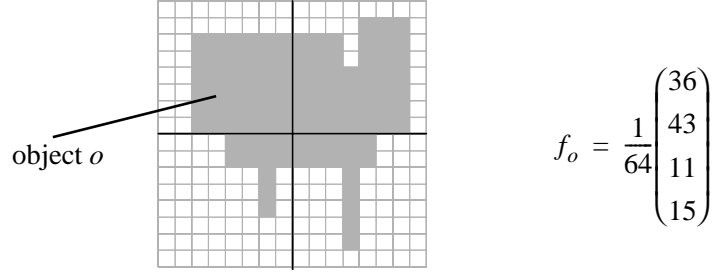


Figure 111: 2D space partitioning with 4 cells.

The feature vector generated by the volume model is depicted on the right hand side.

voxels that represents o where V_i^o are the voxels covered by o in cell i . $\bar{V}^o \subseteq V^o$ denotes the set of voxels at the surface of the objects and $\dot{V}^o \subseteq V^o$ denotes the set of the voxels inside the object, such that $\bar{V}^o \cup \dot{V}^o = V^o$ and $\bar{V}^o \cap \dot{V}^o = \emptyset$ holds. Let r be the number of voxels of the data space in each dimension. In order to ensure a unique assignment of the voxels to a grid cell, we assume that $\frac{r}{p} \in \mathbb{N}$.

After partitioning the data space, we have to determine the spatial features of the objects for each grid cell depending on the chosen model. By scaling the number of partitions, the number of dimensions of the feature vector can be regulated (cf. Figure 111). Obviously, the more partitions we use, the more smaller differences between the objects become decisive. Let f_o be the computed feature vector of an object o . The i -th value of the feature vector of the object o is denoted by $f_o^{(i)}$.

8.2.2 The Volume Model

A simple and established approach to compare two objects is based on the number of the object voxels V_i^o in each cell i of the partitioning. In the following, this model is referred to as the *volume model*. Each cell represents one dimension in the feature vector of the object. The i -th dimension of the feature vector ($1 \leq i \leq p^3$) of object o can be computed by the normalized number of voxels of o lying in cell i . Formally,

$$f_o^{(i)} = \frac{|V_i^o|}{K} \quad \text{where } K = \left(\frac{r}{p}\right)^3$$

Figure 111 illustrates the volume model for the 2D case.

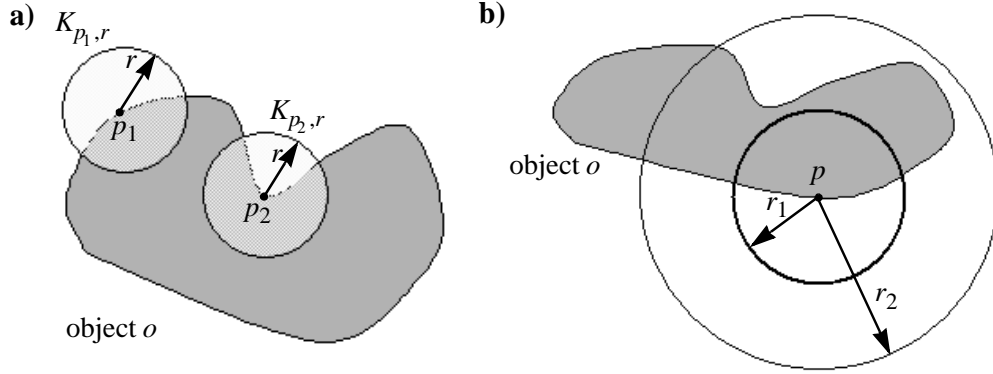


Figure 112: The Solid-Angle model.

a) Different shapes at different surface points, b) Effect of the radius

8.2.3 The Solid-Angle Model

The *solid-angle* method [Con 86] measures the concavity and the convexity of geometric surfaces. It is therefore a good candidate for adequately modelling geometric shapes and has been used in different approaches to spatial similarity modelling. In the following, we describe a model that combines the solid-angle approach with our axis-parallel partitioning.

Let $K_{c,r}$ be a set of voxels that describes a 3D voxelized sphere with central voxel c and radius r . For each surface-voxel \bar{v} of an object o the so called solid-angle value is computed as follows.

The voxels of o which are inside $K_{\bar{v},r}$ are counted and divided by the size of $K_{\bar{v},r}$, i.e. the number of voxels of $K_{\bar{v},r}$. The resulting measure is called the solid-angle value $Sa(\bar{v}, r)$. Formally,

$$Sa(\bar{v}, r) = \frac{|K_{\bar{v},r} \cap V^o|}{|K_{\bar{v},r}|}$$

where $K_{\bar{v},r} \cap V^o = \{w \in K_{\bar{v},r} \mid \exists v \in V^o : w.x = v.x \wedge w.y = v.y \wedge w.z = v.z\}$

A small solid-angle value $Sa(\bar{v}, r)$ indicates that an object is convex at voxel \bar{v} (cf. point p_1 in Figure 112a). Otherwise, a high value of $Sa(\bar{v}, r)$ denotes a concave shape of an object at voxel \bar{v} (cf. point p_2 in Figure 112a). The choice of the radius of the measurement sphere is a crucial parameter. A particular radius could approximate a given object very well (cf. radius r_1 in Figure 112b), whereas another radius might be inept (cf. radius r_2 in Figure 112b).

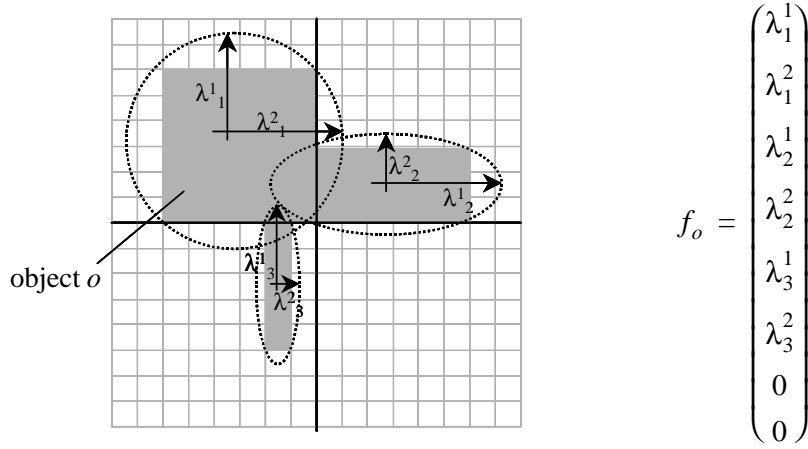


Figure 113: A 2D example for the eigen-value model.

The Solid-Angle values of the cells are transferred into the according histogram bins as described in the following. We distinguish between three different types of cells:

- Cell i contains surface-voxels of object o , i.e. $\bar{V}_i^o \neq \emptyset$. The mean of all Sa -values of the surface-voxels is computed as the feature value of this cell:

$$f_o^{(i)} = \frac{1}{m} \sum_{j=1}^m Sa(\bar{v}_{i_j}, r) \quad \text{where} \quad \bar{V}_i^o = \{\bar{v}_{i_1}, \dots, \bar{v}_{i_m}\}$$

- Cell i contains only inside-voxels of object o , i.e. $\bar{V}_i^o = \emptyset$ and $V_i^o \neq \emptyset$. The feature value of this cell is set to 1 (i.e. $f_o^{(i)} = 1$).
- Cell i contains no voxels of object o , i.e. $V_i^o = \emptyset$. The value of the according bin of the histogram is 0 (i.e. $f_o^{(i)} = 0$).

8.2.4 The Eigen-Value Model

In the following, we introduce a new approach to extract local features which is based on eigen values. The set of voxels of an object can be considered as a set of points in the 3D data space following a particular scattering. The *eigen-value model* uses this scattering of the voxel sets to distinguish the objects by computing the minimum bounding ellipsoid of the voxel set in each cell of the partitioning independently (cf. Figure 113).

A minimum bounding ellipsoid in the 3D space can be described by three vectors. In order to compute these vectors, we consider each voxel v of the object o as a Euclidian vector $\vec{v}^o = (x, y, z)$ in the 3D data space and apply principal axis transfor-

mation. To determine the principal axis of the vectors in cell i , we first compute their centroid \vec{C}^o :

$$\vec{C}^o = \begin{pmatrix} x_C \\ y_C \\ z_C \end{pmatrix} = \frac{1}{|V_i^o|} \begin{pmatrix} \sum_{j=1}^{|V_i^o|} x_j \\ \sum_{j=1}^{|V_i^o|} y_j \\ \sum_{j=1}^{|V_i^o|} z_j \end{pmatrix}$$

After that, for each vector \vec{v}^o in cell i , the following translation is carried out:

$$\vec{v}^o \rightarrow \vec{v}^o - \vec{C}^o$$

Based on these transformed vectors \vec{v}^o , the covariance matrix Cov_i^o for each cell i can be computed as follows:

$$Cov_i^o = \frac{1}{|V_i^o| - 1} \begin{pmatrix} \sum_{j=1}^{|V_i^o|} x_j^2 & \sum_{j=1}^{|V_i^o|} x_j y_j & \sum_{j=1}^{|V_i^o|} x_j z_j \\ \sum_{j=1}^{|V_i^o|} x_j y_j & \sum_{j=1}^{|V_i^o|} y_j^2 & \sum_{j=1}^{|V_i^o|} y_j z_j \\ \sum_{j=1}^{|V_i^o|} x_j z_j & \sum_{j=1}^{|V_i^o|} y_j z_j & \sum_{j=1}^{|V_i^o|} z_j^2 \end{pmatrix}$$

The three eigen vectors \vec{e}_i^j ($j = 1, 2, 3$) of the matrix Cov_i^o correspond to the vectors spanning the minimum bounding ellipsoid of the voxel set V_i^o . The eigen values λ_i^j represent the scaling factors for the eigen vectors (cf. Figure 114). Both eigen values and eigen vectors are determined by the following equation:

$$Cov_i^o \cdot \vec{e}_i^j = \lambda_i^j \vec{e}_i^j$$

The interesting values that are inserted in the bins of the histogram are the eigen values which describe the scattering along the principal axis of the voxel set. These three values can be computed using the characteristic polynomial:

$$\det(Cov_i^o - \lambda_i^j Id) = 0 \quad \text{for } j = 1, 2, 3$$

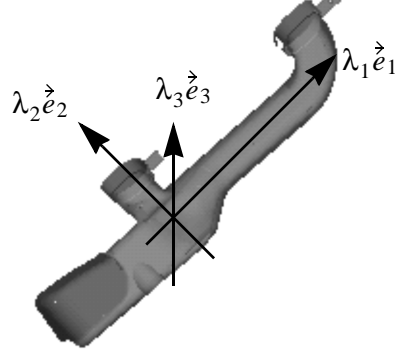


Figure 114: The eigen-value model with the principal axis of a sample object.

Using this equation we obtain three eigen values which are sorted in descending order in the vector $\vec{\lambda}_i$. The highest value represents the variance along the first principal axis, the second value represents the variance along the second principal axis, and the third value represents the variance along the third principal axis.

For each cell i of the partitioning we compute the vector $\vec{\lambda}_i$ of the three eigen values as described right above and register it in the according bins of the histogram:

$$f_o^{(i)} = \vec{\lambda}_i = \begin{pmatrix} \lambda_i^1 \\ \lambda_i^2 \\ \lambda_i^3 \end{pmatrix}$$

Note that for p^3 cells we obtain a feature vector with $3 \times p^3$ dimensions.

8.3 Data Partitioning Similarity Models

In contrast to the last section where we discussed space partitioning similarity models, we turn our attention to data partitioning similarity models in this section. We introduce two different models, namely the *cover sequence model* and the *vector set model*. The cover sequence model (cf. Section 8.3.1) still uses feature vectors for representing objects, whereas the vector set model (cf. Section 8.3.2) uses sets of feature vectors for modelling a 3D voxelized CAD object.

8.3.1 The Cover Sequence Model

The three models described in the foregoing sections are based on a complete partitioning of the data space into disjoint cells. In this section, we adapt a known

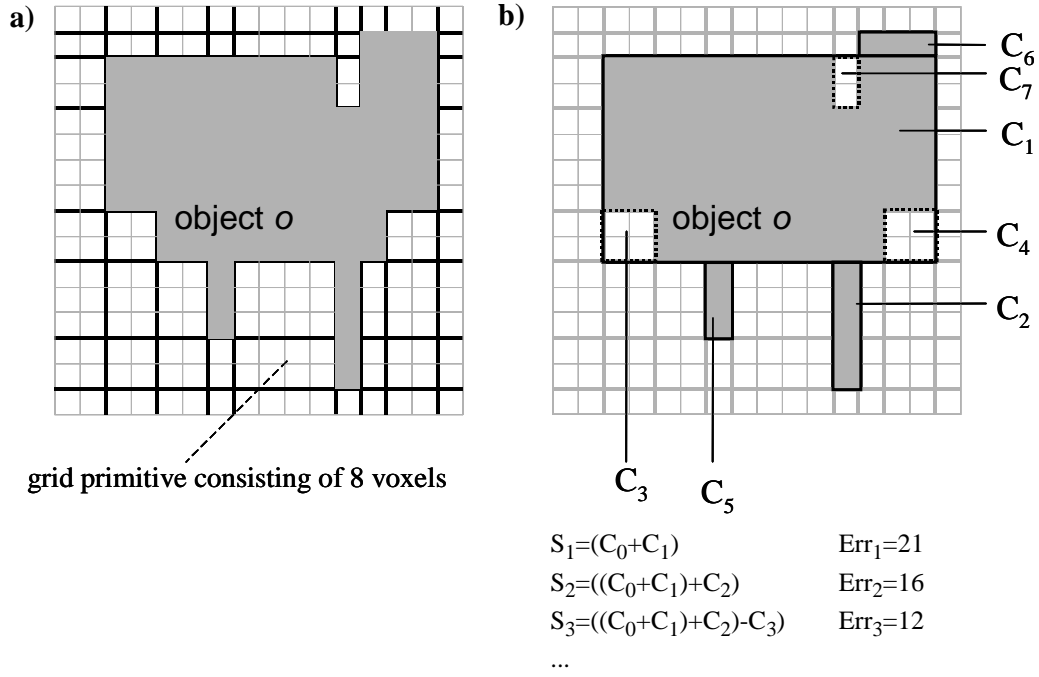


Figure 115: The cover sequence model.

a) Grid primitives, b) Cover sequence

model [Jag 91][JB 91] to voxelized 3D data which is not restricted to this rigid space partitioning but rather uses a more flexible object-oriented partitioning approach. This model is in the following referred to as *cover sequence model*.

General Idea. As depicted in Figure 115a each edge of an object can be extended infinitely in either direction, to obtain a grid of lines. Each rectangle in this grid is called grid primitive, and is located either entirely inside the object, or entirely outside of the object. Furthermore, any pair of adjacent grid primitives must also form a rectangle, respectively a cuboid in the 3D data space. The basic idea of this model is to find large clusters of grid primitives, called *covers*, which approximate the object as good as possible [JB 91]. These covers are organized in a *cover sequence* which provides a sequential description of the object.

Let o be the object being approximated. The quality of a cover sequence S_k of some length $k \in \mathbb{N}$ is measured by the symmetric volume difference Err_k between the object o and the sequence S_k (cf. Figure 115b). Formally, let the covers be drawn from the set C of all possible rectangular covers. Then each unit i of the cover sequence comprises a pair $(C_i \in C, \sigma_i \in \{+, -\})$, where “+” represents set union and “-” represents set difference.

The sequence after k units is:

$$S_k = (((C_0 \sigma_1 C_1) \sigma_2 C_2) \dots \sigma_k C_k),$$

where C_0 is an initial empty cover at the zero point. The symmetric volume difference after k units is:

$$Err_k = |o \text{ XOR } S_k|.$$

Note that there exists some natural number N such that $S_k = o$ and $Err_k = 0$ for all $k \geq N$. At this point an exact description of the object o has been obtained.

If an object o can be described by a sequence S_j with $j < k$ covers and $Err_j = 0$, we assign $((S_j \sigma_{j+1} C_0) \dots \sigma_k C_0)$ to S_k . These dummy covers C_0 do not distort our similarity notion, but guarantee that all feature vectors are of the same dimensionality. Thus we can use common spatial index structures [BKK 96][LJF 94][BBJ+ 00] in order to accelerate similarity queries.

Approximation. Jagadish and Bruckstein [JB 91] suggest two algorithms for the retrieval of a cover sequence S_k : a *branch and bound* algorithm with exponential runtime complexity, and a *greedy* algorithm with polynomial runtime complexity which tries to minimize Err_i in each step $i \leq k$. Throughout our experiments we used this second algorithm.

Let us call a grid primitive *black* if it is in the object, and *white* if it is not. Also, a grid primitive is *in* if it is included in the current description of the object, and *out* if it is not. At the beginning, all grid primitives are either out-black or out-white. At the end, when an exact description of the object is obtained, all grid primitives are either in-black or out-white. The total volume of the grid primitives that are out-black and those that are in-white gives the error in the current description.

To speed up the retrieval of the cover sequence, not every possible cover has to be considered in each step, but only those that are not *dominated* for addition or subtraction. Cover X is said to dominate cover Y for addition iff X contains every out-black grid primitive in Y , Y contains every out-white grid primitive in X , and $X - Y$ is either empty or has at least one out-black grid primitive. Cover X is said to dominate cover Y for subtraction iff X contains every in-white grid primitive in Y , Y contains every in-black grid primitive in X , and $X - Y$ is either empty or has at least one in-white grid primitive.

If cover X dominates cover Y for addition or subtraction, we are guaranteed that the error at the current step is less if cover X is added or subtracted rather than Y , and that the error will continue to be no greater for all future steps.

In [JB 91], it is shown that it is possible, within time proportional to the perimeter of the cover, to determine whether a cover is dominated by another cover. This is faster than calculating the symmetric volume difference for every cover, which takes time proportional to the volume of the cover.

Feature Extraction. In [Jag 91], Jagadish sketches how a 3D cover sequence

$$S_k = (((C_0 \sigma_1 C_1) \sigma_2 C_2) \dots \sigma_k C_k)$$

of an object o , can be transformed into a $6 \times k$ -dimensional feature vector. Thereby, each cover C_{i+1} with $0 \leq i \leq k-1$ is mapped onto 6 values in the feature vector f_o in the following way:

$$\begin{aligned} f_o^{6i+1} &= x\text{-position of } C_{i+1} \\ f_o^{6i+2} &= y\text{-position of } C_{i+1} \\ f_o^{6i+3} &= z\text{-position of } C_{i+1} \\ f_o^{6i+4} &= x\text{-extension of } C_{i+1} \\ f_o^{6i+5} &= y\text{-extension of } C_{i+1} \\ f_o^{6i+6} &= z\text{-extension of } C_{i+1} \end{aligned}$$

The *position* of a cover $C = (x_L, y_L, z_L, x_U, y_U, z_U)$, is given in terms of the mean of the L and U corner points, i.e. the point

$$\left(\frac{x_L + x_U}{2}, \frac{y_L + y_U}{2}, \frac{z_L + z_U}{2} \right)$$

The *extension* of the cover is obtained by taking the difference between the L and U corner points. The ratio between two extension values stemming from different covers, is more meaningful than the difference between these values. By applying the (natural) logarithms to the normalized extension values, the ratio of these extension values is used for computing the distance between two feature vectors. This way common distance functions such as the Euclidian distance can be used to measure the similarity of cover sequences.

$$\sqrt{\dots (\ln A - \ln B)^2 \dots} = \sqrt{\dots \left(\ln \frac{A}{B} \right)^2 \dots} \quad \text{for } A, B \in \mathbb{R}^+$$

To sum up, the *extension* of a cover $C = (x_L, y_L, z_L, x_U, y_U, z_U)$, is given by the triple:

$$(\ln(x_U - x_L), \ln(y_U - y_L), \ln(z_U - z_L))$$

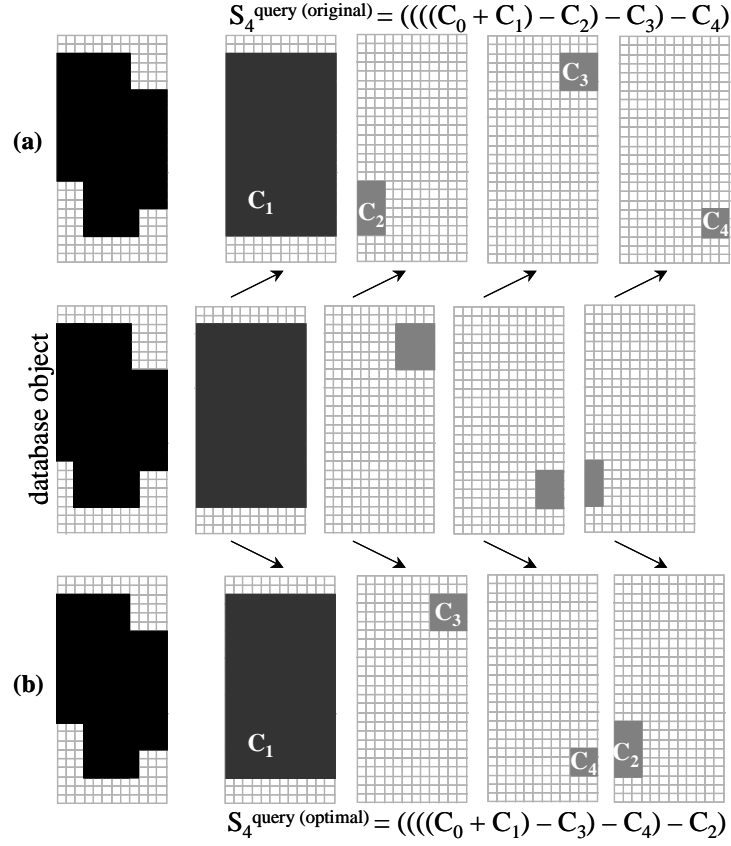


Figure 116: Advantages of free permutations.
 a) Original query object, b) Permuted query object

8.3.2 The Vector Set Model

As described in the foregoing section, a data object is represented as a feature vector which consists of values obtained from a cover sequence approximation. For similarity queries this method yields a major problem. Always comparing the two covers having the same ranking according to the symmetric volume difference, does not make sense in all cases. Two objects can be considered very different, because of the order of their covers, although they are very similar by intuition. The reason for this effect is that the order of the covers does not guarantee that the most similar covers due to size and position will be stored in the same dimensions. Especially for objects generating two or more covers having almost the same volume, the intuitive notion of similarity can be seriously disturbed. Thus, the possibility to match the covers of two compared objects with more degrees of freedom, might offer a better similarity measure. Figure 116 displays a 2-dimensional example of a comparison

between a query object and a very similar database object. The first sequence (cf. Figure 116a) represents the covers of the query object in the order given by the symmetric volume difference. Let us note that the covers C_2 , C_3 and C_4 are not very similar to the corresponding covers of the database object and therefore, the calculated similarity is relatively weak. By rearranging the order of these covers the total distance between the query object and the database object is considerably decreasing, which is displayed in Figure 116b. Thus, the new order preserves the similarity between the objects much better.

To overcome the problem, the author in [Jag 91] proposes to generate several good representations of the query object and then process a query for each of the representations. Afterwards the union of the returned database objects is taken as a result. We can obtain different representations by permuting the order of the found covers and choose the most “promising” orderings to create the query vectors. Though, the method may offer reasonable results in many cases, there is no guarantee that the ordering offering the minimum distance is included within this selection. Thus, the whole similarity measure is dependent on the criteria used to select the most “promising” orderings. Since there is no well defined selection criterion known so far, the solution does not necessarily offer a precisely defined similarity measure.

Minimum Euclidian Distance Under Permutation. Another solution for the problem is to consider all possible permutations. Since the distance between two objects can now be considered as the minimum distance over all possible orderings, the distance is defined precisely this way.

Definition 28 (Minimum Euclidian Distance under Permutation).

Let $exch: IN \times IN \times IR^{(k \cdot d)} \rightarrow IR^{(k \cdot d)}$ be a function, where $exch(i, j, \vec{x})$ exchanges the d successive components beginning with dimension $i \cdot d + 1$ ($0 \leq i \leq k - 1$) with the d successive components beginning with dimension $j \cdot d + 1$ ($0 \leq j \leq k - 1$) of a vector $\vec{x} \in IR^{(k \cdot d)}$. Let $Exch: IR^{(k \cdot d)} \rightarrow 2^{IR^{(k \cdot d)}}$ be the function, that generates the set of all vectors that can be generated by applying $exch(i, j, \vec{x})$ arbitrary many times to a vector \vec{x} using any combination for i and j . Then the *minimum Euclidian distance under permutation* $d_{\Pi-euclid}: IR^{(k \cdot d)} \times IR^{(k \cdot d)} \rightarrow IR$ is defined as follows:

$$d_{\Pi-euclid}(\vec{x}, \vec{y}) = \min_{\vec{z} \in Exch(\vec{y})} \{d_{euclid}(\vec{x}, \vec{z})\}$$

With a growing number of describing covers k , the processing time of considering all possible permutations increases exponentially, since there are $k!$ many permutations. With computation cost rising this rapidly, it is obvious that the description length k has to be kept low, which is not acceptable for all applications.

To guarantee that the permutation with the minimal distance is used, our approach does not work with one single feature vector, but with a set of feature vectors in lower dimensions. By treating the data objects as sets of d -dimensional feature vectors with a maximum cardinality of k , we introduce a new model for representing data objects in similarity search systems, the so called *vector set model*. In this new approach, an object is represented by a vector set $X \subset IR^d$ with $|X| \leq k$.

Reasons for the Use of Vector Set Representation. The representation of extracted features as a set of vectors is a generalization of the use of just one large feature vector. It is always possible to restrict the model to a feature space, in which a data object will be completely represented by just one feature vector. But in our application the possibilities of vector set representation allow us to model the dependencies between the extracted features more precisely. As the development of conventional database systems in the recent two decades has shown, the use of more sophisticated ways to model data can enhance both the effectiveness and efficiency for applications using large amounts of data. In our application the vector set representation is able to avoid the problems that occur by storing a set of covers according to a strict order. Therefore, it is possible to compare two objects more intuitively, causing a relatively small rise of calculation cost compared to the distance calculation in the one-vector model. Another advantage of our new approach is the better storage utilization. It is not necessary to force objects into a common size, if they are represented by sets of different cardinality. For our current application there is no need for dummy covers to fill up the feature vectors. If the quality of the approximation is optimal with less than the maximum number of covers, only this smaller number of vectors has to be stored and loaded. In the case of a one-vector representation avoiding dummies is not possible without further modifications of the access structures used. Furthermore, we are able to distinguish between the distance measure used on the feature vectors of a set and the way we combine the resulting distances between the single feature vectors. For example, this possibility might be useful when defining partial similarity, where it is only necessary to compare the closest $i < k$ vectors of a set.

In the following sections, we will discuss the concept of vector set representation in detail, with the goal of providing a high quality distance measure for vector-set-represented data and an algorithm for its efficient computation.

Distance Measures on Sets of Feature Vectors. There already exist several different distance measures for sets of objects. In [EM 97] the authors survey the following distance functions, which are computable in polynomial time: the *Hausdorff distance*, the *sum of minimal distances*, the *(fair-)surjection distance* and the *link distance*. The Hausdorff distance is a metric, but does not seem to be suitable as a similarity measure, because it relies too much on the extreme positions of the elements of both sets. The last three distance measures are suitable for modelling similarity, but are not metric. This circumstance makes them unattractive, since there are only limited possibilities for processing similarity queries efficiently when using a non-metric distance function. In [EM 97] the authors also introduce a method for expanding the distance measures into metrics, but as a side effect the complexity of distance calculation becomes exponential. Furthermore, the possibility to match several elements in one set to just one element in the compared set, is questionable when comparing sets of covers like in our application.

The Minimal Matching Distance. A distance measure on vector sets that demonstrates to be suitable for defining similarity in our application is based on the *minimal weight perfect matching* of sets. This well known graph problem can be applied here. Let us first introduce some notations.

Definition 29 (Weighted Complete Bipartite Graph).

A Graph $G = (V, E)$ consists of a (finite) set of vertices V and a set of edges $E \subseteq V \times V$. A weighted graph is a graph $G = (V, E)$ together with a weight function $w: E \rightarrow \mathbb{R}$. A *bipartite graph* is a graph $G = (X \cup Y, E)$ with $X \cap Y = \emptyset$ and $E \subseteq X \times Y$. A bipartite graph $G = (X \cup Y, E)$ is called *complete* if $E = X \times Y$.

Definition 30 (Perfect Matching).

Given a bipartite graph $G = (X \cup Y, E)$ a *matching* of X to Y is a set of edges $M \subseteq E$ such that no two edges in M share an endpoint, i.e.

$$\forall (x_1, y_1), (x_2, y_2) \in M: x_1 = x_2 \Leftrightarrow y_1 = y_2$$

A matching M of X to Y is *maximal* if there is no matching M' of X to Y such that $|M| < |M'|$. A maximal matching M of X to Y is called a *complete* matching if

$|M| = \min\{|X|, |Y|\}$. In the case $|X| = |Y|$ a complete matching is also called a *perfect matching*.

Definition 31 (Minimum Weight Perfect Matching).

Given a weighted bipartite graph $G = (X \cup Y, E)$ together with a weight function $w: E \rightarrow \mathbb{R}$. We call a perfect matching M , a *minimum weight perfect matching*, iff for any other perfect matching M' , the following inequality holds:

$$\sum_{(x,y) \in M} w(x,y) \leq \sum_{(x,y) \in M'} w(x,y)$$

In our application we build a complete bipartite graph $G = (X \cup Y, E)$ between two vector sets $X, Y \subset \mathbb{R}^d$ with $|X|, |Y| \leq k$. We set $X' = X \times \{1\}$ and $Y' = Y \times \{2\}$ in order to fulfill the property $X' \cap Y' = \emptyset$. The weight of each edge $((\vec{x}, 1), (\vec{y}, 2)) \in X' \times Y'$ in this graph G is defined by the distance $dist(\vec{x}, \vec{y})$ between the vectors $\vec{x} \in X$ and $\vec{y} \in Y$. For example the Euclidian distance can be used here. A perfect matching is a subset $M \subseteq X' \times Y'$ that connects each $\vec{x} \in X$ to exactly one $\vec{y} \in Y$ and vice versa. A minimal weight perfect matching is a matching with maximum cardinality and a minimum sum of weights of its edges. Since a perfect matching can only be found for sets of equal cardinality, it is necessary to introduce weights for unmatched nodes when defining a distance measure.

Definition 32 (Enumeration of a Set).

Let S be any finite set of arbitrary elements. Then π is a mapping that assigns $s \in S$ a unique number $i \in \{1, \dots, |S|\}$. This is written as $\pi(S) = (s_1, \dots, s_{|S|})$. The set of all possible enumerations of S is named $\Pi(S)$.

Definition 33 (Minimal Matching Distance).

Let $V \subset \mathbb{R}^d$ and let $dist: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ be a distance function between two d -dimensional feature vectors. Let $X = \{\vec{x}_1, \dots, \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \dots, \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Furthermore, let $w: V \rightarrow \mathbb{R}$ be a weight function for the unmatched elements. Then the *minimal matching distance* $d_{mm}^{dist, w}: 2^V \times 2^V \rightarrow \mathbb{R}$ is defined as follows:

$$d_{mm}^{dist, w}(X, Y) = \min_{\pi \in \Pi(Y)} \left(\sum_{i=1}^{|X|} dist(\vec{x}_i, \vec{y}_{\pi(i)}) + \sum_{i=|X|+1}^{|Y|} w(\vec{y}_{\pi(i)}) \right)$$

The weight function $w: V \rightarrow IR$ provides the penalty given to every unassigned element of the set having larger cardinality. Let us note that *minimal matching distance* is a specialization of the *netflow distance* which is introduced in [RB 00]. The authors in [RB 00] show that the netflow distance is a metric and that it is computable in polynomial time. Therefore, we derive the following lemma without further proof.

Lemma 6. Let $V \subset IR^d$. The minimal matching distance $d_{mm}^{dist, w}: 2^V \times 2^V \rightarrow IR$ is a metric if the underlying distance function $dist: IR^d \times IR^d \rightarrow IR$ is a metric and the weight function $w: V \rightarrow IR$ meets the following conditions for all $\vec{x}, \vec{y} \in V$:

- $w(\vec{x}) > 0$
- $w(\vec{x}) + w(\vec{y}) \geq dist(\vec{x}, \vec{y})$

Definition 34 (Dummy Vectors).

Let $V \subset IR^d$ be a set of d -dimensional vectors. Let $\|\vec{x} - \vec{y}\|_2$ be the Euclidean distance between $\vec{x}, \vec{y} \in IR^d$. Furthermore, let $\vec{w} \in IR^d \setminus V$ be a “dummy” vector. Then $w_{\vec{w}}: V \rightarrow IR: w_{\vec{w}}(x) = \|\vec{x} - \vec{w}\|_2$ denotes a set of weight functions based on dummy vectors.

A good choice of \vec{w} for our application is $\vec{0}$, since it has the shortest average distance within the position and has no volume. Since there are no covers having no volume in any data object, the conditions for the metric character of the *minimum matching distance* are satisfied (cf. Lemma 6).

The *minimum Euclidian distance under permutation* can be derived from the *minimal matching distance*. By selecting the squared Euclidian distance as distance measure on V and taking the squared Euclidian norm as weight function, the distance value calculated by the minimal matching distance is the same as the squared value of the minimum Euclidian distance under permutation. This follows exactly from the definitions of both distance measures. Let us note that it is necessary to extract the square root from this distance value to preserve the metric character.

The Kuhn-Munkres Algorithm. Though it was shown in [RB 00] that the netflow distance can be calculated in polynomial time, it is not obvious how to achieve it. Since we are only interested in the minimal matching distance, it is sufficient to calculate a minimal weight perfect matching. Therefore, we apply the method proposed by Kuhn [Kuh 55] and Munkres [Mun 57].

For the remainder of this section we assume that we are given a weighted complete bipartite graph $G = (X \cup Y, E)$ with the weight function $w: X \times Y \rightarrow IR$. As we can use dummy vectors (cf. Definition 34), we assume w.l.o.g. that X and Y have equal cardinality k .

The goal of the Kuhn-Munkres algorithm is to find a *maximal* weight matching in G . To obtain a *minimal* weight matching the following trick can be used. We replace the weight function w by the function w' with $w'(x, y) = -w(x, y)$ and apply the algorithm to G and w' .

Definition 35 (M-Alternating Path).

Given a matching M of X to Y an edge $(x, y) \in X \times Y$ is called *matched* if $(x, y) \in M$, *unmatched* otherwise. An M -alternating path is a alternating sequence of unmatched and matched edges with free end nodes.

If the edges of an M -alternating path are flipped in M , i.e. *matched* edges become *unmatched* and vice versa, a matching M' is obtained with $|M'| = |M| + 1$.

Definition 36 (Feasible Vertex Labeling).

A *feasible vertex labeling* in G is a function $l: X \cup Y \rightarrow IR$ such that

$$\forall x \in X, \forall y \in Y: l(x) + l(y) \geq w(x, y)$$

It is always possible to find a feasible vertex labeling. One way to do this is to set $l(y) = 0$ for all $y \in Y$ and for each $x \in X$ take the maximum value in the corresponding row of edge weights, i.e.

$$\begin{aligned} l(x) &= \max_{y \in Y} \{w(x, y)\} \text{ for } x \in X \\ l(y) &= 0 \text{ for } y \in Y \end{aligned}$$

An example of a feasible vertex labeling is depicted in Figure 117. If l is a feasible labeling, we denote by G_l the subgraph of G which contains those edges where $l(x) + l(y) = w(x, y)$, together with the endpoints of these edges. This graph G_l is called the *equality subgraph* for l . In Figure 117 the edges of G_l are shaded. For $S \subseteq X$ we denote by $J(G_l, S)$ the set $\{y \in Y | x \in S \wedge (x, y) \in G_l\}$ of all vertices in Y which are adjacent to the vertices in S .

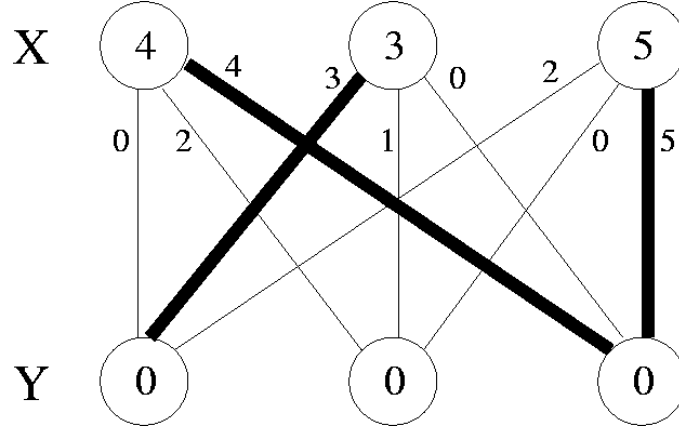


Figure 117: An example of a feasible vertex labeling and an equality subgraph.

Lemma 7. If l is a feasible vertex labeling for G , and M is a perfect matching of X to Y with $M \subseteq G_l$, then M is a maximal weight perfect matching of X to Y .

Proof. We must show that no other complete matching can have weight greater than M . Let any complete matching M' of X to Y be given. Then

$$\begin{aligned}
 w(M') &= \sum_{(x,y) \in M'} w(x,y) \\
 &\leq \sum_{(x,y) \in M'} (l(x) + l(y)) \quad (\text{feasibility of } l) \\
 &= \sum_{(x,y) \in M} (l(x) + l(y)) \quad (\text{all } l(x), l(y) \text{ summed in either matching}) \\
 &= \sum_{(x,y) \in M} w(x,y) \quad (\text{since } M \subseteq G_l) \\
 &= w(M) \quad \blacksquare
 \end{aligned}$$

Thus the problem of finding a maximal weight perfect matching is reduced to the problem of finding a feasible vertex labeling whose equality subgraph contains a perfect matching of X to Y . Using this result, the Kuhn-Munkres algorithm (also known as the *Hungarian method*) works like this:

1. Start with an arbitrary feasible vertex labeling l , determine G_l , and choose an arbitrary matching M in G_l .
2. If M is complete for G , then M is optimal. Stop. Otherwise, there is some unmatched $x \in X$. Set $S = \{x\}$ and $T = \emptyset$.
3. If $J(G_l, S) \neq T$, go to step 4. Otherwise, $J(G_l, S) = T$. Find

$$\alpha_l = \min_{x \in S, y \in Y \setminus T} \{l(x) + l(y) - w(x, y)\}$$

and construct a new labeling l' by

$$l'(v) = \begin{cases} l(v) - \alpha_l & \text{for } v \in S \\ l(v) + \alpha_l & \text{for } v \in T \\ l(v) & \text{otherwise} \end{cases}$$

Note, that $\alpha_l > 0$ and $J(G'_l, S) \neq T$. Replace l by l' and G_l by G'_l .

4. Choose a vertex y in $J(G'_l, S)$, not in T . If y is matched in M , say with $z \in X$, replace S by $S \cup \{z\}$ and T by $T \cup \{y\}$, and go to step 3. Otherwise, there will be an M -alternating path from x to y , and we may use this path to find a larger matching M' in G_l . Replace M by M' and go to step 2.

When necessary, edges are added to the equality subgraph G_l by constructing a new feasible labeling in step 3. This way a perfect matching will eventually be found. Since there are at most k phases in which an M -alternating path is constructed and each phase can be computed in $O(k^2)$ the time complexity of a distance calculation using the method of Kuhn and Munkres is $O(k^3)$ in the worst case. Let us note that for larger numbers of k this is far better than the previously mentioned method on $k!$ many permutations.

Chapter 9

Effectiveness of Similarity Models

In this chapter, we introduce density-based hierarchical clustering as a new and effective way to analyse and compare similarity models (cf. Section 9.1). We show that this new approach is much more suitable for the evaluation of similarity models than the commonly used k -nn queries which are subjective and error-prone. Based on clustering, we investigate the quality of the space partitioning similarity models and the data partitioning similarity models (cf. Section 9.2). Among the space partitioning models, the eigen-value model is the most suitable model. The quality of this space partitioning model is comparable to the quality of the cover sequence model which is clearly outperformed by the vector set model. Generally, the data partitioning similarity models yield more meaningful results than the space partitioning ones. Especially the combination of the data partitioning cover sequence model and the new paradigm of using sets of feature vectors for representing objects is a very powerful approach for the detection of similar CAD parts.

9.1 A New Approach for Evaluating Similarity Models

In general, similarity models can be evaluated by computing k -nearest neighbor queries (cf. Section 9.1.1). A drawback of this evaluation approach is that the quality measure of the similarity model depends on the results of few similarity queries and, therefore, on the choice of the query objects. A model may perfectly reflect the intuitive similarity according to the chosen query objects and would be evaluated as “good” although it produces disastrous results for other query objects. As a consequence, the evaluation of similarity models with sample k -nn queries is subjective and error-prone.

A better way to evaluate and compare several similarity models is to apply a clustering algorithm (cf. Section 9.1.2). Clustering groups a set of objects into classes where objects within one class are similar and objects of different classes are dissimilar to each other. The result can be used to evaluate which model is best suited for which kind of objects.

9.1.1 k -nn Queries

The notion of k -nn queries has been defined in Section 7.1. In Figure 118, the results of a 5-nn query for two different similarity models A and B are presented, whereby model A is an inept model and model B a suitable similarity model for voxelized CAD data. We achieved satisfying results for each model depending on the query object. For a tire, for example, model A performs very well, yielding objects that are intuitively very similar to the query object (cf. Figure 118a). Comparably good results are also produced by model B for a part of the fender (cf. Figure 118b).

Although both models deliver rather accurate results for the chosen query objects, we also see in Figure 118 that these results are delusive. Figure 118c shows a nearest neighbor query for an object where there exist several similar parts to this object within our database. Model A does not recognize this. Furthermore, there might be objects for which no similarity model can yield any intuitively similar parts (cf. Figure 118d). Obviously, we should not discard a similarity model if the chosen query object belongs to noise. This confirms the assumption that the method of evaluating similarity models using several k -nn queries is subjective and error-prone, due to its dependency on the choice of the query objects.

In the next section, we introduce the density-based hierarchical clustering algorithm OPTICS in order to overcome the above described difficulties.

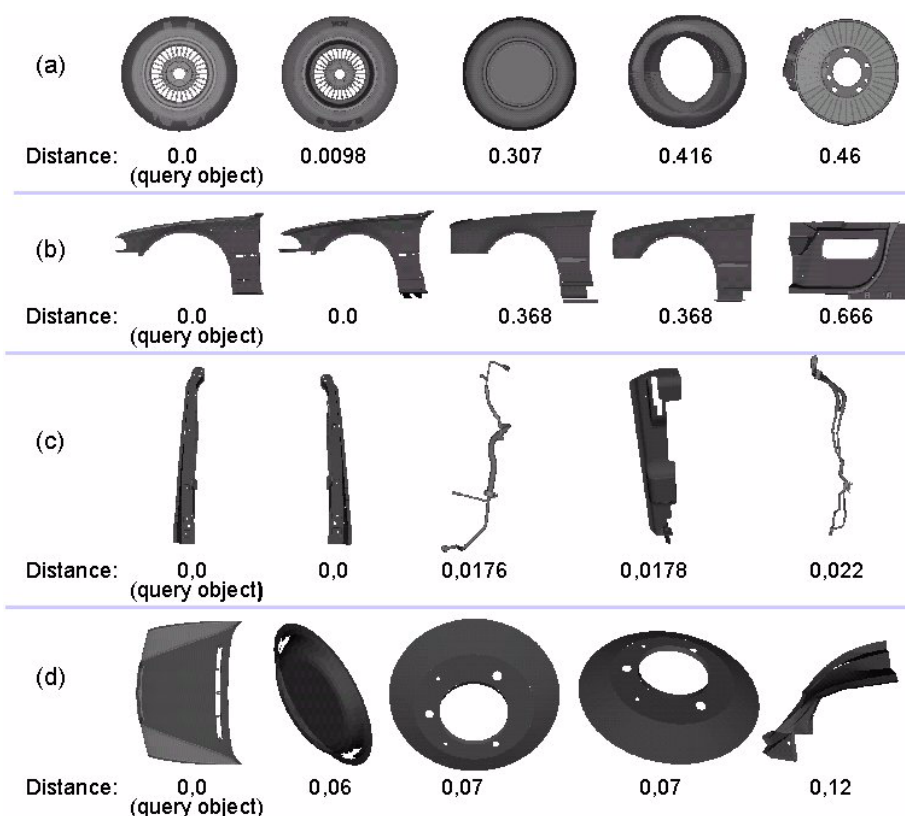


Figure 118: Results of 5-nn queries for a “good” and “bad” similarity model.
a) “good” query object - “bad” model, **b)** “good” query object - “good” model,
c) “good” query object - “bad” model, and **d)** “bad” query object - “good” model

9.1.2 OPTICS: A Density-Based Hierarchical Clustering Algorithm

A more objective way to evaluate and compare several similarity models is to apply a clustering algorithm. Clustering groups a set of objects into classes where objects within one class are similar and objects of different classes are dissimilar to each other. The result can be used to evaluate which model is best suited for which kind of objects. Furthermore, if we use clustering, the evaluation of the models is based on the whole data set and not only on few sample objects.

For the evaluation of similarity models we suggest to use the density-based hierarchical clustering algorithm OPTICS [ABKS 99]. The algorithm is similar to hierarchical Single-Link clustering methods [JD 88] and is an extension of the clustering algorithm DBSCAN [EKSX 96].

We choose OPTICS due to the following reasons. First, OPTICS is - in contrast to most other algorithms - relatively insensitive to its two input parameters ϵ and $MinPts$ [ABKS 99]. Second, OPTICS is a hierarchical clustering method which yields more information about the cluster structure than a method that computes a flat partitioning of the data (e.g. k -means [McQ 67]).

In this section, we provide a detailed description of the clustering algorithm OPTICS, which is based on the idea of density-based clustering.

Density-Based Clustering. The key idea of density-based clustering is that for each object q of a cluster the neighborhood of a given radius ϵ , i.e. the ϵ -neighborhood $N_\epsilon(q)$, has to contain at least a minimum number of objects $MinPts$, i.e. the cardinality of the neighborhood has to exceed a threshold. It is assumed that there is a metric distance function on the objects in the database (e.g. one of the L_p -norms for a database of feature vectors). In the following, we will formally introduce density-based clustering.

Definition 37 (*Directly Density-Reachable*).

Object p is *directly density-reachable* from object q with regard to ϵ and $MinPts$ in a set of objects O if

$$\begin{aligned} (1) \quad & p \in N_\epsilon(q) \\ (2) \quad & |N_\epsilon(q)| \geq MinPts \end{aligned}$$

The condition $|N_\epsilon(q)| \geq MinPts$ is called the *core object condition*. If this condition holds for an object q , then we call q a *core object*. Only from core objects other objects can be directly density-reachable.

Definition 38 (*Density-Reachable*).

An object p is *density-reachable* from an object q with regard to ϵ and $MinPts$ in a set of objects O , if there is a chain of objects p_1, \dots, p_n with $p_1 = q$, $p_n = p$, such that $p_i \in O$ and p_{i+1} is directly density-reachable from p_i with regard to ϵ and $MinPts$.

Density-reachability is the transitive closure of direct density-reachability. This relation is not symmetric in general. Only core objects can be mutually density-reachable.

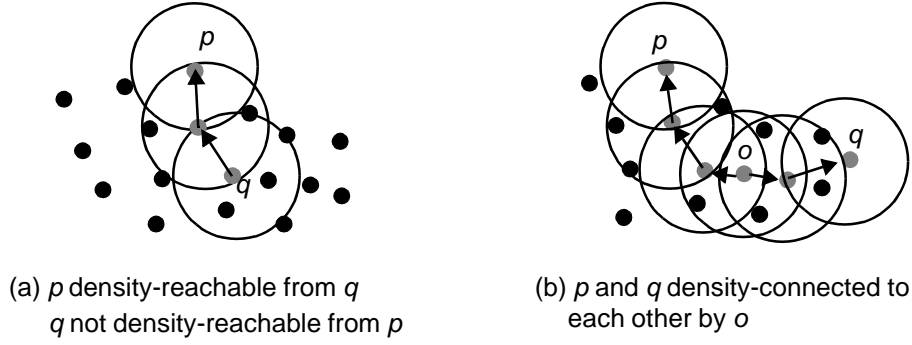


Figure 119: Density-reachability and density-connectivity.

Definition 39 (*Density-Connected*).

Object p is *density-connected* to object q with regard to ϵ and $MinPts$ in the set of objects O , if there is an object $o \in O$ such that both p and q are density-reachable from o with regard to ϵ and $MinPts$ in O .

Density-connectivity is a symmetric relation. Figure 119 illustrates the definitions on a sample database of 2-dimensional points from a vector space. Note that the above definitions only require a distance measure and are also applicable to data from a metric space.

A density-based cluster is now defined as a set of density-connected objects which is maximal with regard to density-reachability and the noise is the set of objects not contained in any cluster.

Definition 40 (*Cluster and Noise*).

Let O be a set of objects. A *cluster* C with regard to ϵ and $MinPts$ in O is a non-empty subset of O satisfying the following conditions:

1. *Maximality*: for all $p, q \in O$: if $q \in C$ and p is density-reachable from q with regard to ϵ and $MinPts$, then also $p \in C$.
2. *Connectivity*: for all $p, q \in C$: p is density-connected to q with regard to ϵ and $MinPts$ in O .

Every object not contained in any cluster is *noise*.

Note that a cluster contains not only core objects but also objects that do not satisfy the core object condition. These objects - called *border objects* of the cluster - are, however, directly density-reachable from at least one core object of the cluster (in contrast to noise objects).

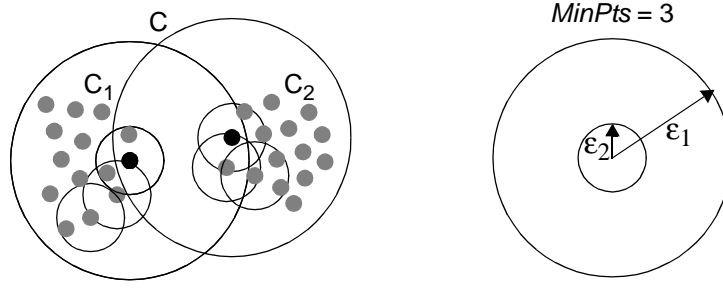


Figure 120: Nested density-based clusters.

DBSCAN. The algorithm DBSCAN [EKSX 96], which discovers the clusters and the noise in a database according to the above definitions, is based on the fact that a cluster is equivalent to the set of all objects in O which are density-reachable from an arbitrary core object in the cluster (cf. lemma 1 and 2 in [EKSX 96]). The retrieval of density-reachable objects is performed by iteratively collecting directly density-reachable objects. DBSCAN checks the ϵ -neighborhood of each point in the database. If the ϵ -neighborhood $N_\epsilon(q)$ of a point p has more than $MinPts$ points, a new cluster C containing the objects in $N_\epsilon(q)$ is created. Then, the ϵ -neighborhood of all points q in C which have not yet been processed is checked. If $N_\epsilon(q)$ contains more than $MinPts$ points, the neighbors of q which are not already contained in C are added to the cluster and their ϵ -neighborhood is checked in the next step. This procedure is repeated until no new point can be added to the current cluster C .

OPTICS. OPTICS emerges from the algorithm DBSCAN which computes a flat partitioning of the data. The original output of OPTICS is an ordering of the objects, a so called cluster ordering. To introduce the notion of a density-based cluster-ordering, we first make the following observation: for a constant $MinPts$ -value, density-based clusters with respect to a higher density (i.e. a lower value for ϵ) are completely contained in density-connected sets with respect to a lower density (i.e. a higher value for ϵ). This fact is illustrated in Figure 120, where C_1 and C_2 are density-based clusters with respect to $\epsilon_2 < \epsilon_1$ and C is a density-based cluster with respect to ϵ_1 completely containing the sets C_1 and C_2 .

Consequently, the DBSCAN algorithm could be extended such that several distance parameters are processed at the same time, i.e. the density-based clusters with respect to different densities are constructed simultaneously.

To produce a consistent result, however, we would have to obey a specific order in which objects are processed when expanding a cluster. We always have to select an object which is density-reachable with respect to the lowest ϵ value to guarantee that clusters with respect to higher density (i.e. smaller ϵ values) are finished first.

OPTICS works in principle like such an extended DBSCAN algorithm for an infinite number of distance parameters ϵ_i which are smaller than a *generating distance* ϵ (i.e. $0 < \epsilon_i \leq \epsilon$). The only difference is that we do not assign cluster memberships. Instead, we store the order in which the objects are processed and the information which would be used by an extended DBSCAN algorithm to assign cluster memberships (if this were at all possible for an infinite number of parameters). This information consists of only two values for each object: the core-distance and a reachability-distance, introduced in the following definitions.

Definition 41 (*Core-Distance*).

Let o be an object from a database DB , let ϵ be a distance value, let $N_\epsilon(o)$ be the ϵ -neighborhood of o , let $MinPts$ be a natural number and let $MinPts\text{-}distance(o)$ be the distance from o to its $MinPts$ -th neighbor. Then, the *core-distance* of o is defined as:

$$core\text{-}distance_{\epsilon, MinPts}(o) = \begin{cases} \infty & , \text{ if } |N_\epsilon(o)| < MinPts \\ MinPts\text{-}distance(o) & , \text{ otherwise} \end{cases}$$

Definition 42 (*Reachability-Distance*).

Let $o, p \in DB$, let $N_\epsilon(o)$ be the ϵ -neighborhood of o , and let $MinPts$ be a natural number. Then, the *reachability-distance* of p w.r.t. o is defined as:

$$reachability\text{-}distance_{\epsilon, MinPts}(p, o) = \begin{cases} \max(core\text{-}distance_{\epsilon, MinPts}(o), distance(p, o)) & , \text{ if } p \in N_\epsilon(o) \\ \infty & , \text{ otherwise} \end{cases}$$

If it is clear from the context, we omit the parameters ϵ and $MinPts$ which leads to the following simplified abbreviations: $core\text{-}dist(o) = core\text{-}distance_{\epsilon, MinPts}(o)$ and $reach\text{-}dist(p, o) = reachability\text{-}distance_{\epsilon, MinPts}(p, o)$.

Intuitively, the reachability-distance of an object p with respect to another object o is the smallest distance such that p is directly density-reachable from o if o is a core object. In this case, the reachability-distance cannot be smaller than the core-distance of o because for smaller distances no object is directly density-reachable from o .

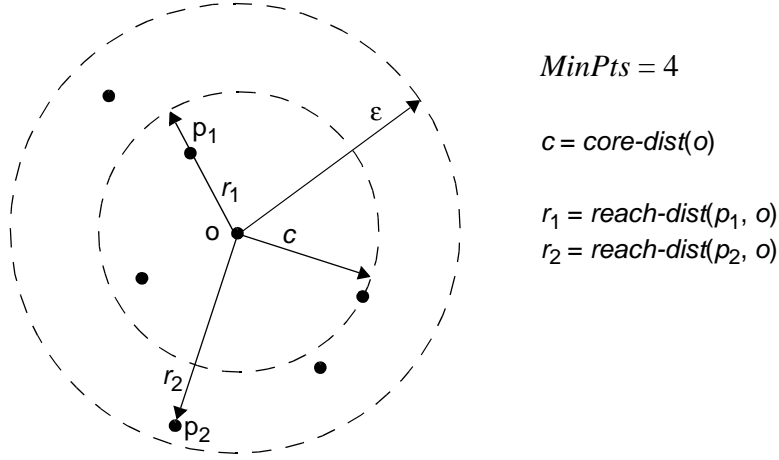


Figure 121: Illustration of core-distance and reachability-distance.

Otherwise, if o is not a core object, even at the generating distance ϵ , the reachability-distance of p with respect to o is undefined. The reachability-distance of an object p depends on the core object with respect to which it is calculated. Figure 121 illustrates the notions of core-distance and reachability-distance.

In contrast to DBSCAN, OPTICS does not assign cluster memberships but computes an *ordering* in which the objects are processed and additionally generates the information which would be used by an extended DBSCAN algorithm to assign cluster memberships. This information consists of only two values for each object, the *core-distance* and the *reachability-distance*.

Definition 43 (*Cluster Ordering*).

Let $MinPts \in \mathbb{N}$, $\epsilon \in \mathbb{R}$, and CO be a totally ordered permutation of objects. To each object o additional attributes $o.P$, $o.C$ and $o.R$ are assigned: $o.P \in \{1, \dots, |CO|\}$, $o.C = core-dist(o)$, and $o.R = \min\{reach-dist(o, o') \mid \forall o' \in CO: o'.P < o.P\}$ where $\min \emptyset = \infty$. We call CO a cluster ordering w.r.t. ϵ and $MinPts$ if the following condition holds:

$$\forall x, y \in CO: x.P < y.P \Rightarrow \neg \exists o \in CO: (o.P < x.P \wedge reach-dist(y, o) \leq x.R)$$

The attribute $o.P$ indicates the position of our object o in the cluster ordering CO , the attribute $o.R$ is the reachability-distance assigned to object o during the generation of CO , and the attribute $o.C$ indicates the core-distance of this object in CO . We call $o.R$ the *reachability* of object o throughout the remainder of this thesis. Note that $o.R$ is only well-defined in the context of a cluster ordering.

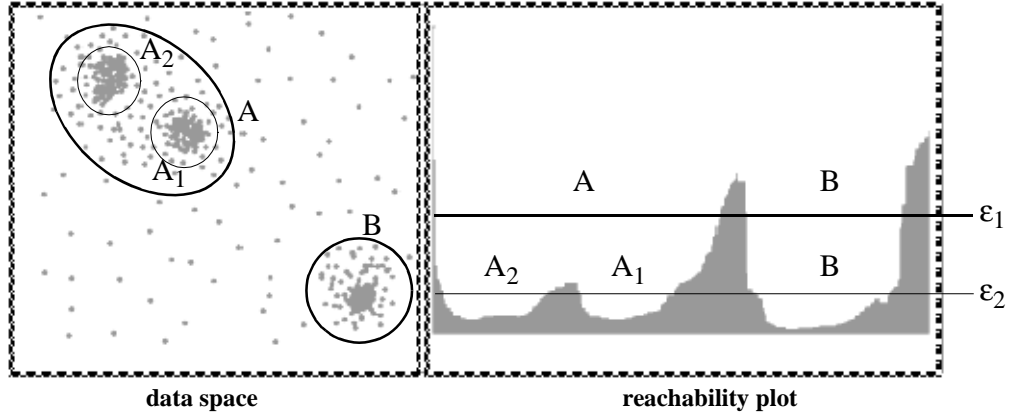


Figure 122: Reachability plot (right) computed by OPTICS for a 2D data set (left).

Intuitively, the condition of Definition 43 states that the order is built on selecting at each position i in CO that object o having the minimum reachability to any object before i .

Reachability Plots. The output of OPTICS is a linear ordering of the database objects minimizing a binary relation called *reachability* which is in most cases equal to the minimum distance of each database object to one of its predecessors in the ordering. This idea is similar to the Single-Link method but instead of a dendrogram, the resulting reachability-plot is much easier to analyse. The reachability values can be plotted for each object of the cluster-ordering computed by OPTICS. Valleys in this plot indicate clusters: objects having a small reachability value are more similar to their predecessor objects than objects having a higher reachability value.

The reachability plot generated by OPTICS can be cut at any level ϵ parallel to the abscissa. It represents the density-based clusters according to the density threshold ϵ : A consecutive subsequence of objects having a smaller reachability value than ϵ belong to the same cluster. An example is presented in Figure 122. For a cut at the level ϵ_1 we retrieve two clusters denoted as A and B . Compared to this clustering, a cut at level ϵ_2 would yield three clusters. The cluster A is split into two smaller clusters denoted as A_1 and A_2 and cluster B has decreased its size.

Note that the visualization of the cluster-order is independent from the dimension of the data set. For example, if the objects of a high-dimensional data set are distributed similar to the distribution of the 2-dimensional data set depicted in Figure 122, i.e. there are three “Gaussian bumps” in the data set, the reachability plot would look very similar to the one presented in Figure 122. Furthermore, density-based cluster-

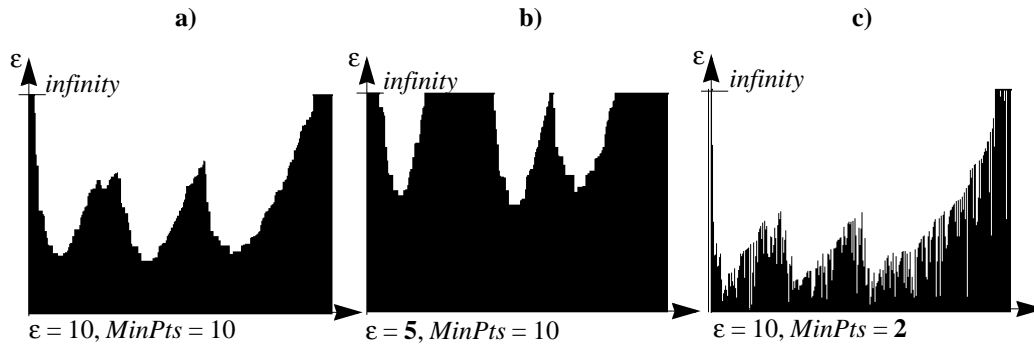


Figure 123: Effects of parameter settings on the cluster ordering of OPTICS.

a) Appropriate parameters, b) ϵ -Parameter too small, c) $MinPts$ -Parameter too small

ing is not only restricted to feature spaces but can be applied to all kinds of metric spaces, e.g. to data spaces where objects are represented by vector sets.

Parameter Settings. A further advantage of this cluster ordering compared to other clustering methods is that the reachability plot is rather insensitive to the input parameters of the method, i.e. the generating distance ϵ and the value for $MinPts$. Roughly speaking, the values just have to be “large” enough to yield a good result. The concrete values are not crucial because there is a broad range of possible values for which we always can see the clustering structure of a data set when looking at the corresponding reachability plot. Figure 123 shows the effects of different parameter settings on the reachability plot for the same data set. The first plot was generated with appropriate parameter settings, whereas for the second plot a smaller generating distance ϵ was used. For the third plot $MinPts$ was set to the smallest possible value. Although, these plots look very different to each other, the overall clustering structure of the data set can be recognized in all plots.

The ϵ -Parameter. Usually, for evaluation purposes, a good value for ϵ would yield as many clusters as possible. The generating distance ϵ influences the number of clustering-levels which can be seen in the reachability-plot. The smaller we choose the value of ϵ , the more objects may have an *infinite* reachability-distance. Therefore, we may not see clusters of lower density, i.e. clusters where the core objects are core objects only for distances larger than ϵ . The optimal value for ϵ is the smallest value so that a density-based clustering of the database with respect to ϵ and $MinPts$ consists of only one cluster containing almost all points of the database. Then, the information of all clustering levels will be contained in the reachability-plot.

The MinPts-Parameter. The effect of the *MinPts*-value on the visualization of the cluster-ordering can also be seen in Figure 123. The overall shape of the reachability-plot is very similar for different *MinPts* values. However, for lower values the reachability-plot looks more jagged and higher values for *MinPts* smoothen the curve. Moreover, high values for *MinPts* will significantly weaken possible “single-link” effects.

9.2 Experimental Evaluation

In this section, we will evaluate the similarity models introduced in Chapter 8 by applying the density-based hierarchical clustering algorithm OPTICS to two real world test data sets. In Section 9.2.1, we shortly describe the two data sets and the parameter settings of our similarity models. In Section 9.2.2, we will evaluate the effectiveness of the space partitioning models, i.e. the volume model, the solid-angle model and the eigen-value model. In Section 9.2.3, we turn our attention to the quality of the data partitioning models, i.e. the cover-sequence model and the vector set model. Finally, we will present a short summary of the main results.

9.2.1 Experimental Setup

Data Sets. We evaluated the proposed models on the basis of two real-world data sets. The first one - in the following referred to as *CAR* data set - contains approximately 200 CAD objects from a German car manufacturer. The *CAR* data set contains several groups of intuitively similar objects, e.g. a set of tires, doors, fenders, engine blocks and kinematic envelopes of seats.

The second data set contains 5,000 CAD objects from an American aircraft producer and is called *PLANE* data set in the following. This data set contains many small objects (e.g. nuts, bolts, etc.) and a few large ones (e.g. wings).

Parameter Settings. For the volume model, the solid-angle model and the eigen-value model we used a raster resolution of $r = 30$. Thus, $|V^o|$ ranges from 1 to $30^3 = 27,000$ for each object o . Furthermore, the data space is partitioned into $p = 3$ cells in each dimension. We retrieve $3^3 = 27$ -dimensional feature vectors for the volume model and the solid-angle model. The eigen-value model yields feature vectors of dimensionality $3 \cdot 3^3 = 81$.

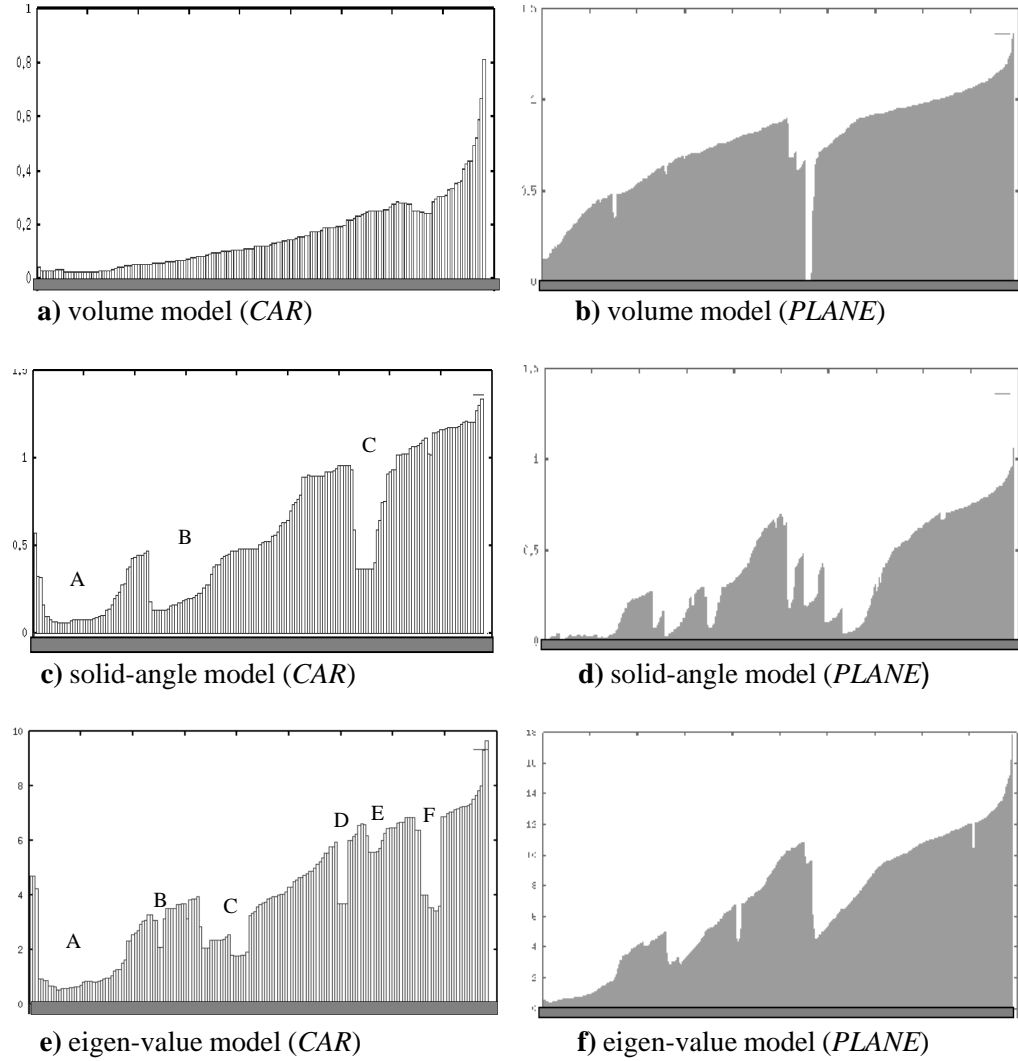


Figure 124: Reachability plots for the space partitioning similarity models.

Using the cover sequence model and the vector set model, the data space of both data sets contains objects represented as voxel approximations using a raster resolution of $r = 15$. Here, $|V^o|$ ranges from 1 to $15^3 = 3375$ for each object o .

These values were optimized to the quality of the evaluation results.

9.2.2 Evaluation of the Space Partitioning Similarity Models

Evaluation of the Volume-Model. The reachability plots computed by OPTICS using the volume model for both the *CAR* data set and the *PLANE* data set are depicted in Figure 124a and 124b. Both plots show a minimum of structure indicating that

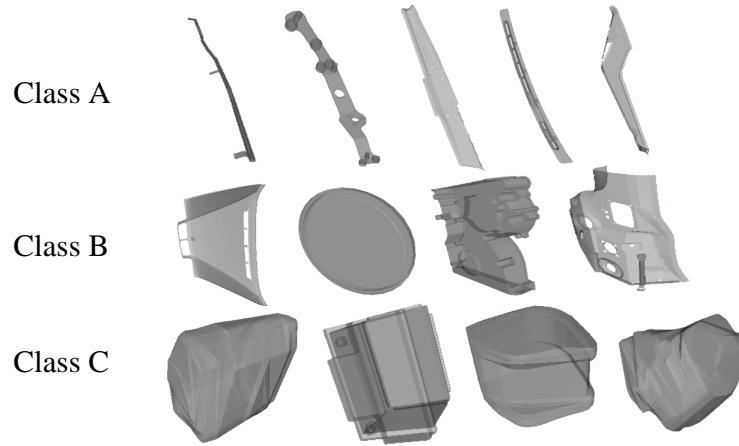


Figure 125: Objects found by the solid-angle model (cf. Figure 124c).

the volume model cannot satisfyingly represent the intuitive notion of similarity. None of the groups described in Section 9.2.1 were distinguished by the clustering algorithm. Although we get satisfying results for specific k -nn queries (cf. Figure 118a where the “bad model” was the volume model), the volume model is rather ineffective if applied to the whole data set. This indicates the suitability of clustering to evaluate the quality of similarity models.

Evaluation of the Solid-Angle Model. The reachability plots computed by OPTICS using the solid-angle model for both the CAR data set and the PLANE data set are depicted in Figure 124c and 124d. On the CAR data set the solid-angle model provides three clusters denoted as *A*, *B*, and *C* in Figure 125. We analyzed the resulting clusters by picking samples out of the set of objects grouped in each cluster. The result of this evaluation is presented in Figure 125. As it can be seen, cluster *A* consists mainly of long and thin objects. This might be still inside the intuitive notion of similarity. The same observation can be made for the objects in cluster *C*. But the objects that are grouped together in cluster *B* are no more intuitively similar. Furthermore, there are clusters of intuitively similar objects (e.g. doors) which are not detected.

Evaluating the solid-angle model using the PLANE data set we made similar observations. The reachability plot computed by OPTICS (cf. Figure 124d) yields a clustering with a large number of hierarchical classes. But the analysis of the objects within each cluster displays that intuitively dissimilar objects are counted as similar

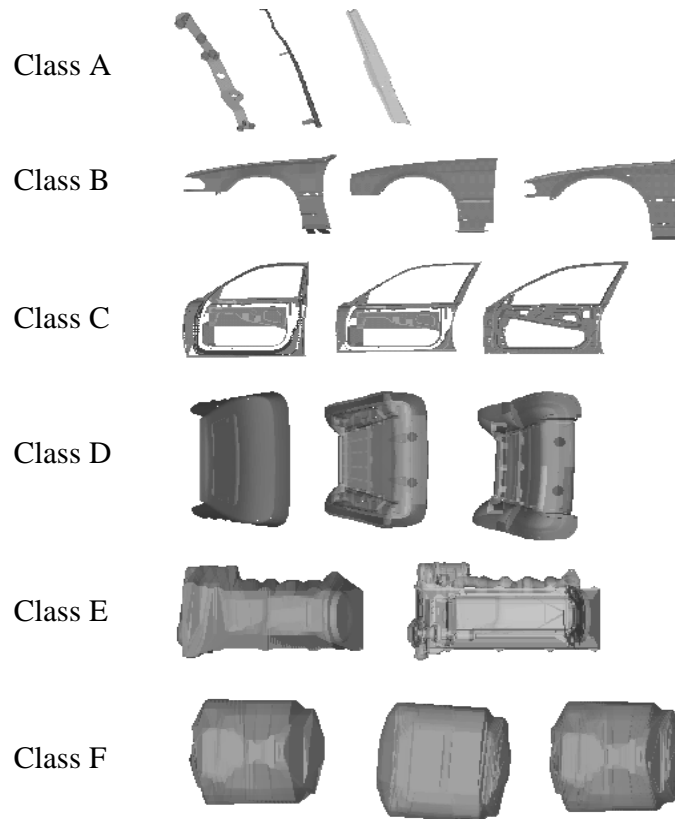


Figure 126: Objects found by the eigen-value model (cf. Figure 124e).

according to the model. A further observation is the following: objects, that are intuitively similar, are clustered in different groups.

To sum up, the solid-angle model does not generate all clusters for the CAR data set, whereas for the PLANE data set it yields clusters with dissimilar parts. This suggests the conclusion that the solid-angle model is also rather unsuitable as a similarity model for our real-world test data sets.

Evaluation of the Eigen-Value Model. In contrast to the other two approaches, the eigen-value model yields valuable results. The plots computed by OPTICS for the eigen-value model are presented in Figure 124e and 124f.

On the CAR data set (cf. Figure 124e) OPTICS finds six clusters which are analyzed in Figure 126. Each class consists of intuitive similar objects. Similar to the solid-angle model (cf. Figure 125), Class A represents a large number of small and thin objects. Class B consists of fenders, class C represents doors, all objects in class D are seats, class E consists of engine blocks and class F represents kinematic enve-

lopes of seats. These results show that based on the eigen-value model, OPTICS finds a lot of meaningful classes in the CAR data set.

Analyzing the PLANE data set with OPTICS based on the eigen-value model yields affirmative results as well. The reachability plot (cf. Figure 124f) depicts five clusters containing similar results.

9.2.3 Evaluation of the Data Partitioning Similarity Models

The plots computed by OPTICS for the *cover sequence model*, the *cover sequence model using the minimum Euclidean distance under permutation* and the *vector set model* (cf. Figure 127, 128 and 129) look even better than the plots computed for the eigen value model.

We will confirm this observation in the following, evaluating the effectiveness of the different models. We analyzed the cover sequence model without permutations as well as under full permutations, i.e. using the Euclidian distance under permutation. Note that the Euclidian distance under permutation is too time consuming for a straightforward calculation, since the runtime complexity increases with the faculty of the number of chosen covers. Therefore, we used the possibility of deriving this distance measure from the matching distance by employing the calculation via the Kuhn-Munkres algorithm as described in Section 8.3.2. Remember that this is achieved by using the squared Euclidian distance for comparing single feature vectors and drawing the square root from the result. The resulting plots (cf. Figure 128) look quite similar to the ones we derived from employing the minimal matching distance based on the normal Euclidian distance, i.e. using the vector set model (cf. Figure 129c and 129d). A careful investigation of the parts contained in the clusters showed that the cover sequence model using the minimum Euclidean distance under permutation and the vector set model lead to basically equivalent results. Due to this observation and the better possibilities for speeding up similarity queries (cf. Chapter 10), we concentrated on the evaluation of the vector set model. We first compared the vector set model to the cover sequence model without permutations (cf. Figure 127). Furthermore, we used different numbers of covers for the vector set model (cf. Figure 129) in order to show the benefits of a relatively high number of covers for complex CAD objects.

Comparing the vector set model with the cover sequence model on the CAR data set (cf. Figure 127a, 129a and 129c) we conclude, that the vector set model is superi-

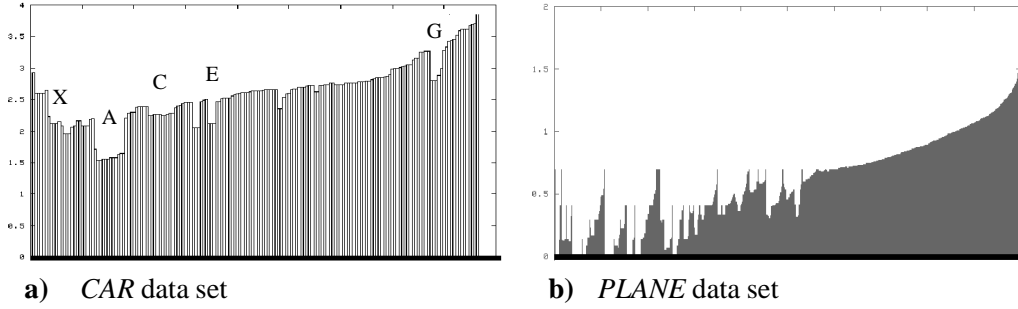


Figure 127: Reachability plots for the cover sequence model with 7 covers.

or. All plots look similar on the first glance. When evaluating the clusters (cf. Figure 130 and 131), it turned out that there are clusters which are detected by both approaches and thus appear in both plots, e.g. classes *E* in Figure 130 and 131. Nevertheless, we observed the following three shortcomings of the cover sequence model:

- Meaningful hierarchies of clusters detected by the vector set model, e.g. G_1 and G_2 in Figure 129c which are visualized in Figure 131 are lost in the plot of the cover sequence model (Class *G* in Figure 127a evaluated in Figure 130).
- Some clusters found by the vector set model are not found using the cover sequence model, e.g. cluster *F* in Figure 131.
- Using the cover sequence model, objects that are not intuitively similar are clustered together in one class (e.g. class *X* in Figure 127a which is evaluated in Figure 130). This is not the case when using the vector set model. A reason for the

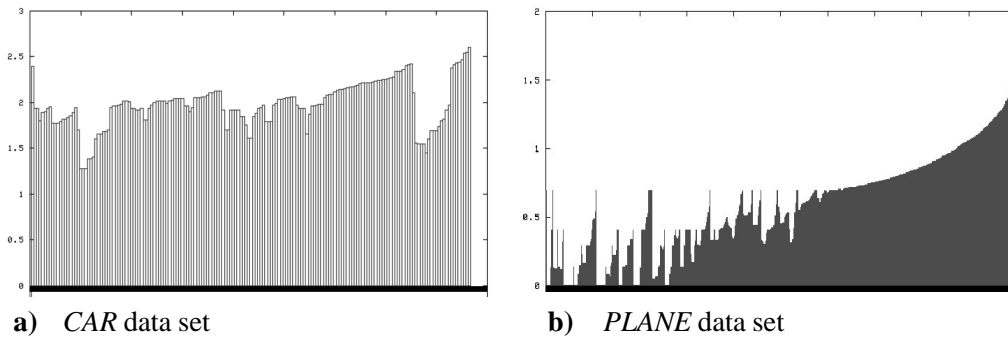


Figure 128: The minimum Euclidean distance under permutation.
Reachability plot for the cover sequence model with the minimum Euclidean distance under permutation with 7 covers.

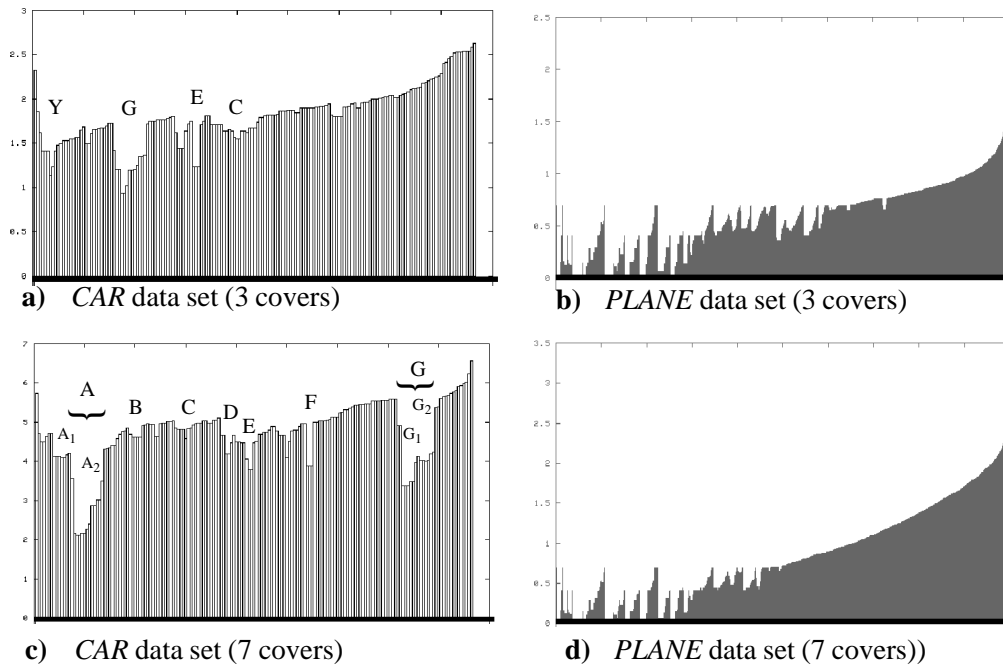


Figure 129: Reachability plots for the vector set model.

superior effectiveness of the vector set model compared to the cover sequence model is the role of permutations of the covers. This is supported by the observations which are depicted in Table 9. In most of all distance calculations carried out during an OPTICS run there was at least one permutation necessary to compute the minimal matching distance.

number of covers	percentage of permutations
3	68.2%
5	95.1%
7	99.0%
9	99.4%

Table 9: Percentage of proper permutations for the vector set model.

The plots in Figure 129a and 129c compare the influence of the number of covers used to generate the vector sets on the quality of the similarity model. An evaluation of the clusters yields the observation, that 7 covers are necessary to model real-world

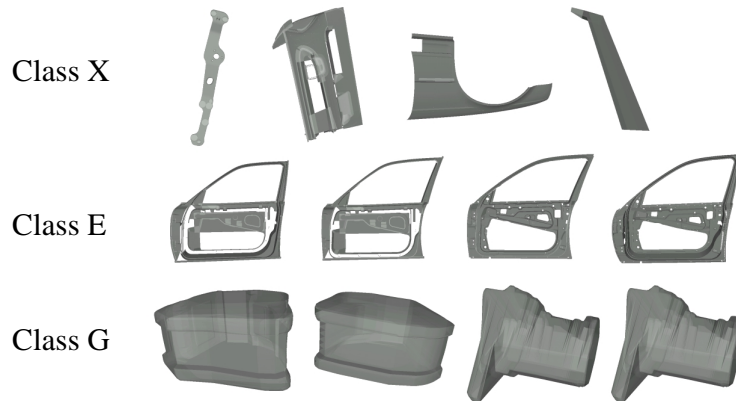


Figure 130: Objects found by the cover sequence model (cf. Figure 127a).

CAD objects accurately. Using only 3 covers we observed basically the same three problems already noticed when applying the cover sequence model.

All the results of the evaluation on the CAR data set can also be observed evaluating the models on the PLANE data set. As a consequence, the evaluation shows that the vector set model outperforms the other models with respect to effectiveness. Furthermore, we see that we need about 7 covers to model similarity most accurately.

9.2.4 Summary

In this section, we shortly summarize the main results of our experimental evaluation. Generally spoken, the data partitioning similarity models, i.e. the *cover sequence model* and the *vector set model*, reflect the intuitive similarity between CAD objects better than the space partitioning similarity models, i.e. the *volume model*, the *solid-angle model* and the *eigen-value model*.

In Section 9.2.2, we showed that the eigen-value model produces the most meaningful results among the three introduced space partitioning similarity models. Using the eigen-value model as basis for the OPTICS run results in clusters containing intuitively similar objects. Nevertheless, the eigen value model suffers from the same shortcomings as the data partitioning cover sequence model. Although both model produce rather meaningful clusters, they fail to detect important cluster hierarchies.

In Section 9.2.2, we showed that only the vector set model detects these hierarchies, which are important for navigating through massive data sets (cf. Chapter 11). To sum up, our new evaluation method based on clustering showed that the combina-

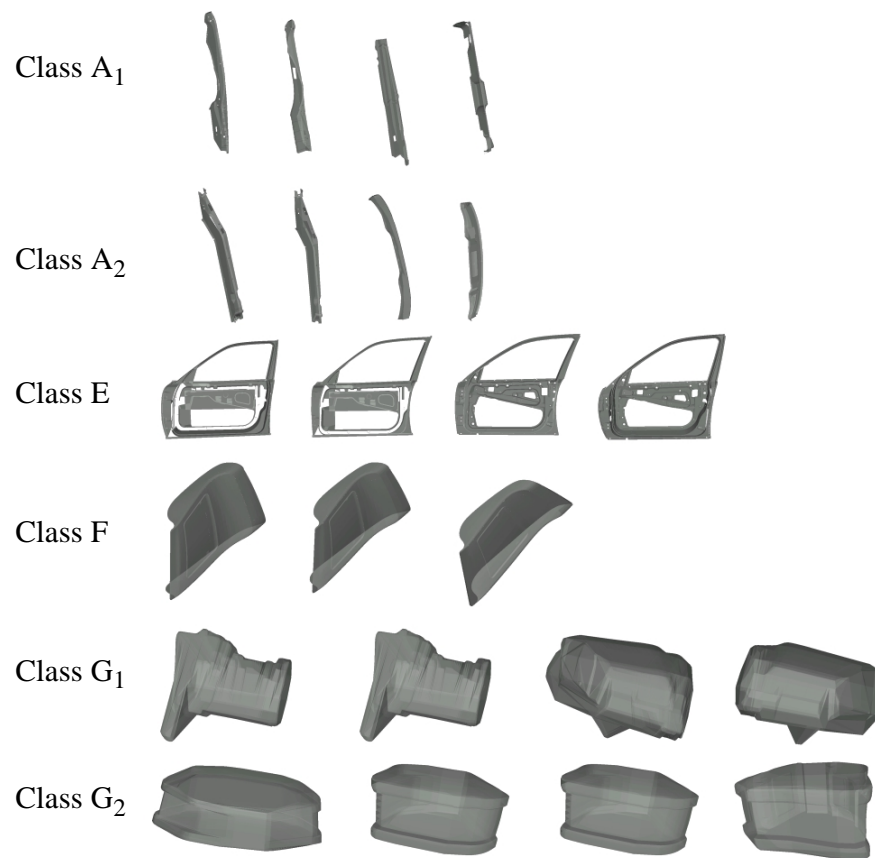


Figure 131: Objects found by the vector set model (cf. Figure 129c).

tion of the data partitioning cover sequence model and the new paradigm of using sets of feature vectors for representing objects is a very powerful approach for the detection of similar CAD parts.

Chapter 10

Efficiency of Similarity Models

In this chapter, we present the efficiency evaluation of the five similarity models introduced in Chapter 8. During this chapter, we do not divide the five models into space- and data-partitioning models, but instead we use a different partitioning which divides the similarity models into models using single feature vectors, i.e. the *volume model*, the *solid-angle model*, the *eigen-value model* and the *cover sequence model*, and models using sets of feature vectors, i.e. the *vector set model*. In order to accelerate the query processing for the feature based similarity models, we can use any of the presented index structures discussed in Section 7.3. We decided to use the NB-tree which can easily be integrated into an ORDBMS (cf. Section 10.1). In order to accelerate the query processing for the vector set model, we introduce three different filter steps, the *centroid approach*, the *Euclidean norm approach* and the *closest pair approach* (cf. Section 10.2). Furthermore, we present the Relational M-tree along with suitable optimizations (cf. Section 10.3). Finally, we present the experimental efficiency evaluation of our different models. This evaluation is based on two real world test data sets for which we investigate both k -nn queries as well as ϵ -range queries (cf. Section 10.4).

10.1 The NB-Tree: A Filter for the Single Vector Models

The NB-tree proposed by Fonseca and Jorge [FJ 03] is a promising approach for multi-step similarity search on top of an ORDBMS. As outlined in Section 7.3.2, the NB-tree maps d -dimensional feature vectors \vec{x} to one-dimensional values by computing their Euclidean norm $\|\vec{x}\|$, which can be organized by a common B^+ -tree. By employing the Euclidean norm for dimension reduction, the NB-tree combines several beneficial properties.

First, similar objects, i.e. objects with a small distance between their corresponding feature vectors, also have similar norm values, i.e. the norm values are positioned near to each other on the 1D-line. A schema of the NB-tree for the 2D-case is depicted in Figure 132.

Second, as mentioned in Section 8.1.1, when performing similarity search in CAD applications, all 48 permutations resulting from 90° -rotations and reflections of the modelled CAD objects have to be taken into account. As these rotations and reflections are accomplished on a feature vector \vec{x} by merely resorting the feature values x_i of single dimensions within the vector and possibly switching the algebraic sign for some of the values x_i , the Euclidean norm of \vec{x} is invariant to rotation and reflection. Thus, a single distance computation between the norms of a query object q and a database object o is sufficient in the filter step of a similarity query.

Third, the NB-Tree fully exploits the advantages of the B^+ -tree. As an implementation of the B^+ -tree is already provided by all commercial ORDBMSs, no additional implementation effort is needed. The underlying B^+ -tree organizes all data objects in ascending order of the indexed values on the leaf level and links these leaves sequentially. This ordering can easily be exploited for the efficient processing of ε -range queries (cf. Section 7.4.1), k -nn queries (cf. Section 7.4.2) or ranking queries (cf. Section 7.4.3). For instance, when computing similarity range queries, the Euclidean norms of all candidate objects lies within a range from $\|q\| - \varepsilon$ to $\|q\| + \varepsilon$ (cf. Figure 132). By means of a single range scan, we can determine the candidate set for which the exact distance calculation has to be carried out (cf. Section 7.4.1). As outlined in Section 7.4.2 and 7.4.3, we also need only 2 range scans for determining the candidate set based on the NB-tree for k -nn queries and ranking queries.

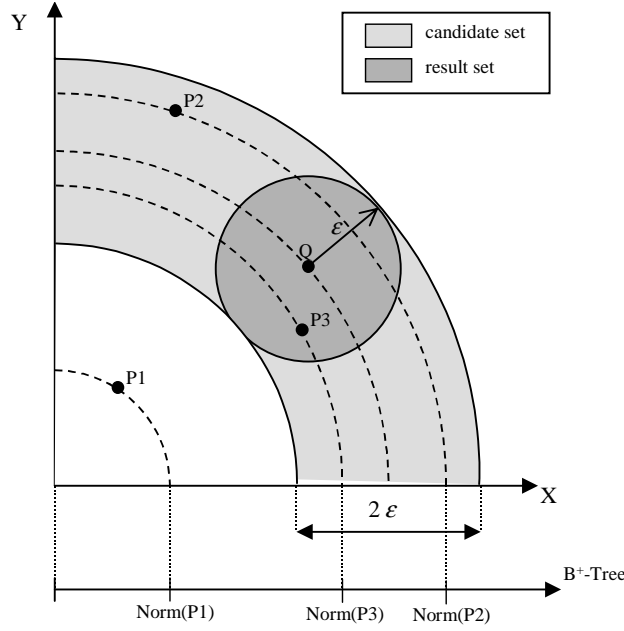


Figure 132: An ϵ -range query on an NB-tree.

In our efficiency studies, we employed the NB-tree as a filter for similarity range and k -nearest neighbor queries on single-vector represented CAD objects. As object distance function d_o on the high-dimensional feature vectors, we used the Euclidean distance (L_2 -distance). Additionally, the filter distance function d_f between the norm values according to the Manhattan distance (L_1 -distance) was used. In order to get correct search results, we have to show that d_f lower bounds d_o (cf. Definition 22), i.e. the L_1 -distance between the Euclidean norms of \vec{x} and \vec{y} underestimates the Euclidean distance between \vec{x} and \vec{y} for all $\vec{x}, \vec{y} \in \mathbb{R}^d$.

Lemma 8. Let $\vec{x}, \vec{y} \in \mathbb{R}^d$ be two arbitrary d -dimensional feature vectors. Then the difference between the L_2 -norms of \vec{x} and \vec{y} underestimates the L_2 -distance between \vec{x} and \vec{y} :

$$|\|\vec{x}\|_2 - \|\vec{y}\|_2| \leq \|\vec{x} - \vec{y}\|_2$$

Proof.:

$$\begin{aligned} \|\vec{x}\|_2 &= \|\vec{x} - \vec{0}\|_2 \stackrel{\text{tri. ineq.}}{\leq} \|\vec{x} - \vec{y}\|_2 + \|\vec{y} - \vec{0}\|_2 = \|\vec{x} - \vec{y}\|_2 + \|\vec{y}\|_2 \Rightarrow \|\vec{x}\|_2 - \|\vec{y}\|_2 \leq \|\vec{x} - \vec{y}\|_2 \\ \|\vec{y}\|_2 &= \|\vec{y} - \vec{0}\|_2 \stackrel{\text{tri. ineq.}}{\leq} \|\vec{x} - \vec{y}\|_2 + \|\vec{x} - \vec{0}\|_2 = \|\vec{x} - \vec{y}\|_2 + \|\vec{x}\|_2 \Rightarrow \|\vec{y}\|_2 - \|\vec{x}\|_2 \leq \|\vec{x} - \vec{y}\|_2 \end{aligned}$$

$$\text{Then } |\|\vec{x}\|_2 - \|\vec{y}\|_2| = \max(\|\vec{x}\|_2 - \|\vec{y}\|_2, \|\vec{y}\|_2 - \|\vec{x}\|_2) \leq \|\vec{x} - \vec{y}\|_2 \quad \blacksquare$$

10.2 Filters for the Minimal Matching Distance

Though we discussed the time for a single distance calculation for sets of feature vectors (cf. Section 8.3.2), the problem of efficiently processing similarity queries in large databases is still unanswered. Since it is necessary here, to locate the objects belonging to the result in comparably short time, the use of index structures that avoid comparing a query object to the complete database is mandatory. For one-vector-represented data objects there exists a wide variety of index structures that are suitable for answering similarity queries efficiently like the TV-Tree [LJF 94], the X-Tree [BKK 96] or the IQ-Tree [BBJ+ 00]. But unfortunately, those index structures cannot be used directly to retrieve vector-set-represented objects.

To accelerate similarity queries on vector-set-represented objects, the simplest approach is the use of more general access structures. Since the minimal matching distance is a metric for the right choice of distance and weight function, the use of index structures for metric objects like the M-Tree [CPZ 97] offers a good possibility here. Another approach is the use of a multi-step strategy to process spatial queries. First, a *filter step* is executed returning a superset of the objects qualifying for the spatial predicate. A cascade of subsequent filter steps may further reduce the number of candidates. The multi-step query process is finished by the *refinement step*, which computes the exact minimal matching distance between two sets of feature vectors. As the minimal matching distance on vector sets has a polynomial runtime complexity of $O(k^3 + k^2d)$, where the cardinality of both sets is not greater than k and each vector has a dimensionality of d , the employment of high-quality filter distance functions for similarity search is very important. Especially in the case of partial similarity search, we benefit from selective filters. For more details about efficient partial similarity search on vector set data we refer the interested reader to [BKP 04a]. In this thesis we concentrate on complete similarity search on vector set data.

In the following, we will introduce three different filters, the *centroid approach* (cf. Section 10.2.1), the *Euclidean norm approach* (cf. Section 10.2.2) and the *closest pair approach* (cf. Section 10.2.3). Furthermore, we shortly discuss how we can beneficially combine them (cf. Section 10.2.4). Finally, we will close this section with a short summary and a few remarks on efficient partial similarity search (cf. Section 10.2.4).

10.2.1 The Centroid Approach

This filter step is based on the relation between a set of d -dimensional vectors and its *extended centroid*.

Definition 44 (Extended Centroid).

Let $V \subset \mathbb{R}^d$ and $\vec{w} \in \mathbb{R}^d \setminus V$. Let $X \in 2^V$ be a vector set with $|X| \leq k$ and $X = \{\vec{x}_1, \dots, \vec{x}_{|X|}\}$. Then the extended centroid $C_{k, \vec{w}}(X)$ is defined as follows:

$$C_{k, \vec{w}}(X) = \frac{\sum_{i=1}^{|X|} \vec{x}_i + (k - |X|) \cdot \vec{w}}{k}$$

Note how the vector \vec{w} is used as a “dummy” vector to fill up vector sets having a cardinality of less than k (cf. Section 8.3.2). Based on this definition, we can state the following theorem.

Theorem 1 (The Centroid Filter).

Let $V \subset \mathbb{R}^d$ and $\vec{w} \in \mathbb{R}^d \setminus V$. Let $w_{\vec{w}}: V \rightarrow \mathbb{R}$, $w_{\vec{w}}(\vec{x}) = \|\vec{x} - \vec{w}\|_2$, be a weight function for unmatched nodes. Furthermore, let $X, Y \in 2^V$, $X = \{\vec{x}_1, \dots, \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \dots, \vec{y}_{|Y|}\}$ be two vector sets with $|X|, |Y| \leq k$, let $C_{k, \vec{w}}(X), C_{k, \vec{w}}(Y)$ be their extended centroids and let $d_{mm}^{L_2, w_{\vec{w}}}: 2^V \times 2^V \rightarrow \mathbb{R}$ be the minimal matching distance using $w_{\vec{w}}$ as a weight function defined on V . Then the following inequality holds:

$$k \cdot \|C_{k, \vec{w}}(X) - C_{k, \vec{w}}(Y)\|_2 \leq d_{mm}^{L_2, w_{\vec{w}}}(X, Y)$$

Proof. Let π be the enumeration of the indices of Y that groups the y_i to x_i according to the minimum weight perfect matching. We assume w.l.o.g. $|X| = m \leq n = |Y|$.

$$\begin{aligned} & k \cdot \|C_{k, \vec{w}}(X) - C_{k, \vec{w}}(Y)\|_2 \\ &= k \cdot \left\| \frac{\sum_{i=1}^m \vec{x}_i + (k - m) \cdot \vec{w}}{k} - \frac{\sum_{i=1}^n \vec{y}_{\pi(i)} + (k - n) \cdot \vec{w}}{k} \right\|_2 \\ &= \left\| \sum_{i=1}^m \vec{x}_i - \sum_{i=1}^n \vec{y}_{\pi(i)} + (n - m) \cdot \vec{w} \right\|_2 \\ &= \left\| \sum_{i=1}^m \vec{x}_i - \sum_{i=1}^m \vec{y}_{\pi(i)} - \sum_{i=m+1}^n \vec{y}_{\pi(i)} + \sum_{i=m+1}^n \vec{w} \right\|_2 \end{aligned}$$

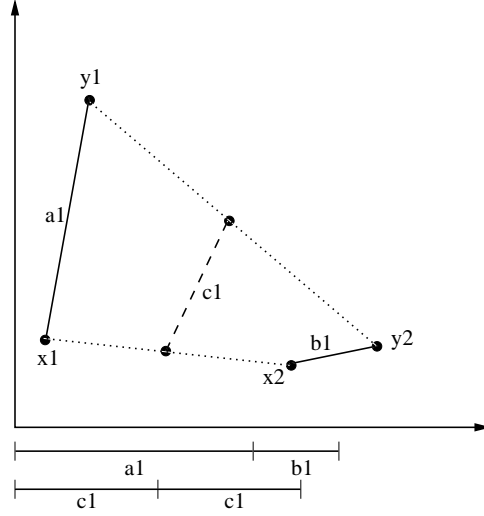
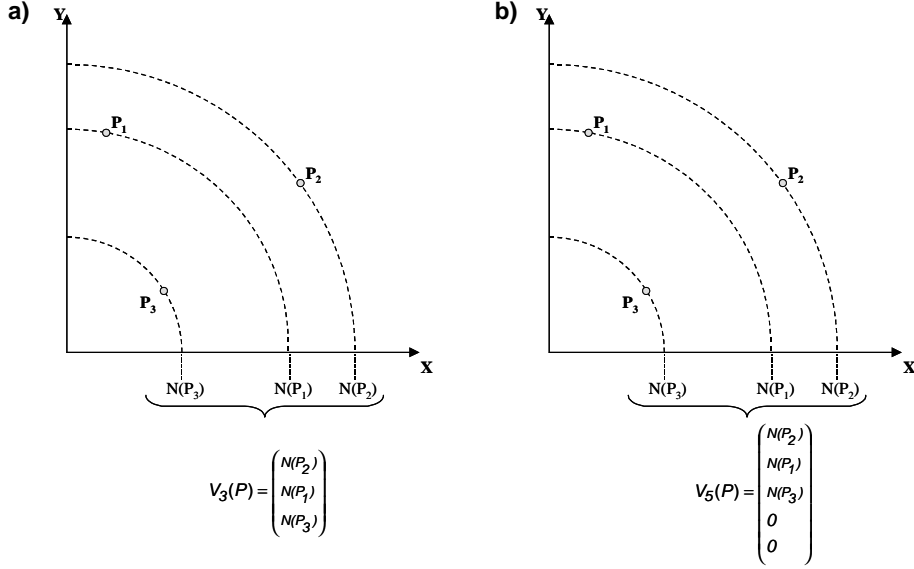


Figure 133: 2-dimensional example for the centroid filter.

$$\begin{aligned}
 \text{tri. ineq.} &\leq \left\| \sum_{i=1}^m (\vec{x}_i - \vec{y}_{\pi(i)}) \right\|_2 + \left\| \sum_{i=m+1}^n (\vec{w} - \vec{y}_{\pi(i)}) \right\|_2 \\
 \text{tri. ineq.} &\leq \sum_{i=1}^m \|\vec{x}_i - \vec{y}_{\pi(i)}\|_2 + \sum_{i=m+1}^n \|\vec{y}_{\pi(i)} - \vec{w}\|_2 \\
 &= \sum_{i=1}^m \|\vec{x}_i - \vec{y}_{\pi(i)}\|_2 + \sum_{i=m+1}^n w_{\vec{w}}(\vec{y}_{\pi(i)}) \\
 &= d_{mm}^{L_2, w_{\vec{w}}} (X, Y) \quad \blacksquare
 \end{aligned}$$

The lemma proves that the Euclidian distance between the extended centroids multiplied with the cardinality of the larger set is a lower bound (cf. Definition 22) for the minimal matching distance under the named preconditions. Therefore, when computing e.g. ε -range queries, we do not need to examine objects whose extended centroids have a distance to the query object q that is larger than ε/k . A good choice of \vec{w} for the cover sequence model is $\vec{0}$, since it has the shortest average distance within the position and has no volume. Since there are no covers having no volume in any data object, the conditions for the metric character of minimum matching distances are satisfied.

A 2-dimensional example for the extended centroid filter is depicted in Figure 133, where $|X| = |Y| = 2$ and $2c_1 = 2\|C_{k, \vec{w}}(X) - C_{k, \vec{w}}(Y)\|_2 \leq d_{mm}^{L_2, w_{\vec{w}}} (X, Y) = a_1 + b_1$.

**Figure 134:** The Euclidean norm vector.

The Euclidean norm vector $V_k(P)$ for a 2-dimensional point set P consisting of 3 points.

a) $V_3(P)$, b) $V_5(P)$

10.2.2 The Euclidean Norm Approach

In this section, we will introduce the *Euclidean norm vector*, which generalizes the NB-tree for vector set data. It is based on the Euclidean norms of all vector elements of a vector set. The idea is as follows: For all vectors \vec{x} of a vector set X , $|X| \leq k$, we compute the Euclidean norms $\|\vec{x}\|_2$ and organize these norm values in descending order in a k -dimensional vector, where k is the maximal length of the cover sequence approximating a CAD object. We call this filter *Euclidean norm vector*.

Definition 45 (Euclidean Norm Vector).

Let $V \subset \mathbb{R}^d$ and let $X \in 2^V$ be a vector set with $|X| \leq k$. Let $(\|\vec{x}_1\|_2, \dots, \|\vec{x}_{|X|}\|_2)$ be the sequence of the L_2 -norm values of the vectors in X in descending order, i.e. for all $i < j \in \{1, \dots, |X|\}$ holds $\|\vec{x}_i\|_2 \geq \|\vec{x}_j\|_2$. Then the Euclidean norm vector $V_k(X) = (v_1, \dots, v_k)^t \in \mathbb{R}^k$ is defined as follows:

$$v_i = \begin{cases} \|\vec{x}_i\|_2 & \text{for } i = 1, \dots, |X| \\ 0 & \text{for } i = |X| + 1, \dots, k \end{cases}$$

Note that if X has a cardinality smaller than k , dimensions $|X| + 1$ to k of the *Euclidean norm vector* are filled with 0 (cf. Figure 134).

We employ the Manhattan distance as a distance function between two Euclidean norm vectors $V_k(X)$ and $V_k(Y)$. This distance measure fulfills the *lower-bounding property* w.r.t. the minimal matching distance $d_{mm}^{L_2, w}$ between the corresponding vector sets X and Y , if the Euclidean norm is used as weight function for unmatched elements. Before we present the corresponding theorem (cf. Theorem 2), we will introduce two additional lemmas which are required for its proof.

Lemma 9. Let $V \subset \mathbb{R}^d$. Let $X = \{\vec{x}_1, \dots, \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \dots, \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $X \leq Y \leq k$. Then the following inequality holds:

$$\sum_{i=1}^{|X|} \left| \|\vec{x}_i\|_2 - \|\vec{y}_i\|_2 \right| \leq \sum_{i=1}^{|X|} \|\vec{x}_i - \vec{y}_i\|_2$$

Proof. The proposition holds if $\forall i \in \{1, \dots, |X|\} : \left| \|\vec{x}_i\|_2 - \|\vec{y}_i\|_2 \right| \leq \|\vec{x}_i - \vec{y}_i\|_2$ and this follows directly from Lemma 8. ■

Lemma 10. Let $V \subset \mathbb{R}^d$ and let $X, Y \in 2^V$ be two vector sets. Their Euclidean norm vectors are denoted by $V_k(X)$ and $V_k(Y)$. Let $(\|\vec{x}_1\|_2, \dots, \|\vec{x}_{|X|}\|_2)$ and $(\|\vec{y}_1\|_2, \dots, \|\vec{y}_{|Y|}\|_2)$ be the sequences of the L_2 -norm values of the vectors in X and Y in descending order. We assume w.l.o.g. $X \leq Y \leq k$. Let $\pi \in \Pi(Y)$. Then the following inequality holds:

$$\|V_k(X) - V_k(Y)\|_1 \leq \sum_{i=1}^{|X|} \left| \|\vec{x}_i\|_2 - \|\vec{y}_{\pi(i)}\|_2 \right| + \sum_{i=|X|+1}^{|Y|} \|\vec{y}_{\pi(i)}\|_2$$

Proof. (Sketch) Let $V_k(X) = (x_1, \dots, x_k)^t$, $V_k(Y) = (y_1, \dots, y_k)^t \in \mathbb{R}^k$.

We first show that $(*) \ \|V_k(X) - V_k(Y)\|_1 = \sum_{i=1}^k |x_i - y_i| \leq \sum_{i=1}^k |x_i - y_{\pi(i)}|$ holds.

Every given permutation $\pi \in \Pi(Y)$ can be constructed from adjacent permutations π_1, \dots, π_n , such that $\pi = \pi_1 \circ \dots \circ \pi_n$ and for each π_l there is some $p \in \{1, \dots, |X|\}$, such that $\pi_l(p) = p+1$, $\pi_l(p+1) = p$ and $\forall q \notin \{p, p+1\} : \pi_l(q) = q$. Given π_l , we show that $|x_p - y_{\pi_l(p)}| + |x_{p+1} - y_{\pi_l(p+1)}| \geq |x_p - y_p| + |x_{p+1} - y_{p+1}|$. There are in total six cases, because of the ordering within the Euclidean norm vectors:

- | | |
|---|---|
| 1. $x_p \leq x_{p+1} \leq y_{\pi_l(p+1)} \leq y_{\pi_l(p)}$ | 4. $y_{\pi_l(p+1)} \leq x_p \leq x_{p+1} \leq y_{\pi_l(p)}$ |
| 2. $x_p \leq y_{\pi_l(p+1)} \leq x_{p+1} \leq y_{\pi_l(p)}$ | 5. $y_{\pi_l(p+1)} \leq x_p \leq y_{\pi_l(p)} \leq x_{p+1}$ |
| 3. $x_p \leq y_{\pi_l(p+1)} \leq y_{\pi_l(p)} \leq x_{p+1}$ | 6. $y_{\pi_l(p+1)} \leq y_{\pi_l(p)} \leq x_p \leq x_{p+1}$ |

We exemplarily show the third case. The proofs of the other five cases are very similar.

$$\begin{aligned}
& |x_p - y_{\pi_l(p)}| + |x_{p+1} - y_{\pi_l(p+1)}| = y_{p+1} - x_p + x_{p+1} - y_p = \\
& (x_{p+1} - y_{p+1}) + (y_{p+1} - y_p) + (y_p - x_p) + (y_{p+1} - y_p) = \\
& |x_{p+1} - y_{p+1}| + |x_p - y_p| + 2|y_{p+1} - y_p| \geq \\
& |x_{p+1} - y_{p+1}| + |x_p - y_p|
\end{aligned}$$

As for each application of a π_l the sum on the right side of proposition (*) will grow or remain equal, the sum will grow or remain equal when applying π . Thus, proposition (*) holds. Finally, the following holds:

$$\begin{aligned}
\|V_k(X) - V_k(Y)\|_1 &\stackrel{(*)}{\leq} \sum_{i=1}^{|k|} |x_i - y_{\pi(i)}| = \\
\sum_{i=1}^{|X|} \left| \|\vec{x}_i\|_2 - \|\vec{y}_{\pi(i)}\|_2 \right| + \sum_{i=|X|+1}^{|Y|} \left| \|0\|_2 - \|\vec{y}_{\pi(i)}\|_2 \right| + \sum_{i=|Y|+1}^{|k|} \left| \|0\|_2 - \|0\|_2 \right| = \\
\sum_{i=1}^{|X|} \left| \|\vec{x}_i\|_2 - \|\vec{y}_{\pi(i)}\|_2 \right| + \sum_{i=|X|+1}^{|Y|} \|\vec{y}_{\pi(i)}\|_2 \quad \blacksquare
\end{aligned}$$

We can now prove the theorem about the Euclidean norm filter.

Theorem 2 (The Euclidean Norm Filter).

Let $V \subset \mathbb{R}^d$ and let $X, Y \in 2^V$ be two vector sets. Their Euclidean norm vectors are denoted by $V_k(X)$ and $V_k(Y)$. Furthermore, let $w: V \rightarrow \mathbb{R}$, $w(\vec{v}) = \|\vec{v}\|_2$, be the Euclidean norm and $d_{mm}^{L_2, w}: 2^V \times 2^V \rightarrow \mathbb{R}$ be the minimal matching distance using w as weight function. Then the following inequality holds:

$$\|V_k(X) - V_k(Y)\|_1 \leq d_{mm}^{L_2, w}(X, Y)$$

Proof. We assume w.l.o.g. $X \leq Y \leq k$. Let $(\|\vec{x}_1\|_2, \dots, \|\vec{x}_{|X|}\|_2)$ and $(\|\vec{y}_1\|_2, \dots, \|\vec{y}_{|Y|}\|_2)$ be the sequences of the L_2 -norm values of the vectors in X and Y in descending order. Let $\pi \in \Pi(Y)$ be the enumeration of Y that results from the minimum weight perfect matching of X and Y . We combine the results from Lemma 9 and Lemma 10.

$$\begin{aligned}
\|V_k(X) - V_k(Y)\|_1 &\stackrel{\text{Lemma 10}}{\leq} \sum_{i=1}^{|X|} \left| \|\vec{x}_i\|_2 - \|\vec{y}_{\pi(i)}\|_2 \right| + \sum_{i=|X|+1}^{|Y|} \|\vec{y}_{\pi(i)}\|_2 \\
&\stackrel{\text{Lemma 9}}{\leq} \sum_{i=1}^{|X|} \|\vec{x}_i - \vec{y}_{\pi(i)}\|_2 + \sum_{i=|X|+1}^{|Y|} \|\vec{y}_{\pi(i)}\|_2 \\
&= d_{mm}^{L_2, w}(X, Y) \quad \blacksquare
\end{aligned}$$

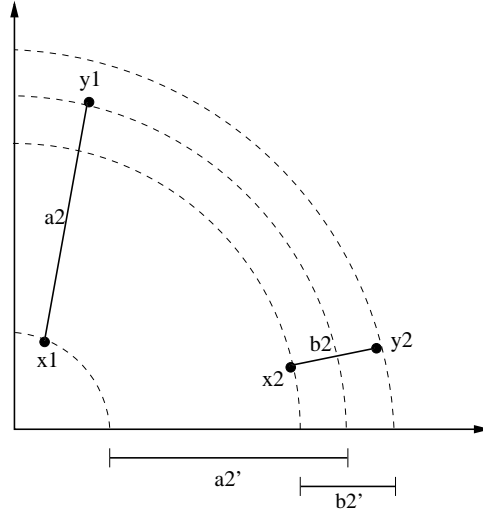


Figure 135: 2-dimensional example for the Euclidean norm filter.

A 2-dimensional example for the Euclidean norm filter is depicted in Figure 135, where $|X| = |Y| = 2$ and $a_2' + b_2' = \|V_k(X) - V_k(Y)\|_1 \leq d_{mm}^{L_2, w}(X, Y) = a_2 + b_2$.

10.2.3 The Closest Pair Approach

The closest pair distance between two vector sets X and Y can be used as a filter distance for the minimal matching distance $d_{mm}^{L_2, w}$ and is defined as follows.

Definition 46 (The Closest Pair Distance).

Let $V \subset \mathbb{R}^d$ and let $\vec{w} \in \mathbb{R}^d \setminus V$. Let $X = \{\vec{x}_1, \dots, \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \dots, \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $X \leq Y \leq k$. Let $X' = \{\vec{x}_1, \dots, \vec{x}_{|Y|}\}$ be a multiset with $\vec{x}_i = \vec{w}$ for $i \in \{|X| + 1, \dots, |Y|\}$. Then the closest pair distance $d_{cp}^{L_2}(X, Y): 2^V \times 2^V \rightarrow \mathbb{R}$ is defined as follows:

$$d_{cp}^{L_2}(X, Y) = \max \left(\sum_{i=1}^{|Y|} \min_{j \in 1, \dots, |Y|} (\|\vec{x}_i - \vec{y}_j\|_2), \sum_{i=1}^{|Y|} \min_{j \in 1, \dots, |Y|} (\|\vec{x}_j - \vec{y}_i\|_2) \right)$$

Let us note that the closest pair filter works directly on the set of vectors, i.e. on the original data, and not on approximated data. The filter distance can be computed by scanning the matrix of distance values between each pair of vectors in X and Y for the closest pairs. The overall runtime complexity is $O(k^2d)$. We will now show that the closest pair distance between two vector sets is a lower bound for the minimal matching distance, if the L_2 -distance is used as a weight function.

Theorem 3 (The Closest Pair Filter).

Let $V \subset \mathbb{R}^d$ and let $\vec{w} \in \mathbb{R}^d \setminus V$. Let $X = \{\vec{x}_1, \dots, \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \dots, \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $X \leq Y \leq k$. Furthermore, let $w: V \rightarrow \mathbb{R}$, $w(\vec{v}) = \|\vec{v} - \vec{w}\|_2$ be a weight function. Then the following inequality holds:

$$d_{cp}^{L_2}(X, Y) \leq d_{mm}^{L_2, w}(X, Y)$$

Proof. Let $\pi \in \Pi(Y)$ be the enumeration of Y that results from the minimum weight perfect matching of X and Y , i.e.

$$d_{mm}^{L_2, w}(X, Y) = \sum_{i=1}^{|X|} \|\vec{x}_i - \vec{y}_{\pi(i)}\|_2 + \sum_{i=|X|+1}^{|Y|} \|\vec{w} - \vec{y}_{\pi(i)}\|_2$$

The proof consists of two cases.

$$(1) d_{cp}^{L_2}(X, Y) = \sum_{i=1}^{|Y|} \min_{j \in 1, \dots, |Y|} (\|\vec{x}_i - \vec{y}_j\|_2).$$

$$\begin{aligned} \sum_{i=1}^{|Y|} \min_{j \in 1, \dots, |Y|} (\|\vec{x}_i - \vec{y}_j\|_2) &= \sum_{i=1}^{|X|} \min_{j \in 1, \dots, |Y|} (\|\vec{x}_i - \vec{y}_j\|_2) + \sum_{i=|X|+1}^{|Y|} \min_{j \in 1, \dots, |Y|} (\|\vec{w} - \vec{y}_j\|_2) \leq \\ &\sum_{i=1}^{|X|} \|\vec{x}_i - \vec{y}_{\pi(i)}\|_2 + \sum_{i=|X|+1}^{|Y|} \|\vec{w} - \vec{y}_{\pi(i)}\|_2 \end{aligned}$$

The inequality holds, if it holds for every pair of i -th addends. This is obviously the case, as we always pick the $\vec{y}_j \in Y$ which minimizes $\|\vec{x}_i - \vec{y}_j\|_2$.

$$(2) d_{cp}^{L_2}(X, Y) = \sum_{i=1}^{|Y|} \min_{j \in 1, \dots, |Y|} (\|\vec{x}_j - \vec{y}_i\|_2).$$

$$\begin{aligned} \sum_{i=1}^{|Y|} \min_{j \in 1, \dots, |Y|} (\|\vec{x}_j - \vec{y}_i\|_2) &= \sum_{i=1}^{|Y|} \min_{j \in 1, \dots, |Y|} (\|\vec{x}_j - \vec{y}_{\pi(i)}\|_2) \leq \\ &\sum_{i=1}^{|X|} \|\vec{x}_i - \vec{y}_{\pi(i)}\|_2 + \sum_{i=|X|+1}^{|Y|} \|\vec{w} - \vec{y}_{\pi(i)}\|_2 \end{aligned}$$

Again, the inequality holds, if it holds for every pair of i -th addends. This is obviously the case, as we always pick the $\vec{x}_j \in X$ which minimizes $\|\vec{x}_j - \vec{y}_{\pi(i)}\|_2$ (note that $\vec{w} \in X$ if $|X| < |Y|$). ■

A 2-dimensional example for the closest pair filter is depicted in Figure 136, where $|X| = |Y| = 3$ and $a'_3 + b_3 + c_3 = d_{cp}^{L_2}(X, Y) \leq d_{mm}^{L_2, w}(X, Y) = a_3 + b_3 + c_3$. During the filter distance calculation, x_3 is matched to both y_1 and y_3 , as $a' \leq a$, whereas the minimal matching distance is based on one-to-one matchings.

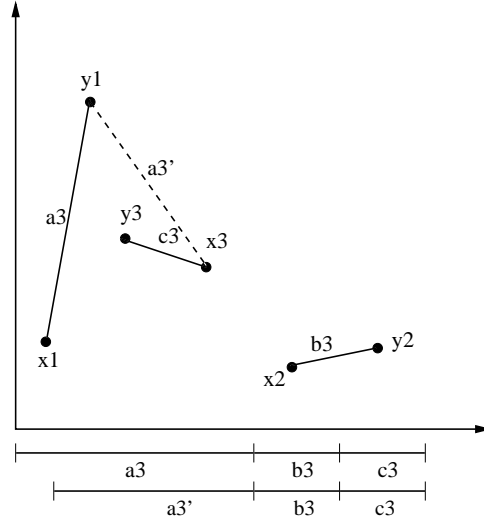


Figure 136: 2-dimensional example for the closest pair filter.

10.2.4 Combined Filters

The *centroid* and the *Euclidean norm vector* filtering techniques can profitably be combined. The exact distance computation is only performed if the results of both filter distance computations on the centroids and the Euclidean norm vectors are small enough. This way, a good deal of the information in the vector sets is incorporated in the filter distance computation. The centroid filter maps each dimension to a single value, resulting in a d -dimensional vector. On the other hand, the Euclidean norm filter maps each vector to a single value resulting in a k -dimensional vector. Thus, the combined filter contains aggregated information over both the dimensions and the vectors and is therefore suitable for a lot of different data distributions. The time complexity for a combined filter distance evaluation is $O(d+k)$.

10.2.5 Conclusion

If we compute the minimal matching distance between two vector sets, where the cardinality of both sets is not greater than k and each vector has a dimensionality of d , we can use the method proposed by Kuhn [Kuh 55] and Munkres [Mun 57] for an efficient computation of the minimal matching distance which has a time complexity of $O(k^3 + k^2d)$. Furthermore, we showed in [BKP 04a] that for partial similarity we can adapt this algorithm resulting in a runtime complexity of $O\left(\binom{k}{p}pk^2 + k^2d\right)$, where $p \in \{1, \dots, k\}$ reflects the degree of the desired partial similarity, i.e. the actual number of matched points.

As the computation of the minimal matching distance is rather time-consuming, we introduced three different filters, namely the *centroid filter*, the *Euclidean norm filter* and the *closest pair filter*. The centroid approach computes the mean value of all vectors for each dimension individually, resulting in a d -dimensional feature vector. The Euclidean norm approach computes the Euclidean norm for each feature vector, resulting in a k -dimensional feature vector. In contrast to the other two approaches, which derive a single feature vector for approximating a vector set, the closest pair filter works directly on the vector sets. The resulting distance measure lower bounds the minimal matching distance and can be computed more efficiently than the exact minimal matching distance. The runtime complexities for partial and complete similarity distance calculations based on the three different filters are summed up in the following table, where we assume a partial similarity parameter $p \in \{1, \dots, k\}$ and point sets containing k d -dimensional vectors [BKP 04a].

	centroid filter	Euclidean norm filter	closest pair filter
complete similarity	$O(d)$	$O(k)$	$O(k^2 d)$
partial similarity	not applicable	$O(k \log p)$	$O(k^2 d)$

Table 10: Filters for Vector Sets

10.3 The Optimized Relational M-Tree

In this section, we introduce the Relational M-tree (*RM-tree*) along with suitable optimizations for efficiently carrying out similarity range queries. We concentrate on range-queries for two reasons. First, range queries form the foundation of density-based clustering [EKSX 96]. Second, we can efficiently carry out k -nn queries based on range-queries [LS 02].

This section is organized as follows. After recalling, in Section 10.3.1, the M-tree as introduced in [CPZ 97], we present the RM-tree in Section 10.3.2. In Section 10.3.3, we show how we can adapt the concept of “*positive pruning*” (cf. Section 5.3) to the RM-tree. In Section 10.3.4, we combine the two worlds of direct metric index structures and multi-step query processing based on filtering. Whereas the filtering part of the M-tree is responsible for a good query response behavior for high selective queries, the concept of positive pruning accelerates low selective queries enormously which often occur in the area of density-based clustering. Furthermore, we show in this section that filters can be used for improving the query re-

sponse time of an M-tree and its creation. In Section 10.3.5, we show how caching can be applied to accelerate the processing of similarity range queries. The experimental evaluation regarding the optimized RM-tree is deferred to Section 10.4.3, where we compare our new metric indexing approach for complex objects with the different filters for vector set represented objects. Let us note that the presented optimizations for the M-tree can also be used for accelerating other metric index structure, e.g. the Slim-Tree [TTSF 00].

10.3.1 The M-tree

The M-tree (*metric tree*) [CPZ 97] is a balanced, paged and dynamic index structure that partitions data objects not by means of their absolute positions in the multi-dimensional feature space, but on the basis of the relative distances of the objects to each other. The only prerequisite is that the distance function between the indexed objects complies to the three requirements of a metric. Thus, the M-tree's domain is not confined to vector spaces but is much more general. For instance, the M-tree can be used to manage CAD objects represented by vector sets.

Tree-Structure. In the following we describe an M-tree and assume that the distance d between two objects forms a metric. The maximum size of all nodes of the M-tree is fixed. A leaf node entry contains objects o_d (or at least references to it), a suitable object representation for distance computations, and the distance $(o_d, P(o_d))$ of the object o_d to its parent object $P(o_d)$. Inner nodes contain so-called *routing objects*, which correspond to database objects to which a *routing role* was assigned by a promotion algorithm. This promotion algorithm is executed whenever a node has to be split. In addition to the leaf node entries, routing objects o_r also store their *covering radius* $r(o_r)$ and a pointer $ptr(T(o_r))$ to the root node of their subtree, the so-called *covering tree* of o_r . For all objects o_d in this covering tree, the condition holds that the distance $d(o_r, o_d)$ is smaller or equal to the covering radius $r(o_r)$. This property induces a hierarchical buildup of an M-tree, with the covering radius of a parent object always being greater or equal than all covering radii of their children and the root object of an M-tree storing the maximum of all covering radii.

In the following, we will shortly sketch the algorithms for similarity range queries as well as the routines for building up the M-tree as proposed in [CPZ 97]. For more details about the corresponding algorithms, we refer the interested reader to [CPZ 97].

```

1  FUNCTION SimRange (q QueryObject,  $\epsilon$  Range) RETURN ResultSet;
2  BEGIN
3      result := NIL;
4      rangeSearch (root, q,  $\epsilon$ );
5      RETURN result;
6  END;

```

```

1  PROCEDURE RangeSearch (N Node, q QueryObject,  $\epsilon$  Range);
2  BEGIN
3      op := parent object of node N;
4      IF (N is not a leaf) THEN
5          FOR EACH or in N DO
6              IF ( $|d(o_p, q) - d(o_p, o_r)| \leq r(o_r) + \epsilon$ ) THEN
7                  compute  $d(o_r, q)$ ;
8                  IF ( $d(o_r, q) \leq r(o_r) + \epsilon$ ) THEN
9                      RangeSearch (T(or), q,  $\epsilon$ );
10                 END IF;
11             END IF;
12         END FOR;
13     ELSE
14         FOR EACH od in N DO
15             IF ( $|d(o_p, q) - d(o_p, o_d)| \leq \epsilon$ ) THEN
16                 compute  $d(o_d, q)$ ;
17                 IF ( $d(o_d, q) \leq \epsilon$ ) THEN
18                     add od to result;
19                 END IF;
20             END IF;
21         END FOR;
22     END IF;
23 END;

```

Figure 137: Similarity range search on M-trees.

Similarity Range Queries. Given a query object q and a similarity range parameter ϵ , a similarity range query starts at the root node of an M-tree and recursively traverses the whole tree down to the leaf level, thereby pruning all subtrees which definitely do not contribute to the result set. A description of the algorithm *simRange* and the recursive procedure *rangeSearch* used to traverse the M-tree is given in Figure 137. In the following sections, we will refer to this algorithm and show how the presented optimizations can be included into the procedure *RangeSearch*.

The subtree of a routing object o_r can be pruned if the distance $d(o_r, q)$ between the routing object o_r and the query object q , is greater than the covering radii $r(o_r)$ plus ϵ , i.e. if the following condition holds (cf. Line 8 of the algorithm of Figure 137):

$$d(o_r, q) > r(o_r) + \epsilon$$

Furthermore, the subtree of a routing object o_r can also be pruned if the absolute value of the distance of the routing object's parent object o_p to the query object q , $d(o_p, q)$, minus the distance between o_r and o_p is greater than the covering radius $r(o_r)$ plus ϵ (cf. Line 6 of the algorithm of Figure 137):

$$|d(o_p, q) - d(o_p, o_r)| > r(o_r) + \epsilon$$

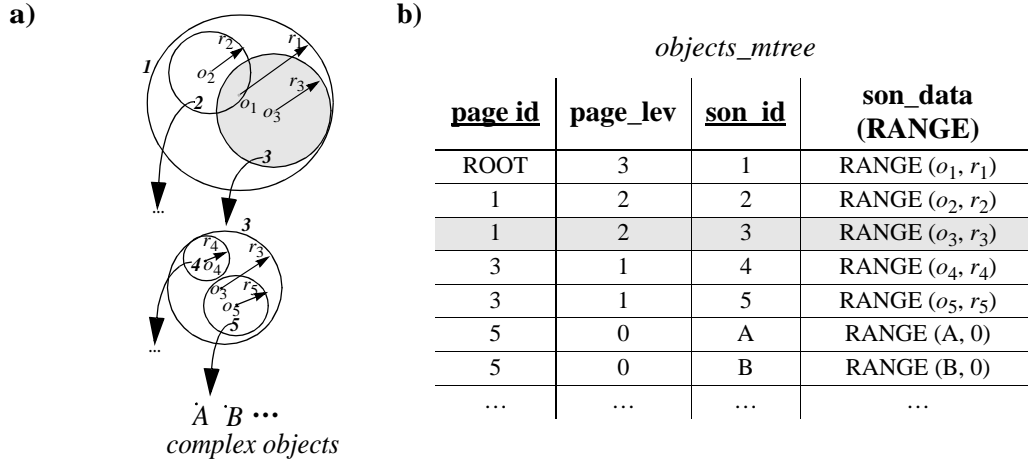
A proof based on the triangle inequality is presented in [CPZ 97]. Thus, as the distance between o_p and q has already been computed when accessing a node N , some subtrees can be pruned without further distance computations.

Insert. Similar to a range query, the insert algorithm recursively descends the M-tree to locate the “most suitable” leaf node for storing the new object o_n . At each level of the tree we descend along a subtree $T(o_r)$, for which no enlargement of the covering radius is needed, i.e. $d(o_r, o_n) \leq r(o_r)$. If multiple subtrees with this property exist, the one for which object o_n is closest to o_r is chosen. If no routing object exists for which $d(o_r, o_n) \leq r(o_r)$ holds, the increase of the covering radius, i.e. $d(o_r, o_n) - r(o_r)$, is minimized.

Split. If an object is inserted into an already full leaf node, a split is triggered. As any other dynamic balanced tree, an M-tree grows in a bottom-up way. The overflow of a node N is managed by allocating a new node N' at the same level of N , *partitioning* the entries among these two nodes, and *promoting* to the parent node N_p two routing objects to reference the two nodes N and N' while preserving the semantics of the covering radii. In [CPZ 97] many different split strategies, i.e. combinations of promoting and partitioning algorithms, were introduced. In this thesis, we concentrate on the *mM_Rad* approach which is the most promising promoting strategy [CPZ 97]. As partitioning strategy, we use the *generalized hyperplane* approach.

mM_Rad. This promoting algorithm considers all possible pairs (o_1, o_2) of objects belonging to the set of entries O of node N . After partitioning O , the pair of objects for which the maximum of the two cover radii becomes minimal is promoted, i.e. $o_1, o_2 \in O$ are promoted iff $\forall o_i, o_j \in O: \max(r(o_1), r(o_2)) \leq \max(r(o_i), r(o_j))$.

Generalized Hyperplane. Given a set of entries O and two routing objects o_1 and o_2 , each object $o_j \in O$ is assigned to the nearest routing object. Formally, if $d(o_j, o_1) < d(o_j, o_2)$ then o_j is assigned to o_1 else to o_2 .

**Figure 138:** Relational mapping of an M-tree directory.

a) Hierarchical directory, b) Index table

10.3.2 The Relational M-tree

In this section, we introduce the *Relational M-tree (RM-tree)*. In Figure 138, a possible relational mapping of an M-tree directory onto a relation (*page_id*, *page_lev*, *son_id*, *son_data*) is depicted. This mapping is similar to the mapping of the Relational R-tree as presented in Section 5.3. The main difference is that we use *routing objects* (o, r) consisting of an object o and a covering radius r instead of MBRs to describe a subtree. For instance, object o_3 together with a radius r_3 covers the subtree indicated by the shaded circle in Figure 138. For clarity, we omitted the distance of an object to its parent object.

Similarity Range Queries. Similarity range queries on the M-trees can be performed analogously to intersection queries on the R-trees (cf. Figure 139). The inter-

```

SELECT son_id AS id FROM objects_mtree
WHERE page_lev = 0 AND                                // select data object
      intersect (son_data, RANGE (:q, :eps))
START WITH page_id = ROOT
CONNECT BY
      intersect (PRIOR son_data, RANGE (:q, :eps)) AND
      PRIOR son_id = page_id                          // declarative tree traversal

```

Figure 139: SQL range query.

SQL range query around q for a radius eps on a Relational M-tree (Oracle syntax).

sect predicate detects whether two ranges intersect each other. If the range around the query object q with radius eps , does not intersect the range around the actual routing object o_i with radius r_i , the corresponding subtree can be pruned.

10.3.3 The Scanning M-Tree

In this section, we adapt our scanning approach based on the cost model for data-partitioning hierarchical index structures (cf. Section 5.3) to the RM-tree. We start with discussing similarity range queries which can be handled similar to intersection queries on the Relational R-tree (cf. Section 5.3) before we look at k -nn queries. Similar to Section 5.3, we use the following notations:

symbol	meaning
m	average number of index entries per directory node
$L(o_r)$	level of the current directory node o_r
$\sigma(q, o_r)$	value between 0 and 1 which denotes the percentage of accessed tuples in the subtree belonging to node o_r , if the index structure is used as usual for the query processing of a query q
k_{CPU}	CPU cost for testing one index directory entry
$k_{I/O}$	I/O cost for reading one page from the disk

Range Queries. As already outlined in Section 5.3, an accurate selectivity estimation is a very difficult task. This is especially true for metric data spaces. If our metric data space is IR^2 and we assume equal data distribution, we can estimate the selectivity similar to the selectivity estimation of collision queries on R-trees (cf. Figure 140). Unfortunately, such a straightforward computation is not possible for high dimensions or other kinds of metric data spaces. Other more general approaches for selectivity estimation are based on sampling and statistics.

If we use statistics, we can add a quantile vector to each routing object reflecting the distances of the elements contained in the subtree to the routing object o_r . Then, the selectivity $\sigma(q, o_r)$ can be computed similar to the approach presented in Section 4.3.2.

We propose to use sampling to estimate the selectivity. As samples we recommend to use the routing objects from level $L(o_r) - 1$ belonging to the subtree of our directory node o_r . These objects are arbitrarily distributed and represent the data stored in the

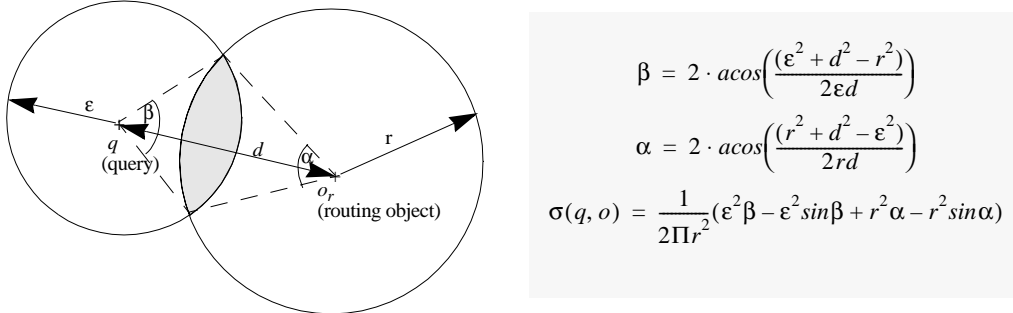


Figure 140: The overlap-factor σ for range queries on the Relational M-tree.
(Simplified determination of σ)

subtree belonging to directory node o_r very well. Note that all the sampling objects have the same *page_id* and are stored in one consecutive range on the disk (cf. Section 10.3.2). Furthermore, we suggest to memorize the performed distance computations as we need them later on anyway.

In contrast to the Relational R-tree, the CPU-cost in metric data spaces are often very high, e.g. distance computations on graphs [KS 03], trees [KKSS 04] or point sets [KBK+ 03]. If we only take these high CPU-cost into consideration and assume uniformly filled nodes, we perform an extended index range scan for a query q on a directory node o_r if

$$m^{L(o_r)} < m \cdot \left(1 + \sigma(q, o_r) \cdot \sum_{i=1}^{L(o_r)-1} m^i \right)$$

If we assume rather high values of m and a directory level $L(o_r)$ higher than 2, we scan if the following simplified condition is fulfilled

$$1 / \left(\sum_{i=1}^{L(o_r)-1} m^{i-(L(o_r)-1)} \right) < \sigma(q, o_r)$$

For high values of m , the left part of the above formula is very close to one. Therefore, we need almost a complete covering of the routing object o_r by the query object q in order to perform the extended range scan.

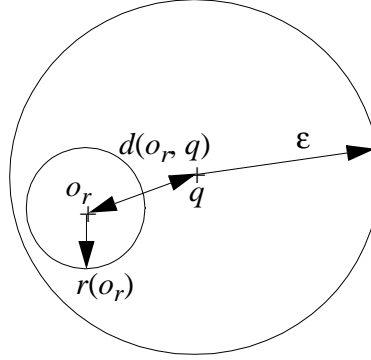


Figure 141: Positive Pruning for the M-tree.

If we also want to include the I/O cost, we can do this analogously to Section 5.3. Taking the I/O cost into consideration would trigger an extended range scan for smaller covering-factors, i.e. the index directory would not be used so extensively. Let us note that the decision whether to scan not only depends on an accurate estimation of the covering-factor σ but also on the ratio of k_{CPU} and $k_{I/O}$, i.e. on the used distance functions and the characteristics of the used computer.

A special case occurs, if the directory node is completely covered by the query range. In this case we can report all objects on the leaf level of the M-tree without performing any cost intensive distance computations (cf. Figure 141).

Lemma 11. Let $q \in O$ be a query object and $\varepsilon \in IR_0^+$ a query range. Furthermore, let o_r be a routing object with a covering radius $r(o_r)$ and a subtree $T(o_r)$. Then the following statement holds:

$$d(o_r, q) + r(o_r) \leq \varepsilon \Rightarrow \forall o \in T(o_r): d(o, q) \leq \varepsilon$$

Proof. The following inequalities hold for all $o \in T(o_r)$ due to the triangle inequality and due to our assumption that $d(o_r, q) + r(o_r) \leq \varepsilon$ holds:

$$d(o, q) \leq d(o, o_r) + d(o_r, q) \leq r(o_r) + d(o_r, q) \leq \varepsilon. \blacksquare$$

In the case of “*negative pruning*”, we skip the recursive tree traversal of a subtree $T(o_r)$, if the query range does not intersect the covering radius $r(o_r)$. In the case of “*positive pruning*” we skip the recursive tree traversal if the query range completely covers the covering radius $r(o_r)$. In this case we can report all objects stored in the corresponding leaf nodes of this subtree without performing any further distance


```

1  PROCEDURE RangeSearch (N Node, q QueryObject,  $\epsilon$  Range);
      :
      :
7      compute  $d(o_r, q)$ ;
7a     IF  $d(o_r, q) + r(o_r) \leq \epsilon$  THEN
7b         report all objects in  $T(o_r)$ ;
8     ELSE IF  $(d(o_r, q) \leq r(o_r) + \epsilon)$  THEN
      :
      :
```

Figure 142: Positive Pruning on M-trees.

computations. Figure 142 shows how this concept can be integrated into the original method *RangeSearch* depicted in Figure 137.

Note that this approach is very beneficial for accelerating density-based clustering on complex objects [KKPS 04]. DBSCAN, for instance, only needs the information whether an object is contained in $sim_{range}(q, \epsilon) = \{o \in DB / d(o, q) \leq \epsilon\}$, but not the actual distance of this object to the query object q . For a detailed discussion, we refer the interested reader to [KKPS 04].

10.3.4 The Filtering M-tree

So far the concepts of multi-step query processing and metric index structure have only been used separately. We claim that those concepts can easily be combined and that through the combination a significant speed-up compared to both separate approaches can be achieved. In the following, we will demonstrate the ideas for range queries.

The M-tree reduces the number of distance calculations by partitioning the data space even if no filters are available. Unfortunately, the M-tree may suffer from the navigational cost related to the distance computations during the recursive tree traversal. On the other hand, the filtering approach heavily depends on the quality of the filters.

When combining both approaches these two drawbacks are reduced. We use the filter distances to optimize the required number of exact object distance calculations needed to traverse the M-tree. Thereby, we do not save any I/O cost compared to the original M-tree, as the same nodes are traversed, but we save a lot of costly distance calculations necessary for the traversal. The filtering M-tree, stores the objects along with their corresponding filter values within the M-tree. A similarity query based on the filtering M-tree always computes the filter distance values prior to the exact dis-

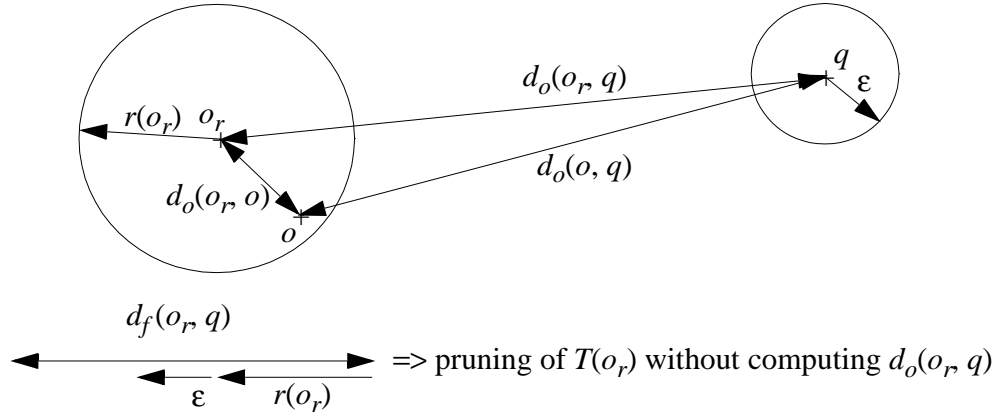


Figure 143: Similarity range query based on the filtering M-tree.

tance computations. If a filter distance value is already a sufficient criterion to prune branches of the M-tree, we can spare the exact distance computation. If we have several filters, the filter distance computation always returns the maximum value of all filters.

The pruning quality of the filtering M-tree benefits from both the quality of the filters and the clustering properties of the index structure. The filter distance can be used in addition to the distance of the routing object's parent object to the query object. In the following, we will show that the number of distance calculations used for range queries as well as for the creation of an M-tree can be optimized by using lower-bounding filters (cf. Definition 22).

Similarity Range Queries. Similarity range queries are used to retrieve all objects from a database which are within a certain similarity range from the query object (cf. Definition 19). By computing the filter distance prior to the exact distance we can save on many distance computation. Based on the following lemma, we can prune many subtrees without computing the exact distances between a query object q and a routing object o_r (cf. Figure 143).

Lemma 12. Let O be a set of objects and $d_o, d_f: O \times O \rightarrow IR_0^+$ be two distance functions, for which d_f lower bounds d_o , i.e. $\forall o_1, o_2 \in O: d_f(o_1, o_2) \leq d_o(o_1, o_2)$ holds. Let $q \in O$, $\epsilon \in IR_0^+$. For each routing object $o_r \in O$ in an M-tree with covering radius $r(o_r) \in IR_0^+$ and subtree $T(o_r)$ the following statement holds:

$$\forall o \in T(o_r) : (d_f(q, o_r) > r(o_r) + \epsilon) \Rightarrow d_o(q, o) > \epsilon$$

```

1  PROCEDURE RangeSearch (N Node, q QueryObject,  $\epsilon$  Range);
    :
    :
6      IF ( $|d(o_p, q) - d(o_p, o_r)| \leq r(o_r) + \epsilon$ ) THEN
6a         compute filter distance  $d_f(o_r, q)$ ;
6b         IF ( $d_f(o_r, q) \leq r(o_r) + \epsilon$ ) THEN
7             compute  $d(o_r, q)$ ;
8             IF ( $d(o_r, q) \leq r(o_r) + \epsilon$ ) THEN
    :
    :
15     IF ( $|d(o_p, q) - d(o_p, o_d)| \leq \epsilon$ ) THEN
15a        compute filter distance  $d_f(o_d, q)$ ;
15b        IF ( $d_f(o_d, q) \leq \epsilon$ ) THEN
16            compute  $d(o_d, q)$ ;
17            IF ( $d(o_d, q) \leq \epsilon$ ) THEN
    :
    :

```

Figure 144: Filtering M-tree.

Proof. As $\forall o_1, o_2 \in O: d_f(o_1, o_2) \leq d_o(o_1, o_2)$ holds, the following statement is true:

$$d_f(q, o_r) > r(o_r) + \epsilon \Rightarrow d_o(q, o_r) > r(o_r) + \epsilon.$$

Based on the triangle inequality and our assumption that $d_o(o, o_r) \leq r(o_r)$ holds, we can prove the above lemma as follows:

$$\begin{aligned}
 d_f(q, o_r) > r(o_r) + \epsilon &\Rightarrow d_o(q, o_r) > r(o_r) + \epsilon \Rightarrow d_o(q, o_r) - r(o_r) > \epsilon \\
 &\Rightarrow d_o(q, o_r) - d_o(o, o_r) > \epsilon \Rightarrow d_o(q, o) > \epsilon \blacksquare
 \end{aligned}$$

Let us note that a similar optimization can be applied, to the objects stored in the leaf level with the assumption that their “covering radius” is 0. Figure 144 shows how this concept can be integrated into the original method *RangeSearch* depicted in Figure 137.

Building of an M-tree. Filters can also beneficially be used for accelerating the creation of an M-tree, i.e. the insert routine (cf. Figure 145) and the split routine (cf. Figure 146).

Insert. Figure 145 depicts the routine *FindSubTree* which decides which tree to follow during the recursive tree-traversal of the insert operation. The main idea is that we sort all objects according to the filter distance and then walk through this sorted list. Thereby, we first test those candidates which might not lead to an increase in the covering radius. If we detect a routing object for which no increase is necessary, we postpone the reporting of this object. We first investigate all routing objects which are

```

FUNCTION FindSubTree ( $o_R$  RoutingObject,  $o$  Object) RETURN RoutingObject;
BEGIN
   $ActResult := (nil, false, \infty);$            //(object, InCovRad, distance)
  FOR EACH  $o_r$  in  $T(o_R)$  DO
    compute  $d_f(o_r, o)$ ;
  END FOR;
   $C_1 = \{o_r \mid d_f(o_r, o) - r(o_r) < 0\};$ 
   $C_2 = T(o_R) \setminus C_1$ ;
  Sort all  $o_r \in C_1$  and  $o'_r \in C_2$  ascending according to  $d_f(o_r, o)$ ,
  resulting in a  $SortedList = \langle o_{r1}, o_{r2}, \dots, o_{r|C1|} \rangle \circ \langle o'_{r1}, o'_{r2}, \dots, o'_{r|C2|} \rangle$ ;
  FOR EACH  $o_r$  in  $SortedList$  DO
    IF  $ActResult.InCovRad$  AND  $o_r \in C_2$  THEN
      RETURN  $ActResult.object$ ;
    END IF;
    IF  $d_f(o_r, o) > ActResult.distance$  THEN
      IF  $ActResult.InCovRad$  THEN
        RETURN  $ActResult.object$ ;
      ELSE
        compute  $d_o(o, o_r)$ ;
        IF  $(d_o(o, o_r) - r(o_r)) < 0$  THEN
           $ActResult := (o_r, true, d_o(o, o_r));$ 
        ELSE IF  $(d_o(o, o_r) - r(o_r)) < (ActResult.distance - r(ActResult.object))$  THEN
           $ActResult := (o_r, false, d_o(o, o_r));$ 
        END IF;
      END IF;
    ELSE
      compute  $d_o(o, o_r)$ ;
      IF  $ActResult.InCovRad$  THEN
        IF  $(d_o(o, o_r) - r(o_r) < 0)$  AND  $(d_o(o, o_r) < ActResult.distance)$  THEN
           $ActResult := (o_r, true, d_o(o, o_r));$ 
        END IF;
      ELSE
        IF  $(d_o(o, o_r) - r(o_r)) < 0$  THEN
           $ActResult := (o_r, true, d_o(o, o_r));$ 
        ELSE IF  $(d_o(o, o_r) - r(o_r)) < (ActResult.distance - r(ActResult.object))$  THEN
           $ActResult := (o_r, false, d_o(o, o_r));$ 
        END IF;
      END IF;
    END IF;
  END FOR;
  RETURN  $ActResult.object$ ;
END;

```

Figure 145: FindSubTree-function for an M-tree.

closer to the given query object and possibly also do not have to increase their covering radius. If several of those routing objects exist, we take the one closest to the inserted object. If no such routing object exists, we walk through the list until we have found the routing object which leads to a minimal increase of its covering radius. Let us note that this idea is closely related to the optimum multi-step k -nearest neighbor search algorithm [SK 98] presented by Seidl and Kriegel.

```

FUNCTION NodeSplit (N Node) RETURN PromotingObjects;
BEGIN
  ActResult := ((nil, nil), ∞); // ((PromotingObject1, PromotingObject2), mMRad)
  FOR EACH object pair (oi, oj) of node N DO
    compute  $d_f(o_i, o_j)$ ;
  END FOR;
  Compute for each of these pairs (oi, oj) the mMRad value mMradFilter based on the filters;
  Sort the resulting mMrad_filter values ascending,
  resulting in a SortedList = <(oa1, ob1, mMradFilter1), ..., (oan, obn, mMradFiltern)>;
  FOR EACH object oi of node N DO
    IF mMradFilteri > ActResult.mMRad THEN
      RETURN ActResult;
    END IF;
    mMRadi := 0;
    Sort all objects ok of node N descending according to  $\min (d_f(o_k, o_{a_i}), d_f(o_k, o_{b_i}))$ ;
    FOR EACH object ok of this sorted list DO
      IF  $d_f(o_k, o_{a_i}) < d_f(o_k, o_{b_i})$  THEN
        compute  $d_o(o_k, o_{a_i})$ ;
        IF  $d_o(o_k, o_{a_i}) < d_f(o_k, o_{b_i})$  THEN
          mMRadi := max (mMRadi,  $d_o(o_k, o_{a_i})$ );
        ELSE
          compute  $d_o(o_k, o_{b_i})$ ;
          mMRadi := max (mMRadi, min( $d_o(o_k, o_{a_i})$ ,  $d_o(o_k, o_{b_i})$ ));
        END IF;
      ELSE
        compute  $d_o(o_k, o_{b_i})$ ;
        IF  $d_o(o_k, o_{b_i}) < d_f(o_k, o_{a_i})$  THEN
          mMRadi := max (mMRadi,  $d_o(o_k, o_{b_i})$ );
        ELSE
          compute  $d_o(o_k, o_{a_i})$ ;
          mMRadi := max (mMRadi, min( $d_o(o_k, o_{a_i})$ ,  $d_o(o_k, o_{b_i})$ ));
        END IF;
      END IF;
    IF mMRadi > ActResult.mMRad THEN
      BREAK;
    END IF;
  END FOR;
  IF mMRadi < ActResult.mMRad THEN
    ActResult := ((oai, obi), mMRadi);
  END IF;
END FOR;
RETURN ActResult;
END;

```

Figure 146: NodeSplit-function for an M-tree.

Split. If a node overflow occurs due to an insertion, the node has to be split adequately. The “ideal” split strategy should promote two new routing objects, such that for the thereby obtained regions volume and overlap are minimized.

In Figure 146, it is shown how the filter distances can be used to speed-up the split of an M-tree node, i.e. the *mM_Rad* promoting strategy and the *generalized hyper-plane* partitioning strategy. The main idea is that we generate a priority queue con-

taining pairs of promoting objects based on the filter distances. We walk through this list and if we detect that the *mM_Rad* value based on the filters is higher than the best already found *mM_Rad* value based on exact distance computations, we can stop. Thus we do not necessarily have to test all $O(n^2)$ pairs of promoting objects. Again this approach is similar to [SK 98]. Furthermore, if we test two actual promoting objects o_{p_1} and o_{p_2} , we have to assign an object o either to o_{p_1} or to o_{p_2} . This test can be accelerated by computing first the actual distance between o and the promoting object for which the filter distance is smaller. If the resulting exact distance is still smaller than the filter distance to the other promoting object, we can save on the second exact distance computation. Note, that we can easily alter the *NodeSplit* function in such a way that it returns the two resulting nodes instead of the promoting objects. For clarity, we only return the promoting objects.

10.3.5 The Caching M-tree

In this section, we present a further technique which helps to avoid costly distance computations for index construction and query processing.

Cache Based Construction. If we have to cope with distance computations which are more expensive than accessing a row on secondary storage, we suggest to use an additional table where we can save the already processed distance computations. This approach could easily be integrated into an ORDBMS as outlined in Chapter 3. Especially when splitting the same overflowing node repeatedly, accessing stored distance computation values can speed up the insertion process, since otherwise the same distances are computed several times.

Cache Based Range Queries. Efficient query processing of range queries also benefits from the idea of caching distance calculations. During the navigation through the M-tree directory, the same distance computations may have to be carried out several times. Although each object o is stored only once on the leaf level of the M-tree, it might be used several times as a routing object. Furthermore, we often have the situation that distance calculations carried out on the directory level have to be repeated at the leaf level.

As shown in Figure 137, a natural way to implement range queries is by means of recursion resulting in a depth-first search. We suggest to keep all distance computations in main memory which have been carried out on the way from the root to the actual node. After leaving the node, i.e. when exiting the recursive function, we de-

1	PROCEDURE <i>RangeSearch</i> (<i>N</i> Node, <i>q</i> QueryObject, ϵ Range);
7	DistCache(<i>N</i> , o_r , <i>q</i>) ;
16	DistCache(<i>N</i> , o_d , <i>q</i>) ;
22	END IF ;
22a	DeleteCache(<i>N</i>);
23	END ;

FUNCTION <i>DistCache</i> (<i>N</i> Node, o_1 Object, o_2 Object) RETURN float; BEGIN <i>result</i> := hashlookup.get(o_1 , o_2); IF <i>result</i> = nil THEN <i>result</i> := compute $d(o_1, o_2)$; hashlookup.add (<i>N</i> , o_1 , o_2 , <i>result</i>); END IF ; RETURN <i>result</i> ; END ;
--

PROCEDURE <i>DeleteCache</i> (<i>N</i> Node); BEGIN hashlookup.delete (<i>N</i>); END ;
--

Figure 147: Caching M-tree.

lete all distance computations carried out at this node. This limits the actual main memory footprint to $O(h \cdot b)$ where h denotes the maximum height of a tree and b denotes the maximum number of stored elements in a node. Even in multi-user environments this rather small worst-case main memory footprint is tolerable. The necessary adaptations of the *rangeSearch* algorithm are drafted in Figure 147.

For more details and optimizations suitable for the RM-tree, we refer the interested reader to [Sch 04].

10.4 Experimental Evaluation

In this section, we present the results of our extensive efficiency evaluation for different similarity models, different data sets and different query types. We processed similarity range queries as well as k -nearest neighbor queries for all five models described in Chapter 8 on our two different data sets *CAR* and *PLANE* (cf. Section 9.2.1) using ten representative database objects as query objects. For k -nearest neighbor queries we used the algorithm of Seidl and Kriegel (cf. Section 7.5.4)

```

SELECT attribute a1, ...,attribute ak
FROM DB
WHERE  $d_{f_1}(o,q) \leq \epsilon$  AND  $d_{f_2}(o,q) \leq \epsilon$  ... AND  $d_{f_n}(o,q) \leq \epsilon$  AND  $d_o(o,q) \leq \epsilon$ 

```

Figure 148: SQL-Statement for range queries.

and for similarity range queries the SQL-statement of Figure 148. By using hints we forced the query optimizer to evaluate the predicates in the WHERE-clause from left to right. As outlined in Section 7.2, we could easily encapsulate both query types into an extensible indexing framework. The Euclidean distance was used as distance measure between single-vector represented data. Distances between sets of feature vectors were computed using an implementation of the Kuhn-Munkres algorithm.

We conducted our experiments on top of the Oracle9i Server using PL/SQL for the computational main memory based programming. 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.

For all similarity queries, we took 90°-rotations and reflections into account, resulting in 48 permutations of the query object.

10.4.1 Single Vector Models: The NB-Tree

For the four single vector models - the *volume model*, the *solid-angle model*, the *eigen-value model* and the *cover sequence model* (with 7 covers) - we used the NB-tree as filter.

Similarity Range Queries. The average query times resulting from the different similarity range queries are depicted in Figure 149. As expected, the time needed for an exact distance computation is linearly dependent on the dimensionality of the feature vectors. For example, a full table scan on the 27-dimensional feature vectors resulting from the volume model is approximately 3 times faster than a full table scan on the 81-dimensional feature vectors of the eigen-value model. The NB-tree proves to be a suitable filter for all four single-vector models. The additional complexity of the filter is almost negligible. Even if the NB-tree does not filter out any candidates at all, the resulting query response times are only slightly higher than for the corresponding full table scans.

If we compare the average number of candidates after the filter step with the average number of results, the filter quality strongly varies for the different similarity models (cf. Figure 150). Data represented by the volume model and by the solid-angle model are filtered exceptionally good, especially for the *PLANE* data set (cf. Figure 150b and 150d) which contains some very large objects like aircraft wings besides a lot of small objects. The small objects are represented by only a few voxels and are contained entirely in only one histogram bin. This circumstance is also the cause for the bad effectiveness of these models (cf. Section 9.2.2). Our analysis yielded only 160 different feature vectors for the volume model respectively 1454 different vectors for the solid-angle model. Moreover, we also received the same number of different norm values for these two models, which indicates that the similarity is modelled by one-dimensional values without a noteworthy loss of quality.

For the eigen-value model, which produced 3510 different Euclidean norm values, the NB-tree also shows good filtering results, whereas the filter performance is relatively poor for the cover sequence model. A reason for the weaker filter performance of the NB-tree on the cover sequence model stems from the fact that this is the only model where feature values may be negative. The algebraic sign is irrelevant for a vector's Euclidean norm, as all feature values get squared during its computation. Thus, two cover sequence vectors with similarly sized cover extensions may have similar Euclidean norm values, although the cover positions of one feature vector have positive values and the positions of the other one are negative. On the other hand, different algebraic signs of the cover positions contribute to a greater Euclidean distance value between these two vectors. Thus, the difference between the Euclidean distance and the filter distance value is quite big resulting in a bad filter performance.

Similarity k -nn Queries. We conducted k -nearest neighbor queries for six different parameter values k , ranging from 1 to 100 for the *PLANE* data set and 1 to 50 for the *CAR* data set (cf. Figure 151 and 152). The average query times for single-step and multi-step k -nn queries are displayed in Figure 151. The query time for a filtered nearest neighbor search is directly dependent on the number of candidate objects for which an exact distance computation is necessary (cf. Figure 152). As for similarity range queries, the average filter quality of the NB-tree is best for data represented by the volume model and the solid-angle model. Furthermore, objects modelled by eigen values are also filtered well. Again, the worst filtering performance was ob-

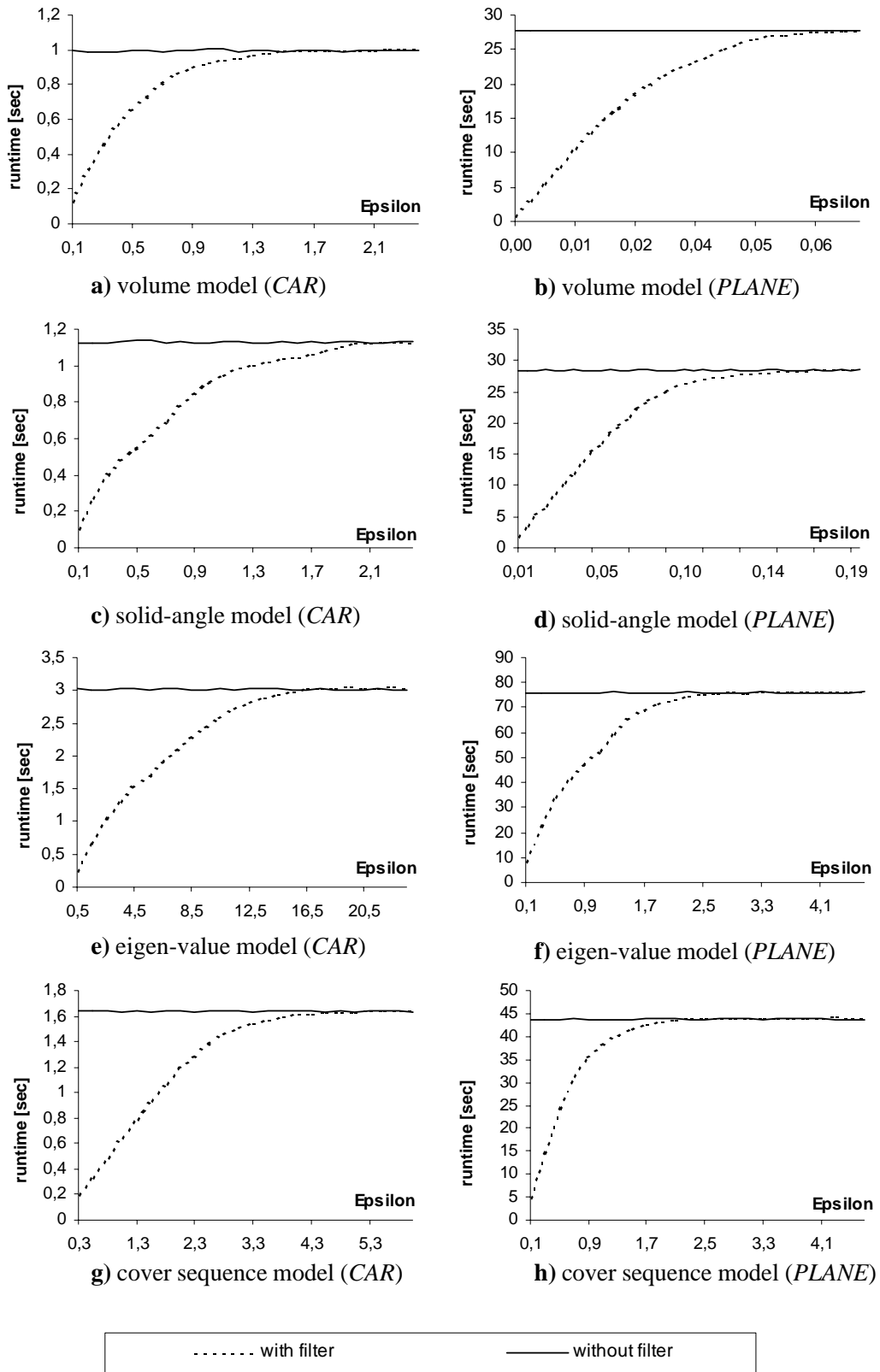


Figure 149: Query times of range queries on the single vector models.

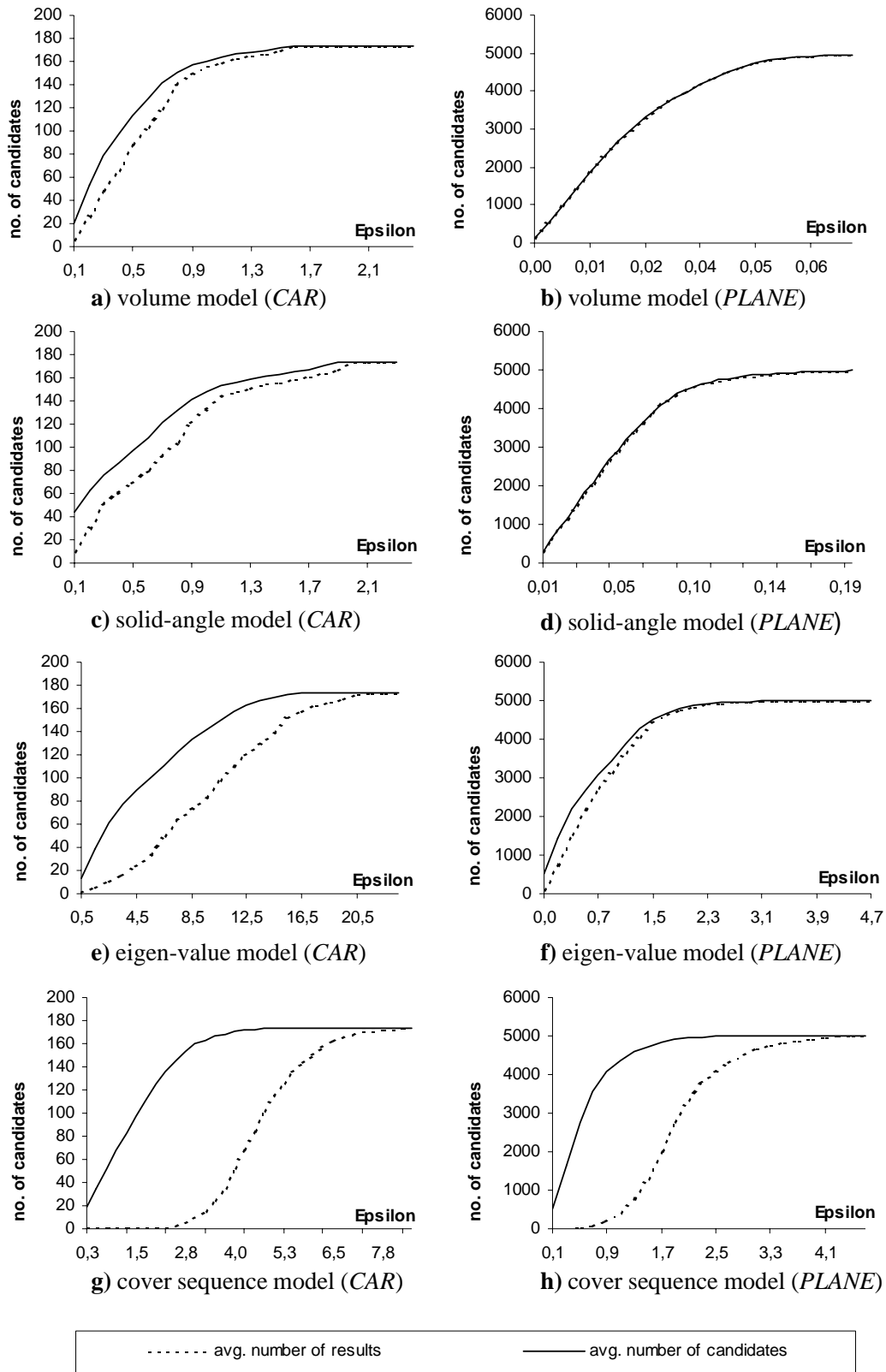


Figure 150: Number of candidates of range queries on the single vector models.

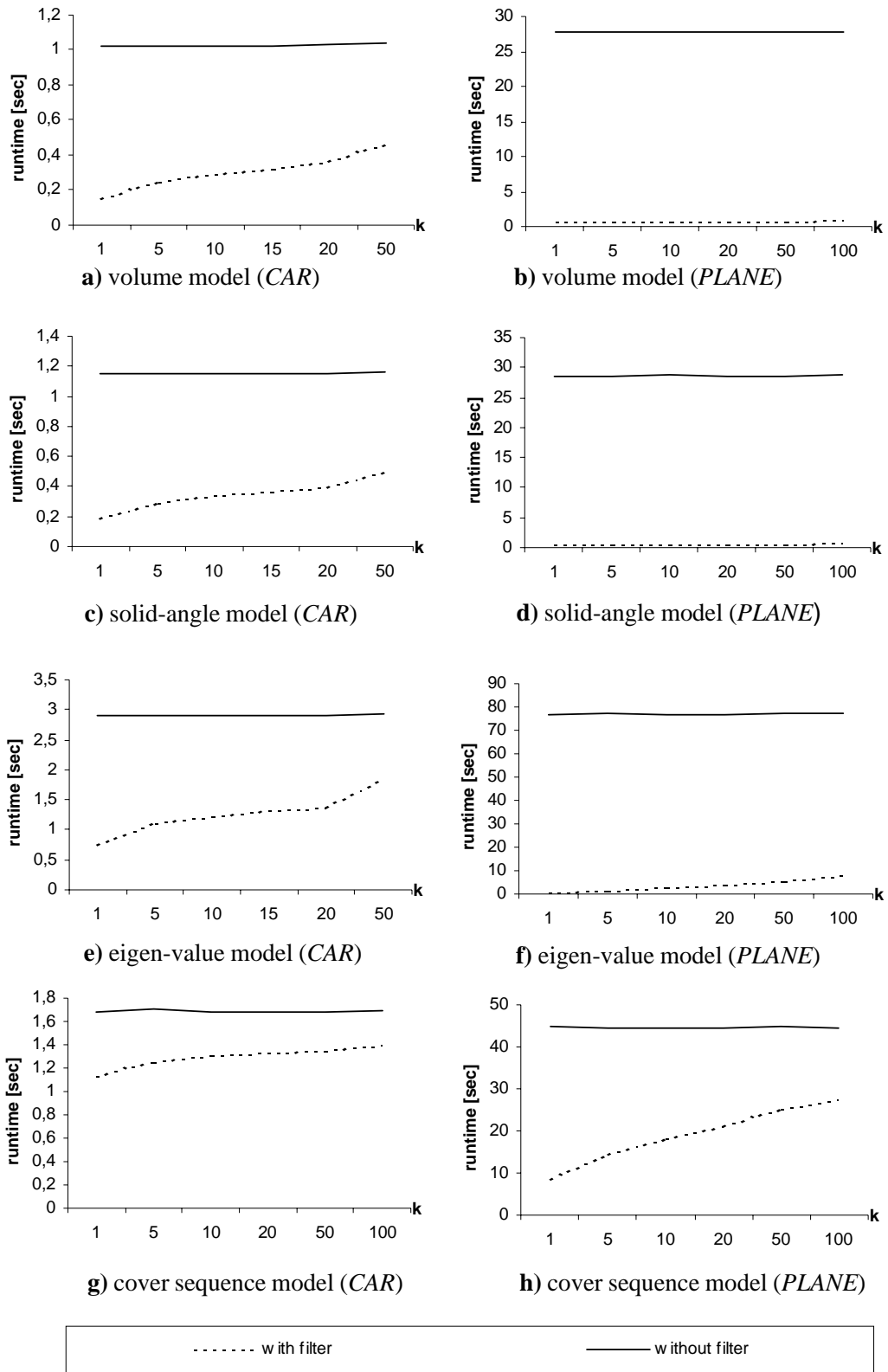


Figure 151: Query times of k -nn queries on the single vector models.

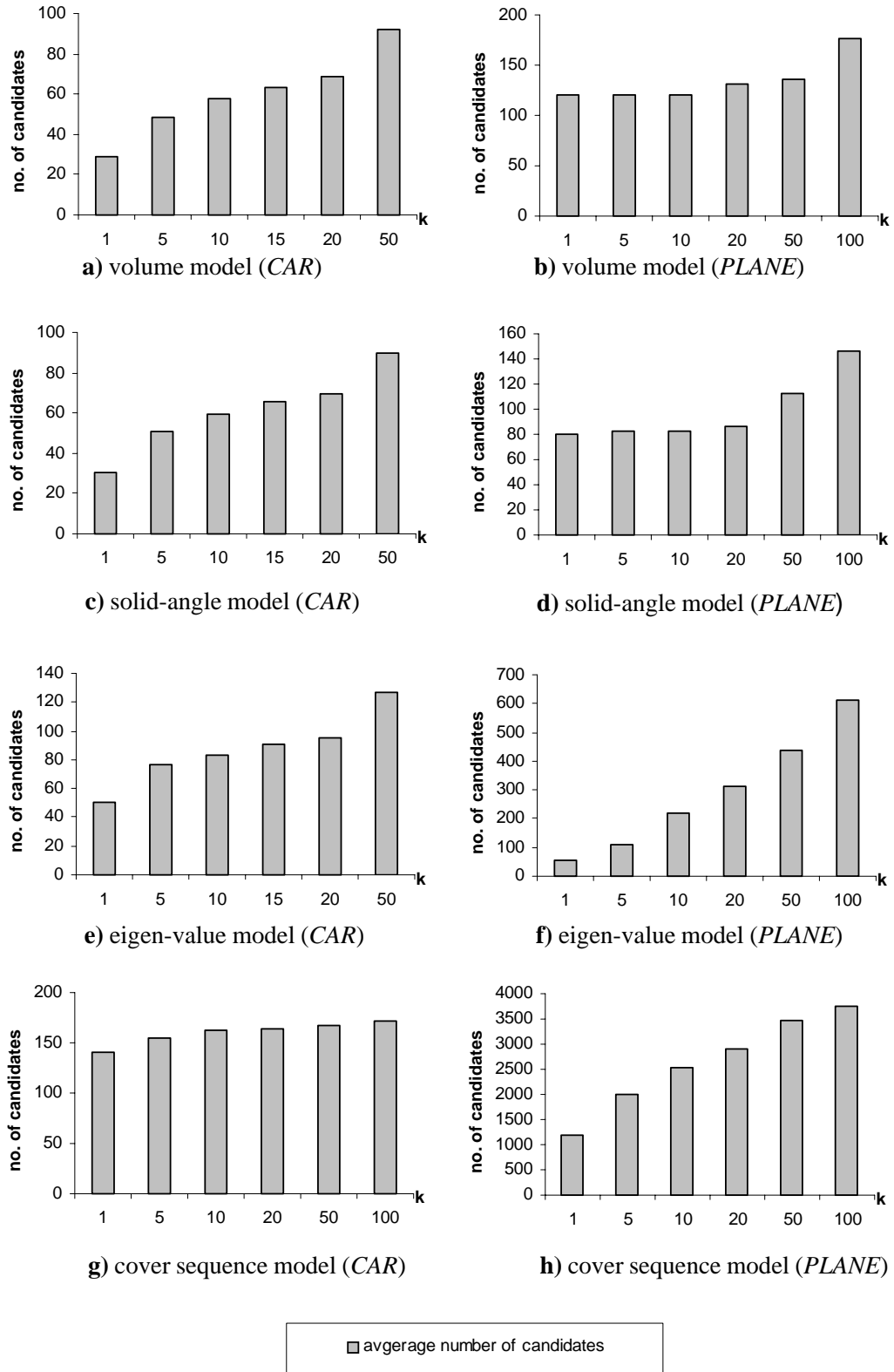


Figure 152: Number of candidates of k -nn queries on the single vector models.

served for data objects represented by cover sequences. The reasons for these results are the same as the ones described for similarity range queries.

Summary. This experimental evaluation shows that the runtime for similarity queries based on a full table scan, depends linearly on the dimensionality of the used feature vectors. Similarity queries based on the eigen-value model are almost three times slower than for the volume-model or the solid-angle model. Furthermore, the experiments show, that the NB-tree is a very effective way to accelerate similarity queries. Especially the space-partitioning models benefit from the NB-tree, which can easily be integrated into any ORDBMS. For instance, 1-nn queries for the eigen-value model can be accelerated by two orders of magnitude when using the NB-tree. Although the NB-tree also accelerates similarity queries based on the cover sequence model, the benefits are less pronounced. This worse filter behavior is due to the negative feature values which might occur in the feature vectors of the cover sequence model. The NB-tree does not take the algebraic sign into account, so that equally sized covers located differently might be treated similar by the NB-tree which results in a poor filter performance.

In [Lei 04] a lot more experiments were carried out which pinpoint the differences of the query response behavior of similarity queries on our different similarity models for CAD data. In this section, we only presented a meaningful summary of these tests. For more details, we refer the interested reader to [Lei 04].

10.4.2 Vector Set Model: Filters for the Minimal Matching Distance

In this section, we investigate the runtime and selectivity of the three presented filters for vector sets, i.e. the centroid filter, the Euclidean norm filter and the closest-pair filter. Furthermore, we present the results from a combined filter, which always computes the maximum from the Euclidean norm and the centroid approach. Both, the filter steps and the exact distance computations were based on a sequential scan.

Similarity Range Queries. In a first experiment, we carried out different range queries on a vector set consisting of 3, 5 and 7 6-dimensional points. Figure 153 shows that the selectivity of the closest-pair filter is almost optimal, i.e. almost no unnecessary candidates are produced. Nevertheless, the overall runtime of this filter-step is very high, as the runtime complexity of the closest-pair filter is almost as high as the computation of the minimal matching distance itself (cf. Figure 154).

Although less selective than the closest-pair filter, all the other filters accelerate the query process more profoundly, especially for sufficiently small values of the range parameter ε . Generally spoken, queries using the combined filter are the fastest ones. The combined filter performs only slightly better than the centroid approach but significantly better than the Euclidean norm filter. However, when high ε -values are used, i.e. only a few or no data objects could be excluded by the filter step, the Euclidean norm vector performs slightly superior to the other filters and is nearly as fast as a full table scan. This is due to the fact that the filter distance between norm vectors is cheaper to compute than the Euclidean distance between centroids, because rotation and reflection of the query object do not need to be taken into account (cf. remark 2 in Section 10.1).

We observed the highest performance gains of combined filters for very small values of ε and data sets consisting only of three covers (cf. Figure 154a and 154b). Using the Euclidean norm filter prior to the centroid filter proves to be more gainful for combined filters than vice versa. Again, this is due to the low computation cost of the Manhattan distance between the norm vectors. However, these differences could only be observed for small values of ε vanishing for higher values.

Similarity k -nn Queries. We conducted k -nearest neighbor queries for six different parameter values k , ranging from 1 to 100 for the *PLANE* data set and ranging from 1 to 50 for the *CAR* data set (cf. Figure 155 and Figure 156). The average query times of these nearest neighbor queries based on a full table scan are given in the Table 11:

data set	average runtime [sec.] (3 covers)	average runtime [sec.] (5 covers)	average runtime [sec.] (7 covers)
CAR	8.3	19.7	40.1
PLANE	236.1	539.6	1014.6

Table 11: Query response times of k -nn queries based on a full table scan for the vector set model.

The results of the k -nearest neighbor queries based on the multi-step query processing paradigm are quite similar to the ones for similarity range queries. Again, the closest-pair filter produces the smallest number of candidates, and the centroid filter proved to be superior to the Euclidean norm filter (cf. Figure 155). A combination of

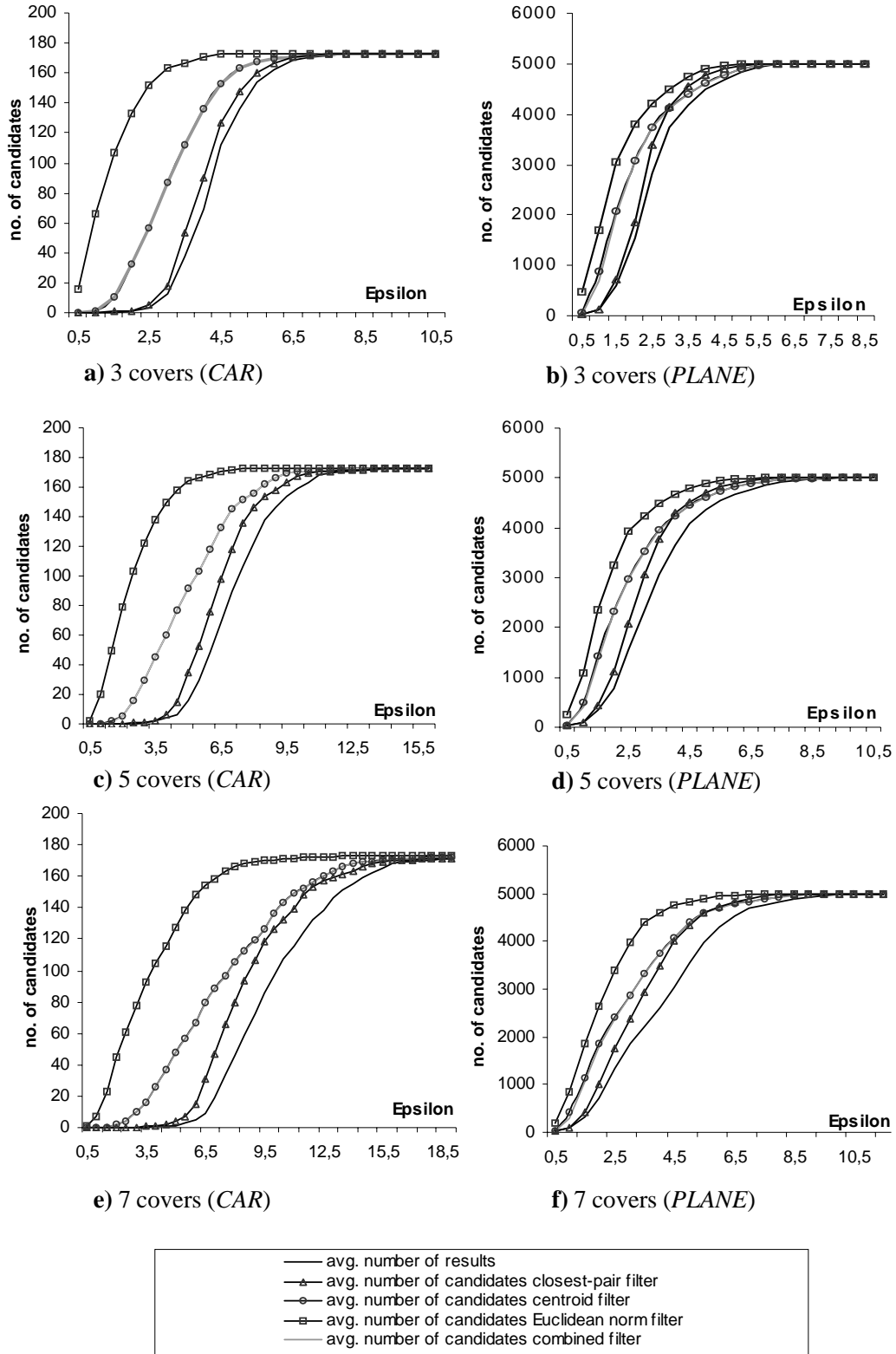


Figure 153: Number of candidates of range queries on the vector set model.

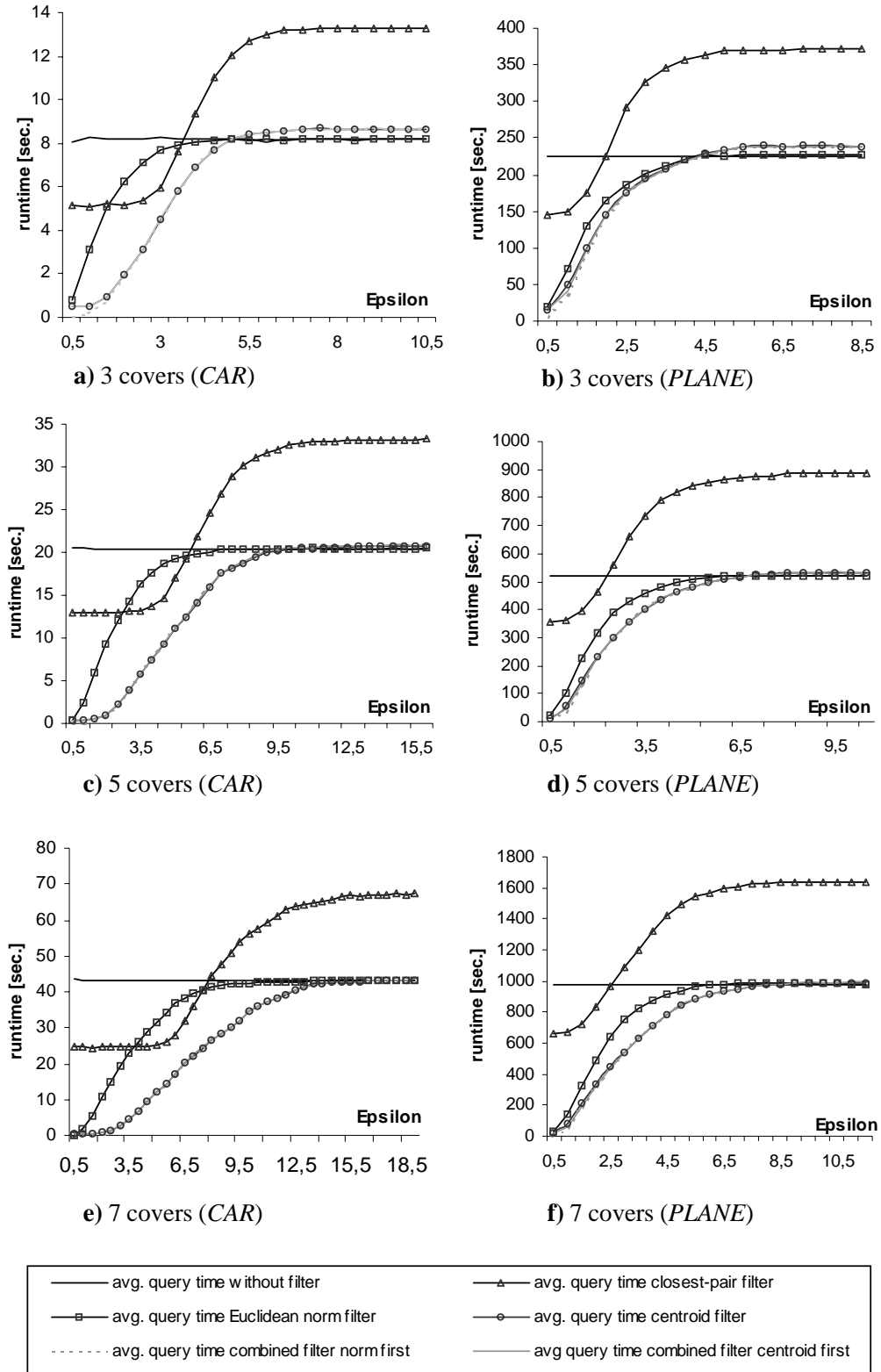


Figure 154: Query times of range queries on the vector set model.

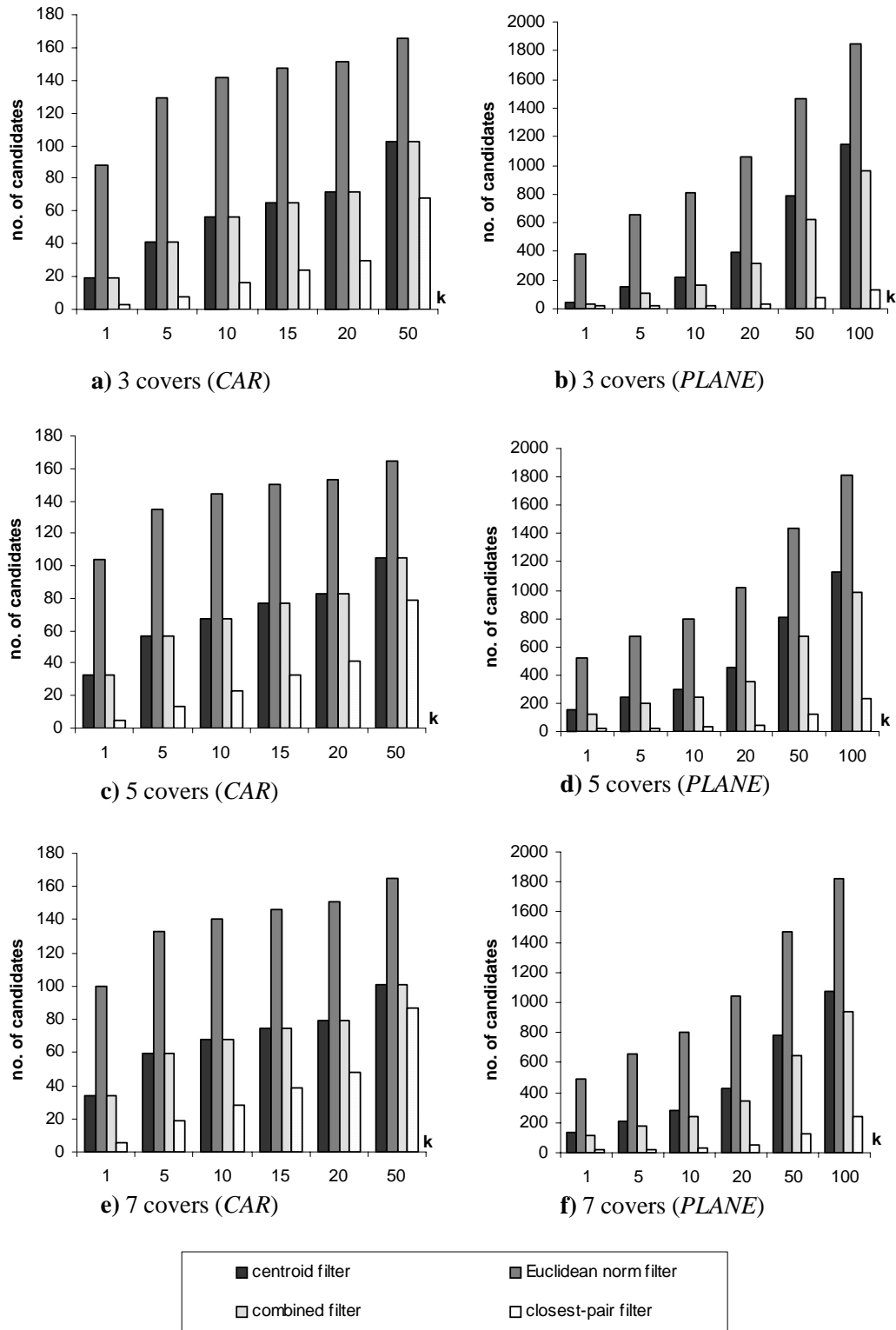


Figure 155: Number of candidates of k -nn queries on the vector set model.

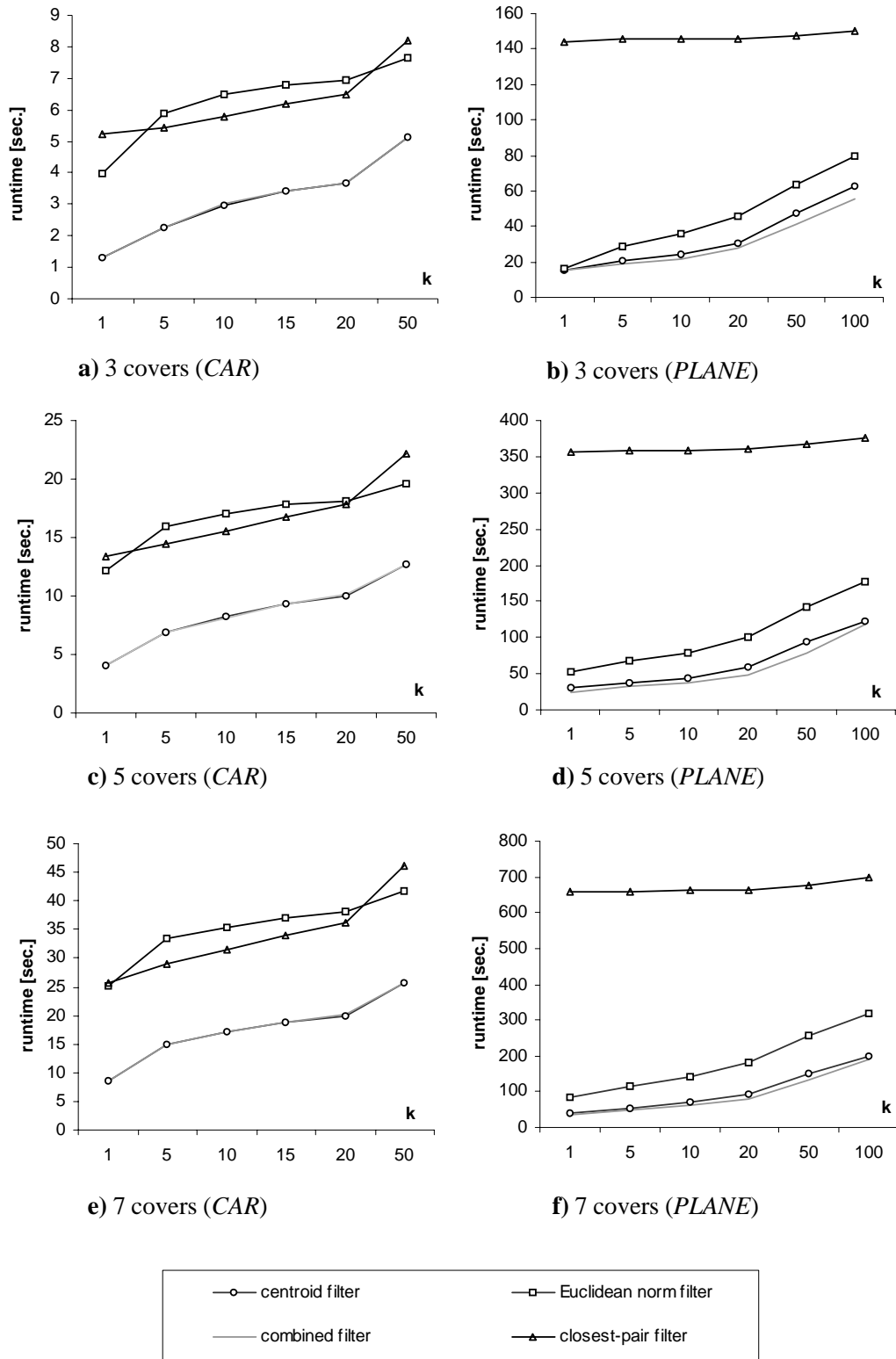


Figure 156: Query times of k -nn queries on the vector set model.

the latter two filters, as described in Section 10.2.4 has a weaker filtering quality than the closest-pair filter, but produces less candidates than each filter on its own. For instance on the *PLANE* data set, the average number of candidates is 17% lower for the combined filter than for the centroid filter and 61% lower than for the Euclidean norm vector.

Due to the high computation cost of the closest-pair distance, k -nearest neighbor queries using this filter are much slower on average than queries employing one of the other filter types (cf. Figure 156). However, they are still up to five times faster than the corresponding full table scans. Similarity queries employing the Euclidean norm vector as a filter are much slower than queries using the centroid or the combined filter. For small values of the parameter k , the centroid and combined filter are more than 25 times faster than corresponding full table scans. On the *CAR* data set, query times for these two filter types are nearly identical. On the *PLANE* data set, queries employing the combined filter were more than 11% faster than corresponding queries using the centroid filter.

Summary. For all types of queries, the closest pair filter was the most selective filter. Nevertheless, this filter does not pay off, as it is rather expensive itself. Let us note, that for the much more expensive partial similarity search on vector sets, this filter is both the most selective one and the most efficient one [BKP 04a]. The combination between the Euclidean norm filter and the centroid filter is the best choice for efficient similarity search on CAD objects modelled by vector sets. The combination of these two filters can very efficiently be computed and is still selective enough to accelerate similarity queries on vector sets by up to two orders of magnitude.

10.4.3 Vector Set Model: M-tree

We also tested a cartridge implementation of the M-tree managing vector set represented data. In all our tests, we used an optimal M-tree fanout of 19. For carrying out similarity k -nn queries, we used a slightly adapted version of the algorithm presented in [HS 95] (cf. Section 7.4.3). For more details we refer the reader to [KKPR 04d].

Furthermore, we integrated the optimizations, introduced in Section 10.3, into this cartridge implementation. In this section, we present a meaningful summary of a detailed experimental evaluation carried out on the optimized Relational M-tree. For more details, we refer the interested reader to [Sch 04].

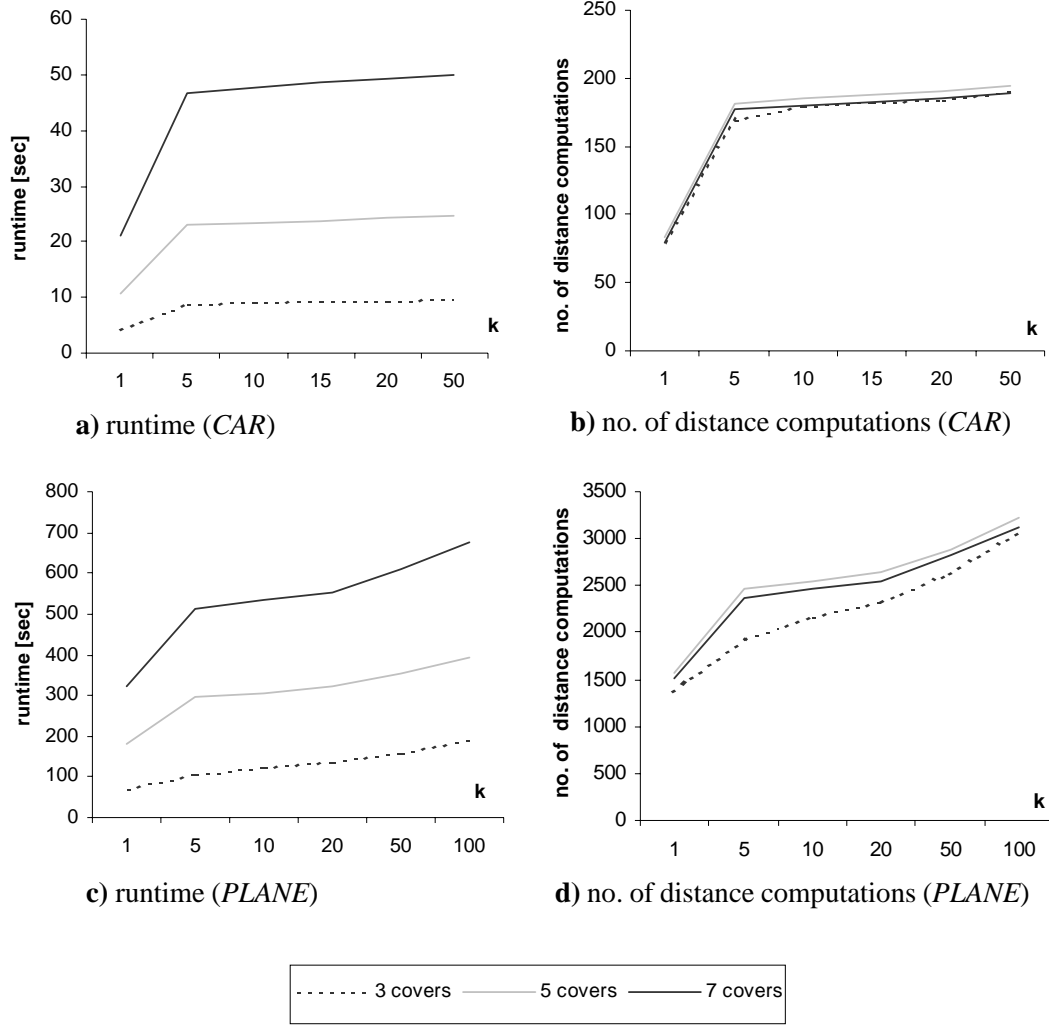


Figure 157: k -nn queries on the vector set model (RM-tree).

Relational M-tree. Figure 157 and Figure 158 depict the results of similarity queries carried out on the cartridge implementation of the M-tree. Comparing these two figures with the figures presented in the foregoing section, we can see that the M-tree outperforms the sequential scan and the closest-pair filter for some values of ϵ . Nevertheless, in all our experiments, the M-tree was considerably slower than the combination between the Euclidean norm filter and the centroid filter. The rather bad performance of the M-tree is due to the overhead involved in the cartridge implementation and due to the high amount of distance computations which are necessary even for small values of k and ϵ .

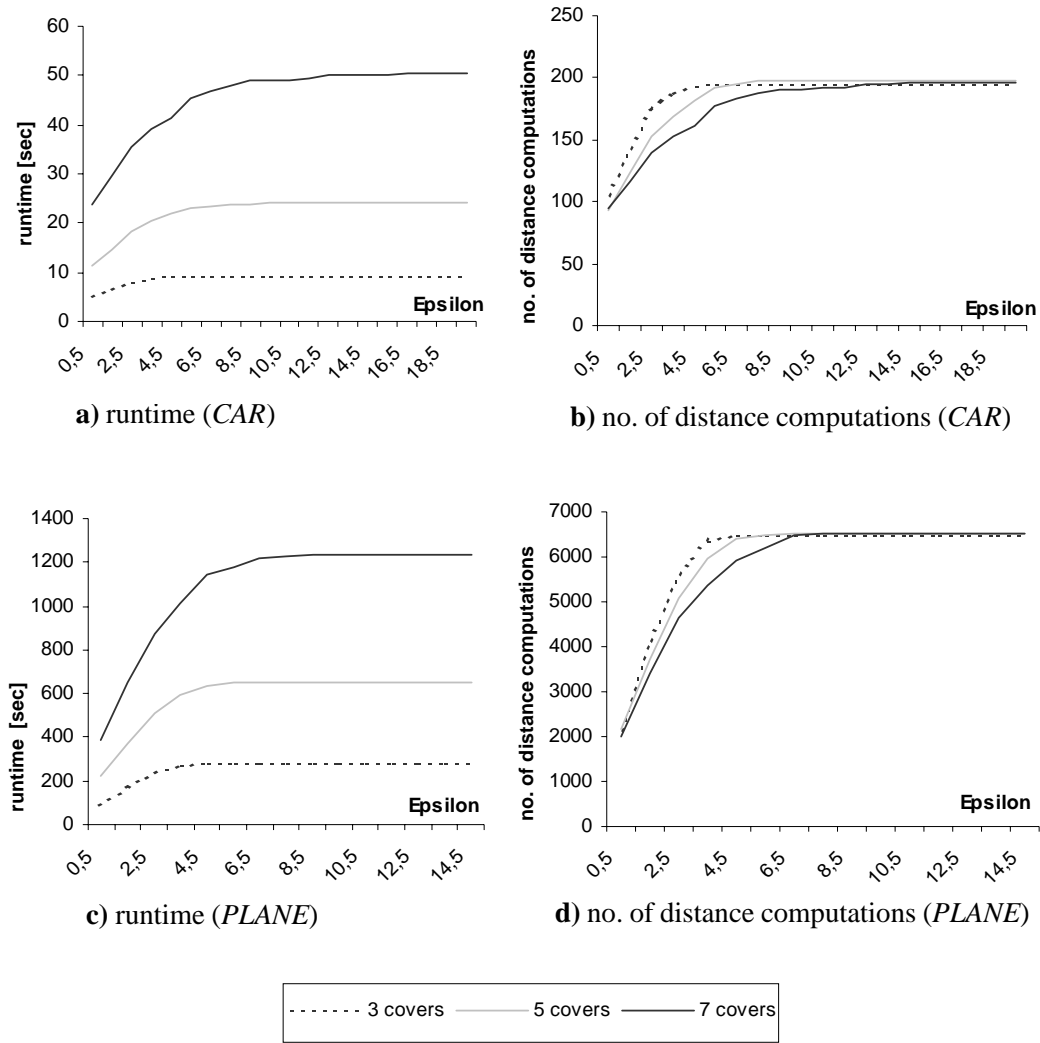


Figure 158: Range queries on the vector set model (RM-tree).

Let us note that similar observations can be made for other metric data spaces for which there exist suitable filter steps, e.g. graphs [KS 03] and trees [KKSS 04]. The M-tree seems to be the method of choice only for metric data spaces for which there exist no suitable filters.

Optimized Relational M-tree. In this section, we will investigate how the optimizations introduced in Section 10.3 accelerate the Relational M-tree. Thereby we will concentrate on range queries carried out on the *PLANE* data set as they form the foundation of density-based clustering [EKSX 96], and, furthermore, k -nn queries can efficiently be carried out based on range-queries [LS 02].

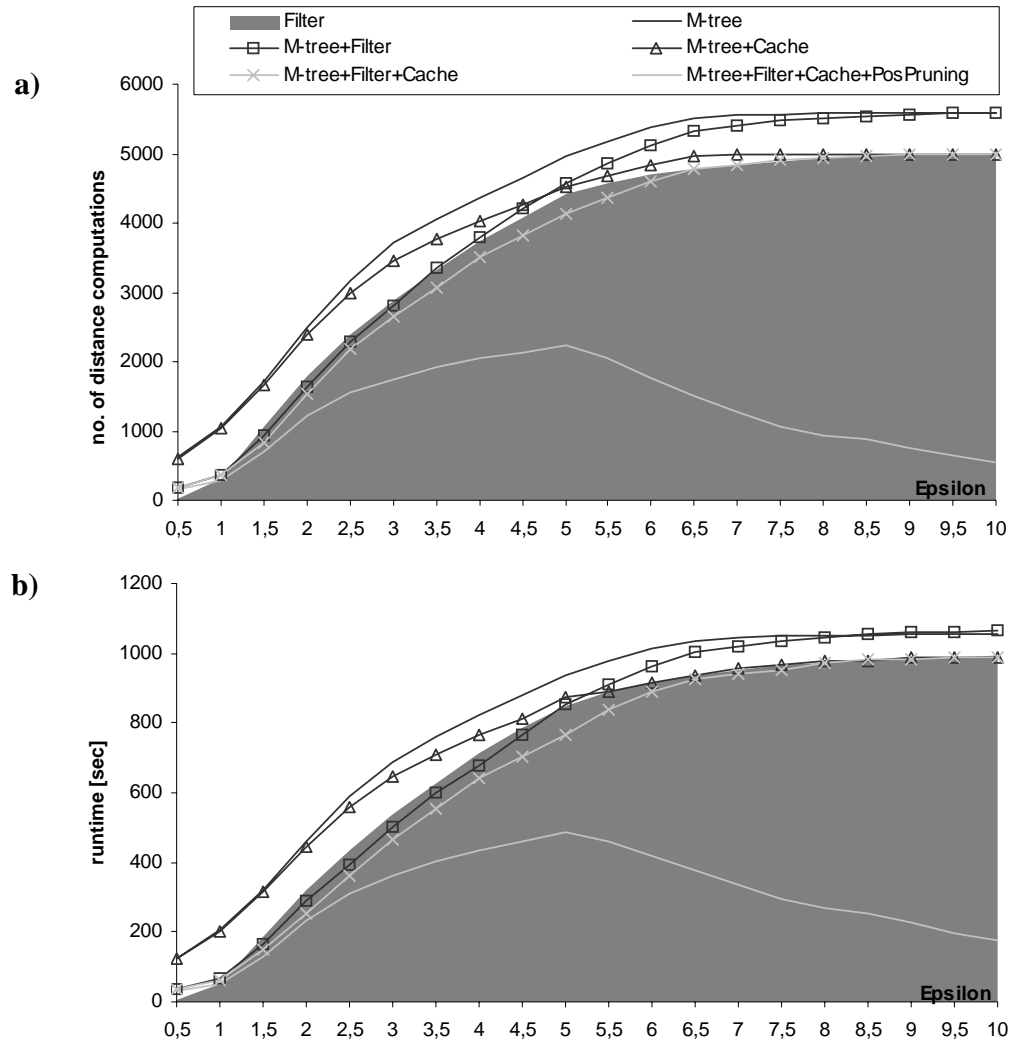


Figure 159: Range queries on the vector set model (7 covers) (optimized RM-tree).
(PLANE data set)

a) Number of distance computations, **b)** Runtime

Range Queries. In Figure 159 and Figure 160 the optimized Relational M-tree is compared to the original RM-tree and to the best possible filter. First, the figures show clearly that the used filter, i.e. the combination of the Euclidean norm filter and the centroid filter, outperforms the RM-tree for all values of ϵ .

The combination of filtering and metric indexing, i.e. the Filtering M-tree (cf. Section 10.3.4), outperforms both the best possible filter and the Relational M-tree for ϵ -values interesting for density-based clustering, e.g. values around 2 for the data set of Figure 159.

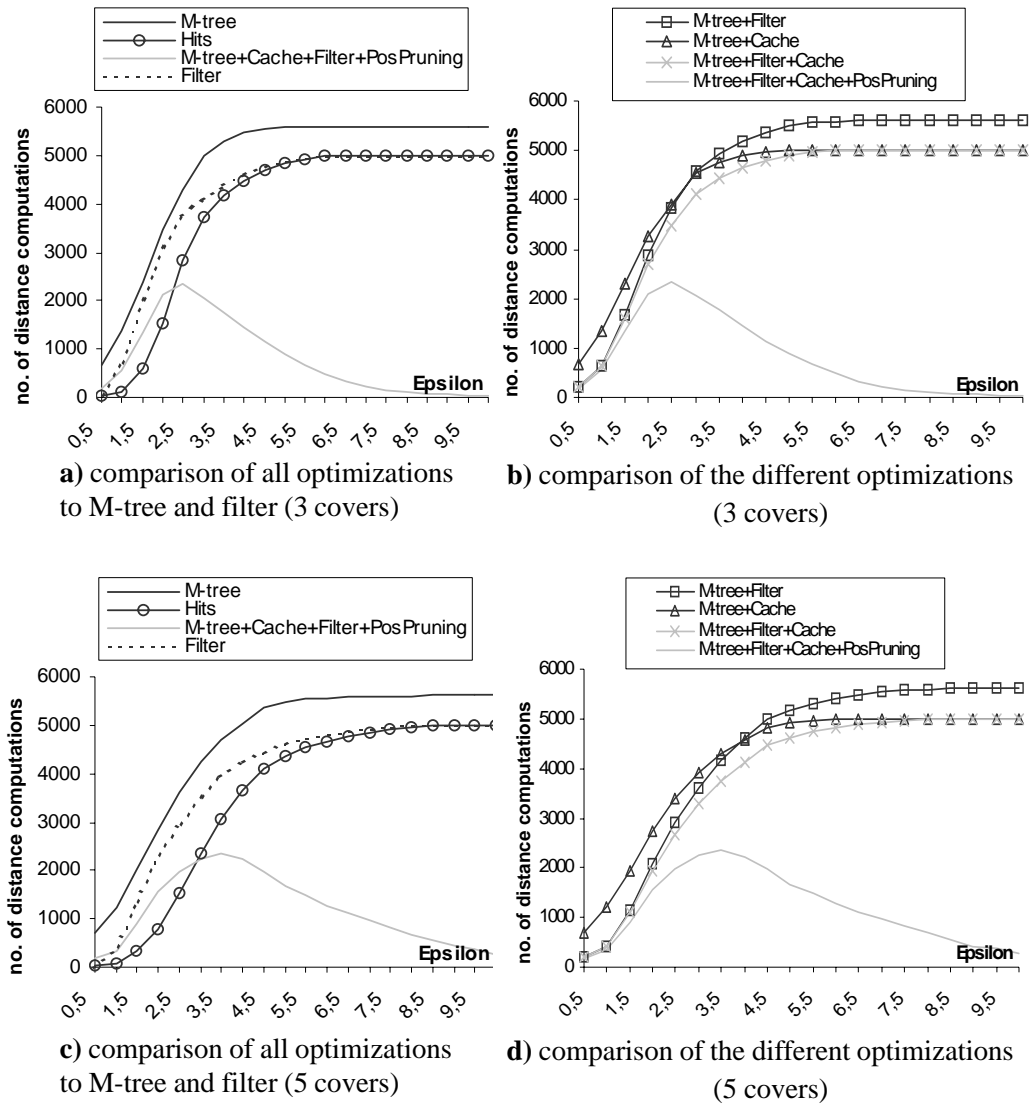


Figure 160: Range queries on the vector set model (optimized RM-tree).

The Caching M-tree (cf. Section 10.3.5) leads to a further improvement of the M-tree. All distance computations carried out at directory nodes are memorized. Therefore, for high ϵ -values the Caching M-tree does not need more distance computations than the sequential scan.

The Relational M-tree using filter and caching techniques considerably accelerates both the filter and the Relational M-tree. It is the method of choice for range queries, if the boolean information whether an object is included in an ϵ -range of a

given query object is not enough. If a database object o is within the ε -range of a given query object q , this approach also computes the exact distance between o and q . The density-based clustering algorithm OPTICS, for instance, is based on this additional information.

On the other hand, if we are satisfied with the boolean information whether an object is included in the ε -range of a given query object, we can exploit the pre-clustering of the index structure not only for the negative pruning but also for the positive one. If a directory node is completely covered by a given ε -range all objects of the corresponding subtree can be reported without carrying out any additional distance computations (cf. Section 10.3.3). Taking all these optimizations together, we can efficiently carry out range queries for all ε -values. The density-based clustering algorithm DBSCAN, for instance, does benefit from all of these optimizations and can considerably be accelerated by the introduced techniques [KKPS 04].

Creation of M-tree. The generation of the optimized M-tree was carried out without caching (cf. Figure 161a) and with caching (cf. Figure 161b). Without caching, the number of necessary distance calculations is very high, due to the repeated splitting of nodes. Note that the number of distance calculations for one node split is quadratic w.r.t. the number of elements of this node. In this case, our *NodeSplit* function (cf. Figure 146) only needs 1/4 of the distance calculations while still producing the same M-tree. If we apply caching, the overall number of required distance computations is much smaller as many distance computations necessary for splitting a node can be fetched from disk. In this case our *FindSubTree* function (cf. Figure 145) allows us to reduce the number of required distance calculations even further, i.e. the number of distance computations is bisected. To sum up, both optimizations, which are based on the exploitation of available filter information, allow us to build up an M-tree much more efficiently.

10.5 Summary

In this chapter, we presented a detailed efficiency evaluation of our similarity models. Thereby, we concentrated on efficient query processing for vector set data. We introduced three different filters for the minimal matching distance which computes the distance between two vector sets: the *centroid filter*, the *Euclidean norm filter* and the *closest pair filter*. We showed that the combination of the rather cheap

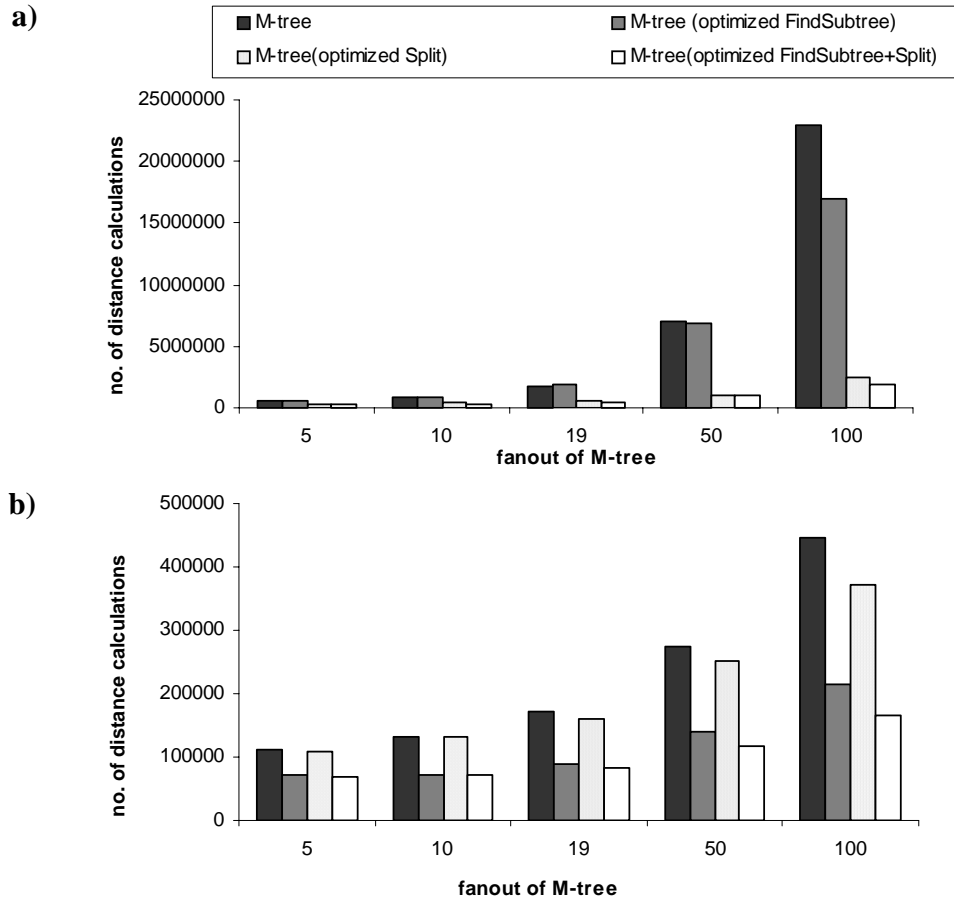


Figure 161: Creation of a Relational M-tree.

a) Without caching distance calculations, **b)** With caching distance calculations

Euclidean norm filter and the centroid filter is the most efficient filter for the minimal matching distance although it is less selective than the closest pair filter.

Furthermore, we introduced suitable optimizations for the Relational M-tree, i.e. the *Scanning M-tree*, the *Filtering M-tree* and the *Caching M-tree*. We showed that although the Relational M-tree itself is clearly outperformed by the presented filters, the optimized Relational M-tree is the method of choice for efficiently carrying out range queries.

Chapter 11

BOSS: Browsing Optics-Plots for Similarity Search

Similarity search in database systems is becoming an increasingly important task in modern application domains such as multimedia, molecular biology, medical imaging, computer aided engineering, marketing and purchasing assistance as well as many others. In this chapter, we show how visualizing the hierarchical clustering structure of a database of objects can aid the user in his time consuming task to find similar objects. We present related work and explain its shortcomings which led to the development of our new methods. Based on reachability plots, we introduce approaches which automatically extract the significant clusters in a hierarchical cluster representation along with suitable cluster representatives. These techniques can be used as a basis for visual data mining. We implemented our algorithms resulting in an industrial prototype which we used for the experimental evaluation. This evaluation is based on real world test data sets and points out that our new approaches to automatic cluster recognition and extraction of cluster representatives create meaningful and useful results in comparatively short time.

11.1 Introduction

In the last ten years, an increasing number of database applications has emerged for which efficient and effective support for similarity search is substantial. The importance of similarity search grows in many different application areas. Particularly, the task of finding similar shapes in 2D and 3D becomes more and more important.

Hierarchical clustering was shown to be effective for evaluating similarity models (cf. Section 9.1). Especially, the reachability plot generated by OPTICS [ABKS 99] is suitable for assessing the quality of a similarity model. Furthermore, visually analyzing cluster hierarchies helps the user, e.g. an engineer, to find and group similar objects. Solid cluster extraction and meaningful cluster representatives form the foundation for providing the user with significant and quick information.

In this chapter, we introduce algorithms for automatically detecting hierarchical clusters along with their corresponding representatives. In order to evaluate our ideas, we developed a prototype called *BOSS* (Browsing *OPTICS*-Plots for Similarity Search). *BOSS* is based on techniques related to *visual data mining*. It helps to visually analyze cluster hierarchies by providing meaningful cluster representatives.

To sum up, the main contributions of this chapter are as follows:

- We explain how different important application ranges would benefit from a tool which allows visually mining through cluster hierarchies.
- We reason why the hierarchical clustering algorithm OPTICS forms a suitable foundation for such a browsing tool.
- We introduce a new cluster recognition algorithm for the reachability plots generated by OPTICS. This algorithm generalizes all the other known cluster recognition algorithms for reachability plots. Although our new algorithm does not need a sophisticated and extensive parameter setting, it outperforms the other cluster recognition algorithms with respect to the quality and number of recognized clusters and subclusters.
- We tackle the complex problem of finding suitable cluster representatives. We introduce two new approaches and show that they yield more intuitive representatives than the known medoid approach.

- We describe a new browsing tool which comprises algorithms for cluster recognition and representation. This tool, called BOSS, is based on a client-server architecture which allows the user to get a quick and meaningful overview over large data sets.

The remainder of the chapter is organized as follows. In Section 11.2, we first look at hierarchical clustering with a special emphasize on the advantages of OPTICS. Furthermore, we discuss the requirements and application areas in the industrial and in the scientific community which motivated the development of BOSS. In Section 11.3 and Section 11.4, we introduce suitable algorithms for cluster recognition and cluster representatives. In Section 11.5, we describe the actual industrial prototype we developed and used it, in Section 11.6, to evaluate our new cluster recognition and representation algorithms. In Section 11.7, the chapter concludes with a short summary.

11.2 Hierarchical Clustering

In this section, we outline the application ranges which led to the development of our interactive browsing tool, called BOSS (cf. Section 11.2.2). In order to understand the connection between BOSS and the application requirements, we first recall the major concepts of the hierarchical clustering algorithm OPTICS as introduced in Section 9.1.2.

11.2.1 Major Advantages of OPTICS

As already mentioned in Section 9.1.2, one of the main advantages of OPTICS is that the cluster structure can be visualized through so called reachability plots which are 2D plots. They are generated as follows: the clustered objects are ordered along the x-axis according to the cluster ordering computed by OPTICS and the reachabilities assigned to each object are plotted along the abscissa. An example reachability plot is depicted in Figure 122. Valleys in this plot indicate clusters: objects having a small reachability value are closer and thus more similar to their predecessor objects than objects having a higher reachability value. The reachability plots computed by OPTICS help the user to get a meaningful and quick overview over a large data set. Instead of a dendrogram, which is the common representation of hierarchical cluster-

ings, the resulting reachability plot is much easier to analyze (cf. Section 9.1.2). Further reasons why we chose OPTICS as a foundation of BOSS are the following:

- OPTICS is - in contrast to most other algorithms - relatively insensitive to its two input parameters, ϵ and *MinPts*. The authors in [ABKS 99] state that the input parameters just have to be large enough to produce good results.
- OPTICS is a hierarchical clustering method which yields more information about the cluster structure than a method that computes a flat partitioning of the data (e.g. *k*-means [McQ 67]).
- There exists a very efficient variant of the OPTICS algorithm which is based on a sophisticated data compression technique called “Data Bubbles” [BKKS 01], where we have to trade only very little quality of the clustering result for a great increase in performance.
- There exists an efficient incremental version of the OPTICS algorithm [KKG 03].

11.2.2 Application Ranges

BOSS was designed for three different purposes: visual data mining, similarity search and evaluation of similarity models. For the first two applications, the choice of the representative objects of a cluster is the key step. It helps the user to get a meaningful and quick overview over a large existing data set. Furthermore, BOSS helps scientists to evaluate new similarity models.

Visual Data Mining. As defined in [Ank 00], visual data mining is a step in the KDD process that utilizes visualization as a communication channel between the computer and the user to produce novel and interpretable patterns. Based on the balance and sequence of the automatic and the interactive (visual) part of the KDD process, three classes of visual data mining can be identified.

- Visualization of the data mining result:

An algorithm extracts patterns from the data. These patterns are visualized to make them interpretable. Based on the visualization, the user may want to return to the data mining algorithm and run it again with different input parameters (cf. Figure 162a).

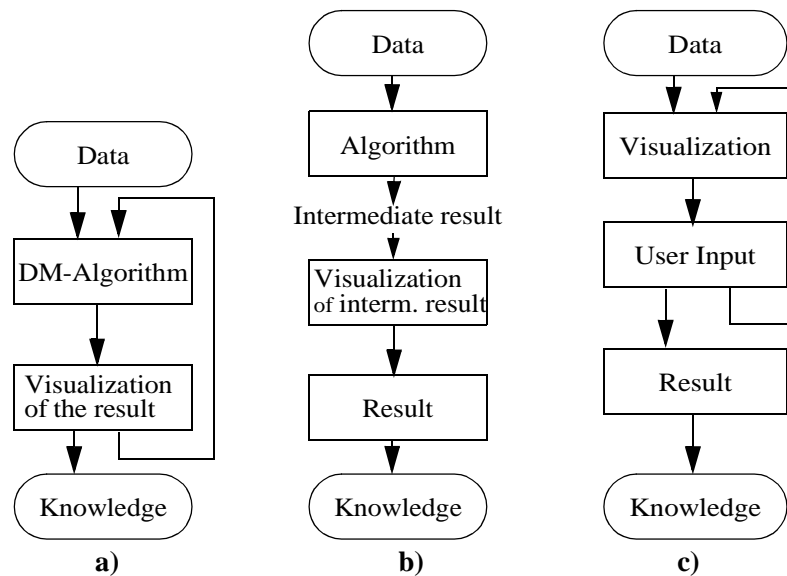


Figure 162: Different approaches to visual data mining.

- Visualization of an intermediate result:

An algorithm performs an analysis of the data not producing the final patterns but an intermediate result which can be visualized. Then the user retrieves the interesting patterns in the visualization of the intermediate result (cf. Figure 162b).

- Visualization of the data:

Data is visualized immediately without running a sophisticated algorithm before. Patterns are obtained by the user by exploring the visualized data (cf. Figure 162c).

The approach presented in this chapter belongs to the second class. A hierarchical clustering algorithm is applied to the data, which extracts the clustering structure as an intermediate result. There is no meaning associated with the generated clusters. However, our approach allows the user to visually analyze the contents of the clusters. The clustering algorithm used in the algorithmic part is independent from an application. It performs the core part of the data mining process and its result serves as a multi-purpose basis for further analysis directed by the user. This way the user may obtain novel information which was not even known to exist in the data set. This is in contrast to similarity search where the user is restricted to find similar parts respective to a query object and a predetermined similarity measure.

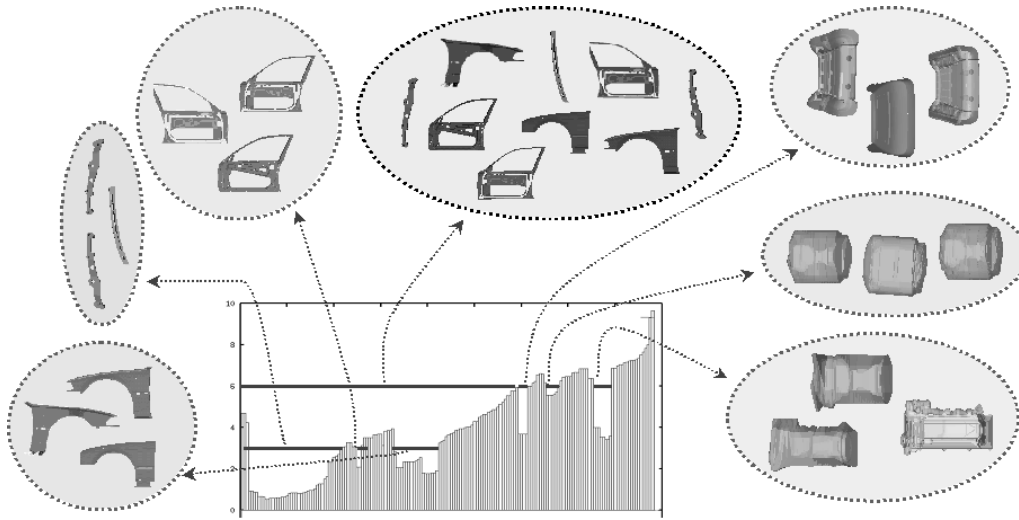


Figure 163: Browsing through reachability plots with different thresholds ϵ_{cut} .

Similarity Search. The development, design, manufacturing and maintenance of modern engineering products is a very expensive and complex task. Effective similarity models are required for two- and three-dimensional CAD applications to cope with rapidly growing amounts of data. Shorter product cycles and a greater diversity of models are becoming decisive competitive factors in the hard-fought automobile and aircraft market. These demands can only be met if the engineers have an overview of already existing CAD parts. It would be desirable to have an interactive data browsing tool which depicts the reachability plot computed by OPTICS in a user friendly way together with appropriate representatives of the clusters. This clear illustration would support the user in his time-consuming task to find similar parts. From the industrial user's point of view, this browsing tool should meet the following two requirements:

- The hierarchical clustering structure of the data set is revealed at a glance. The reachability plot is an intuitive visualization of the clustering hierarchy which helps to assign each object to its corresponding cluster or to noise. Furthermore, the hierarchical representation of the clusters using the reachability plot helps the user to get a quick overview over all clusters and their relation to each other. As each entry in the reachability plot is assigned to one object, we can easily illustrate some representatives of the clusters belonging to the current density threshold ϵ_{cut} (cf. Figure 163).

- The user is not only interested in the shape and the number of the clusters, but also in the specific parts building up a cluster. As for large clusters it is rather difficult to depict all objects, representatives of each cluster should be displayed. To follow up a first idea, these representatives could be simply constructed by superimposing all parts belonging to the regarded cluster (cf. Figure 164). We can browse through the hierarchy of the representatives in the same way as through the OPTICS plots.

This way, the cost of developing and producing new parts could be reduced by maximizing the reuse of existing parts, because the user can browse through the hierarchical structure of the clusters in a top-down way. Thus the engineers get an overview of already existing parts and are able to navigate their way through the diversity of existing variants of products, such as cars.

Evaluation of Similarity Models. In general, similarity models can be evaluated by computing k -nearest neighbor queries (k -nn queries). As shown in Section 9.1, this evaluation approach is subjective and error-prone because the quality measure of the similarity model depends on the results of a few similarity queries and, therefore, on the choice of the query objects. A model may perfectly reflect the intuitive similarity according to the chosen query objects and would be evaluated as “good” although it produces disastrous results for other query objects.

A better way to evaluate and compare several similarity models is to apply a clustering algorithm. Clustering groups a set of objects into classes where objects within one class are similar and objects of different classes are dissimilar to each other. The result can be used to evaluate which model is best suited for which kind of objects. It is more objective since each object of the data set is taken into account to evaluate the data models.

11.3 Cluster Recognition

In this section, we address the first task of automatically extracting clusters from the reachability plots. After a brief discussion of recent work in that area, we propose a new approach for hierarchical cluster recognition based on reachability plots, called *gradient-clustering*.

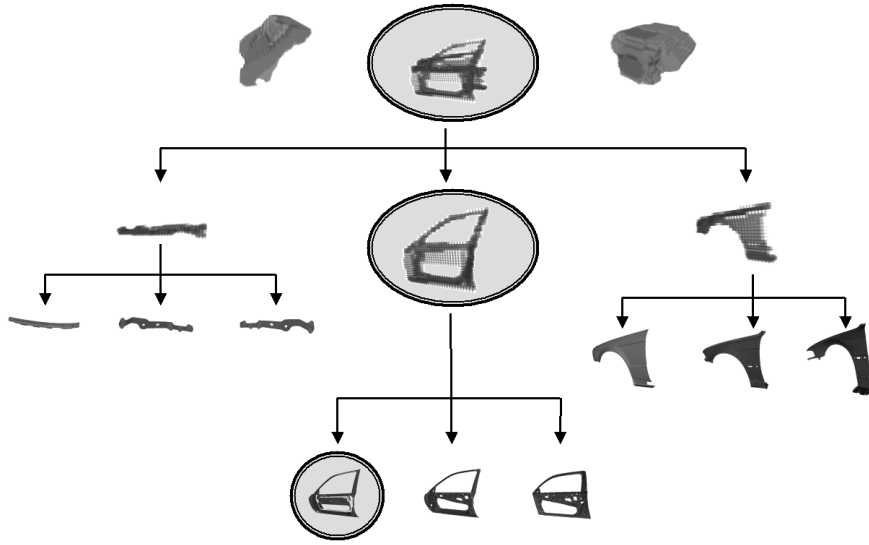


Figure 164: Hierarchically ordered representatives.

11.3.1 Recent Work

To the best of our knowledge, there are only two methods for automatic cluster extraction from hierarchical representations such as reachability plots or dendrograms which are both based on reachability plots. Since clusters are represented as valleys in the reachability plot, the task of automatic cluster extraction is to identify significant valleys.

The first approach proposed in [ABKS 99] called ξ -clustering is based on the steepness of the valleys in the reachability plot. The steepness is defined by means of an input parameter ξ . The method suffers from the fact that this input parameter is difficult to understand and hard to determine. Rather small variations of the value ξ often lead to drastic changes of the resulting clustering hierarchy. As a consequence, this method is unsuitable for our purpose of automatic cluster extraction.

The second approach was proposed recently by Sander et al. [SQL+ 03]. The authors describe an algorithm called *cluster_tree* that automatically extracts a hierarchical clustering from a reachability plot and computes a cluster tree. It is based on the idea that *significant* local maxima in the reachability plot separate clusters. Two parameters are introduced to decide whether a local maximum is significant: The first parameter specifies the minimum cluster size, i.e. how many objects must be located between two significant local maxima. The second parameter specifies the ratio between the reachability of a significant local maximum m and the average reachability

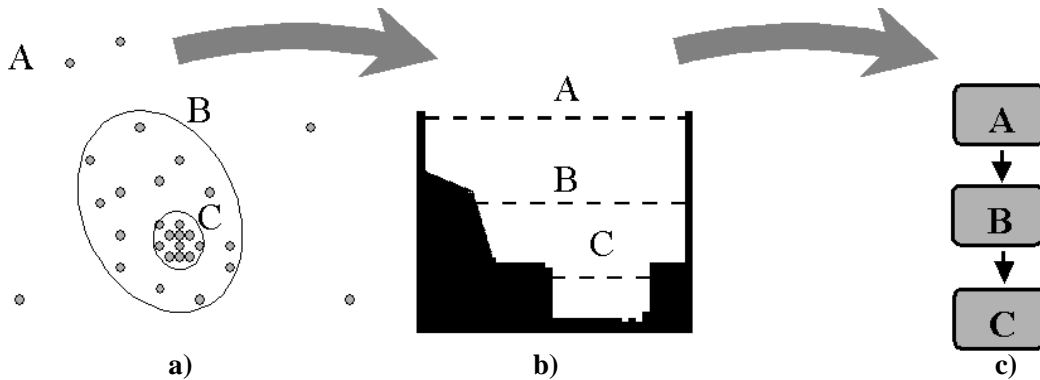


Figure 165: Sample narrowing clusters.

a) Data space, b) Reachability plot, c) Cluster hierarchy

ties of the regions to the left and to the right of m . The authors in [SQL+ 03] propose to set the minimum cluster size to 0.5% of the data set size and the second parameter to 0.75. They empirically show, that this default setting approximately represents the requirements of a typical user.

Although the second method is rather suitable for automatic cluster extraction from reachability plots, it has one major drawback. Many real-world data sets contain narrowing clusters, i.e. clusters consisting of exactly one smaller sub-cluster (cf. Figure 165).

Since the algorithm *cluster_tree* runs through a list of all local maxima (sorted in descending order of reachability) and decides at each local maximum m , whether m is significant to split the objects to the left of m and to the right of m into two clusters, the algorithm cannot detect such narrowing clusters. These clusters cannot be split by a significant maximum. Figure 165 illustrates this fact. The narrowing cluster A consists of one cluster B which is itself narrowing consisting of one cluster C . The algorithm *cluster_tree* will only find cluster A since there are no local maxima to split clusters B and C . The ξ -clustering will detect only one of the clusters A , B or C dependent on the ξ -parameter but also fails to detect the cluster hierarchy.

A new cluster recognition algorithm should meet the following requirements:

- It should detect all kinds of subclusters, including narrowing subclusters.
- It should create a clustering structure which is close to the one which an experienced user would manually extract from a given reachability plot.

- It should allow an easy integration into the OPTICS algorithm. We do not want to apply an additional cluster recognition step after the OPTICS run is completed. In contrast, the hierarchical clustering structure should be created on-the-fly during the OPTICS run without causing any noteworthy additional cost.

11.3.2 Gradient Clustering

In this section, we introduce our new *gradient-clustering* algorithm which fulfills all of the above mentioned requirements. The idea behind our new cluster extraction algorithm is based on the concept of *inflexion points*. During the OPTICS run, we decide for each point added to the result set, i.e. the reachability plot, whether it is an inflexion point or not. If it is an inflexion point, we might be at the start or at the end of a new subcluster. We store the possible starting points of the subclusters in a list, called *startPts*. This stack consists of pairs $(o.P, o.R)$, where $o.P$ denotes the position of the object o in the cluster ordering CO and $o.R$ the reachability value belonging to o w.r.t. CO (cf. Section 9.1.2). Our *gradient-clustering* algorithm can easily be integrated into OPTICS and is described in full detail, after we have formally introduced the new concept of *inflexion points*.

In the following, we assume that CO is a cluster ordering as defined in Definition 43. We call two objects $o_1, o_2 \in CO$ *adjacent in CO*, if $o_2.P = o_1.P + 1$. Let us recall, that $o.R$ is the reachability of $o \in CO$ assigned by OPTICS while generating CO . For any two objects $o_1, o_2 \in CO$ adjacent in the cluster ordering, we can determine the gradient of the reachability values $o_1.R$ and $o_2.R$. The gradient can easily be modelled as a 2D vector where the y-axis measures the reachability values ($o_1.R$ and $o_2.R$) in the ordering, and the x-axis represent the ordering of the objects. If we assume that each object in the ordering is separated by width w , the gradient of o_1 and o_2 is the vector

$$\vec{g}(o_1, o_2) = \begin{pmatrix} w \cdot (o_2.P - o_1.P) \\ o_2.R - o_1.R \end{pmatrix}$$

An example for a gradient vector of two objects x and y adjacent in a cluster ordering is depicted in Figure 166.

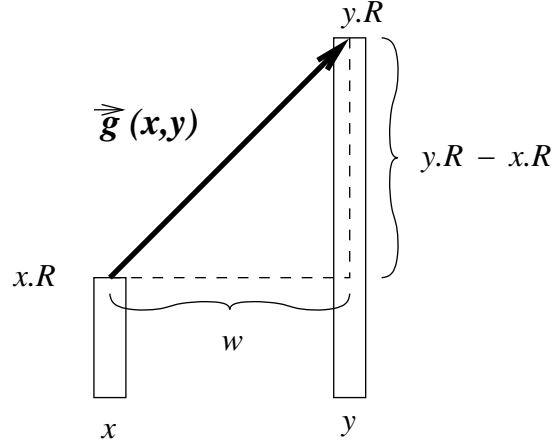


Figure 166: Gradient vector.

Gradient vector $\vec{g}(x, y)$ of two objects x and y adjacent in the cluster ordering.

Intuitively, an inflexion point should be an object in the cluster ordering where the gradient of the reachabilities changes significantly. This significant change indicates a starting or an end point of a cluster.

Let $x, y, z \in CO$ be adjacent, i.e. $x.P+1 = y.P = z.P-1$. We can now measure the differences between the gradient vector $\vec{g}(x, y)$ and $\vec{g}(z, y)$ ($= -\vec{g}(y, z)$) by computing the cosinus function of the angle between these two vectors. The cosinus of this angle is equal to -1 if the angle is 180° , i.e. the two vectors $\vec{g}(x, y)$ and $\vec{g}(y, z)$ have the same direction. On the other hand, if the gradient vectors differ a lot, the angle between them will be clearly smaller or larger than 180° and thus the cosinus will be significantly greater than -1. This observation motivates the concepts of inflection index and inflexion points:

Definition 47 (*Inflexion Index*).

Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO . The inflexion index of y , denoted by $II(y)$, is defined as the cosinus of the angle between the gradient vector of x, y ($\vec{g}(x, y)$) and the gradient vector of z, y ($\vec{g}(z, y)$), formally:

$$II(y) = \cos \varphi(\vec{g}(x, y), \vec{g}(z, y)) = \frac{-w^2 + (x.R - y.R)(z.R - y.R)}{\|\vec{g}(x, y)\| \cdot \|\vec{g}(z, y)\|},$$

where $\|\vec{v}\| := \sqrt{v_1^2 + v_2^2}$ is the length of the vector \vec{v} .

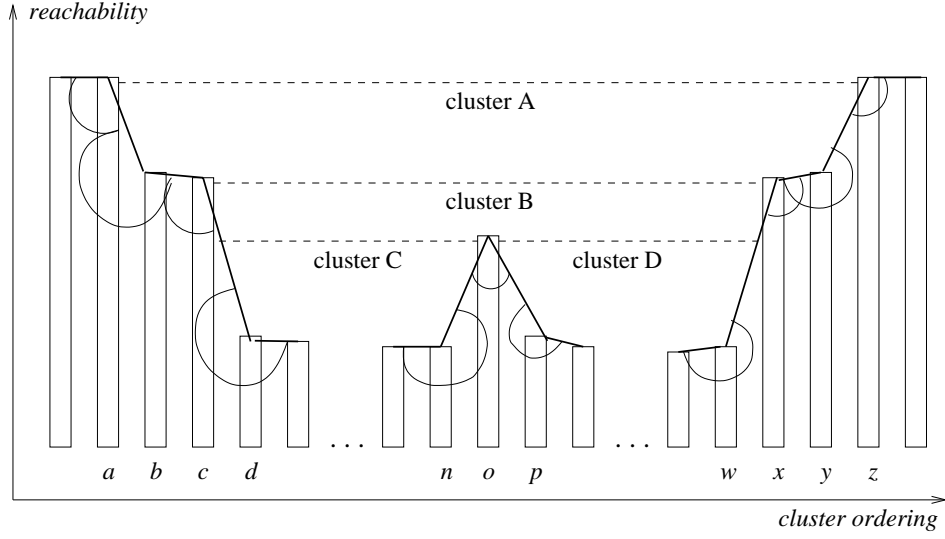


Figure 167: Inflexion points.

Illustration of inflexion points measuring the angle between the gradient vectors of objects adjacent in the ordering.

Definition 48 (*Inflexion Point*).

Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO and let $t \in \mathbb{R}$. Object y is an inflexion point iff

$$II(y) > t$$

The concept of inflexion points is suitable to detect objects in CO which are interesting for extracting clusters. For further distinguishing whether an inflexion point indicates a starting point or an end point, we introduce the *gradient determinant*.

Definition 49 (*Gradient Determinant*).

Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO . The gradient determinant of the gradients $\vec{g}(x, y)$ and $\vec{g}(z, y)$ is defined as

$$gd(\vec{g}(x, y), \vec{g}(z, y)) = \begin{vmatrix} w & -w \\ y.R - x.R & y.R - z.R \end{vmatrix}$$

If x, y, z are clear from the context, we use the short form $gd(y)$ for the gradient determinant $gd(\vec{g}(x, y), \vec{g}(z, y))$. The sign of $gd(y)$ indicates whether $y \in CO$ is a starting point or end point of a cluster. In fact, we can distinguish the following two cases which are visualized in Figure 167:

- $II(y) > t$ and $gd(y) > 0$:
Object y is either a starting point of a cluster (e.g. object a in Figure 167) or the first object outside of a cluster (e.g. object z in Figure 167).
- $II(y) > t$ and $gd(y) < 0$:
Object y is either an end point of a cluster (e.g. object n in Figure 167) or the second object inside a cluster (e.g. object b in 167).

Let us note that a local maximum $m \in CO$ which is the cluster separation point in [SQL+ 03] is a special form of the first case (i.e. $II(m) > t$ and $gd(m) > 0$).

The threshold t is independent from the absolute reachability values of the objects in CO . The influence of t is also very comprehensible because if we know which values for the angles between gradients are interesting, we can easily compute t . For example, if we are interested in angles $< 120^\circ$ and $> 240^\circ$ we set $t = \cos 120^\circ = -0.5$.

Obviously, the gradient-clustering algorithm is able to extract narrowing clusters. Experimental comparisons with the methods in [ABKS 99] and [SQL+ 03] are presented in Section 11.6.

The pseudo code of the gradient-clustering algorithm is depicted in Figure 168, which works like this. Initially, the first object of the cluster ordering CO is pushed to the stack of starting points $startPts$. Whenever a new starting point is found, it is pushed to the stack. If the current object is an end point, a new cluster is created containing all objects between the starting point on top of the stack and the current end point. Starting points are removed from the stack if their reachability is lower than the reachability of the current object. Clusters are created as described above for all removed starting points as well as for the starting point which remains in the stack. The input parameter $MinPts$ determines the minimum cluster size and the parameter t was discussed above. Finally, we suggest to set $w = \frac{\max\{o.R \mid \forall o \in CO: o.R \neq \infty\}}{100}$ which yields good results for many different cluster orderings.

11.4 Cluster Representatives

In this section, we present three different approaches to determine representatives for clusters computed by OPTICS. A simple approach could be to superimpose all objects of a cluster to build the representative as it is depicted in Figure 164.

```

ALGORITHM gradient_clustering (ClusterOrdering CO, Integer MinPts, Real t)
BEGIN
    startPts := emptyStack;
    setOfClusters := emptySet;
    currCluster := emptySet;
    o := CO.getFirst(); // first object is a starting point
    startPts.push(o);
    WHILE o.hasNext() DO // for all remaining objects
        o := o.next;
        IF o.hasNext() THEN
            IF  $\Pi(o) > t$  THEN // inflexion point
                IF  $gd(o) > 0$  THEN
                    IF currCluster.size()  $\geq$  MinPts THEN
                        setOfClusters.add(currCluster);
                    END IF;
                    currCluster := emptySet;
                    IF startPts.top().R  $\leq$  o.R THEN
                        startPts.pop();
                    END IF;
                    WHILE startPts.top().R < o.R DO
                        setOfClusters.add(set of objects from startPts.top() to last end point);
                        startPts.pop();
                    END DO;
                    setOfClusters.add(set of objects from startPts.top() to last end point);
                    IF o.next.R < o.R THEN // o is a starting point
                        startPts.push(o);
                    END IF;
                ELSE
                    IF o.next.R > o.R THEN // o is an end point
                        currCluster := set of objects from startPts.top() to o;
                    END IF;
                END IF;
            ELSE // add clusters at end of plot
                WHILE NOT startPts.isEmpty() DO
                    currCluster := set of objects from startPts.top() to o;
                    IF (startPts.top().R > o.R) AND (currCluster.size()  $\geq$  MinPts) THEN
                        setOfClusters.add(currCluster);
                    END IF;
                    startPts.pop();
                END DO;
            END IF;
        END DO;
    RETURN setOfClusters;
END. // gradient_clustering

```

Figure 168: Pseudo code of the *gradient-clustering* algorithm.

However, this approach has the huge drawback that the representatives on a higher level of the cluster hierarchy become rather unclear. Therefore, we choose real objects of the data set as cluster representatives.

In the following, CO denotes the cluster ordering from which we want to extract clusters. A cluster $C \subseteq CO$ will be represented by a set of k objects of the cluster, denoted as $REP(C)$. The number of representatives k can be a user defined number or a number which depends on the size and data distribution of the cluster C .

11.4.1 The Extended Medoid Approach

Many partitioning clustering algorithms are known to use medoids as cluster representatives. The medoid of a cluster C is the closest object to the mean of all objects in C . The mean of C is also called centroid. For $k > 1$ we could choose the k closest objects to the centroid of C as representatives.

The choice of medoids as cluster representative is somehow questionable. Obviously, if C is not of convex shape, the medoid is not really meaningful.

An extension of this approach coping with the problems of clusters with non-convex shape is the computation of k medoids by applying a k -medoid clustering algorithm to the objects in C . The clustering using a k -medoid algorithm is rather efficient due to the expectation that the clusters are much smaller than the whole data set. This approach can also be easily extended to cluster hierarchies. At any level we can apply the k -medoid clustering algorithm to the merged set of objects from the child clusters or -due to performance reasons- merge the medoids of child clusters and apply k -medoid clustering on this merged set of medoids.

11.4.2 The Minimum Core-Distance Approach

The second approach to choose representative objects of hierarchical clusters uses the density-based clustering notion of OPTICS. The core-distance $o.C = core-dist(o)$ of an object $o \in CO$ (cf. Definition 40) indicates the density of the surrounding region. The smaller the core-distance of o , the denser the region surrounding o . This observation led us to the choice of the object having the minimum core-distance as representative of the respective cluster. Formally, $REP(C)$ can be computed as:

$$REP(C) := \{o \in C \mid \forall x \in C : o.C \leq x.C\}.$$

We choose the k objects with the minimum core-distances of the cluster as representatives. The straightforward extension for cluster hierarchies is to choose the k objects from the merged child clusters having the minimum core-distances.

11.4.3 The Maximum Successor Approach

The third approach to choose representative objects of hierarchical clusters also uses the density-based clustering notion of OPTICS but in a more sophisticated way. In fact, it makes use of the density-connected relationships underlying the OPTICS algorithm.

As mentioned above, the result of OPTICS is an ordering of the database minimizing the reachability relation. At each step of the ordering, the object o having the minimum reachability w.r.t. the already processed objects occurring before o in the ordering is chosen. Thus, if the reachability of object o is not ∞ , it is determined by $reach-dist(o, p)$ where p is a unique object located before o in the cluster ordering. We call p the *predecessor* of o , formally:

Definition 50 (*Predecessor*).

Let CO be a cluster ordering. For each entry $o \in CO$ the predecessor is defined as

$$Pre(o) = \begin{cases} p & \text{if } \begin{pmatrix} p.P < o.P \\ \wedge o.R = reach-dist(o, p) \neq \infty \\ \wedge \forall x \in CO: x.P < o.P \Rightarrow \\ reach-dist(o, x) \geq reach-dist(o, p) \end{pmatrix} \\ UNDEFINED & \text{if } o.R = \infty \end{cases}$$

Intuitively, $Pre(o)$ is the object in CO from which o has been reached. Let us note that an object and its predecessor need not to be adjacent in the cluster ordering.

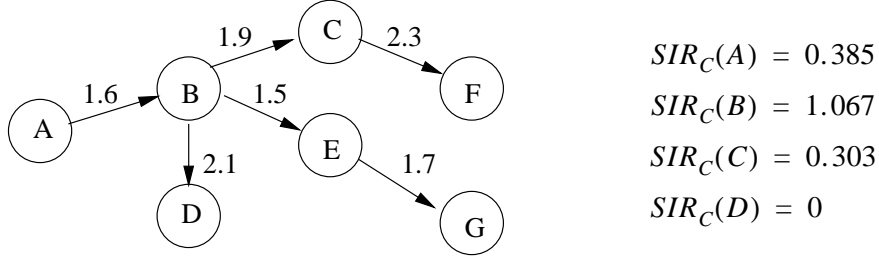
Definition 51 (*Successor*).

Let CO be a cluster ordering. For each entry $o \in CO$ in a cluster ordering computed by OPTICS, the set of successors is defined as

$$Suc(o) = \{s \in CO \mid Pre(s) = o\}$$

Let us note that objects may have no predecessor, e.g. each object having a reachability of ∞ does not have a predecessor, including the first object in the ordering. On the other hand, some objects may have more than one successor. In that case, some other objects have no successors. Again, an object and its successors need not to be adjacent in the ordering.

We can model this successor-relationship within each cluster as a directed *successor graph* where the nodes are the objects of one cluster and a directed edge from

**Figure 169:** Successor graph.

Sample successor graph for a cluster of seven objects with some SIR-values.

object o to s represents the relationship $s \in Suc(o)$. Each edge (x, y) can further be labeled by *reach-dist* (y, x) , i.e. by $y.R$. A sample successor graph is illustrated in Figure 169.

For the purpose of computing representatives of a cluster, the objects having many successors are interesting. Roughly speaking, these objects are responsible for the most density-connections within a cluster. The reachability values of these “connections” further indicate the distance between the objects. For example, for the objects in the cluster visualized in Figure 169, object B is responsible for the most density-connections since its node in the successor graph has the most out-going edges.

Our third strategy selects the representatives of clusters by maximizing the number of successors and minimizing the according reachabilities. For this purpose, we compute for each object o of a cluster C , the Sum of the *Inverse Reachability* distances of the successors of o within C , denoted by $SIR_C(o)$:

$$SIR_C(o) = \begin{cases} 0 & , \text{ if } Suc(o) = \emptyset \\ \sum_{\substack{s \in Suc(o) \\ s \in C}} \frac{1}{1 + reach-dist(s, o)} & , \text{ otherwise} \end{cases}$$

We add 1 to *reach-dist* (s, o) in the denominator to weight the impact of the number of successors over the significance of the reachability values. Based on $SIR_C(o)$, the representatives can be computed as follows:

$$REP(C) := \{o \in C \mid \forall x \in C : SIR_C(o) \geq SIR_C(x)\}.$$

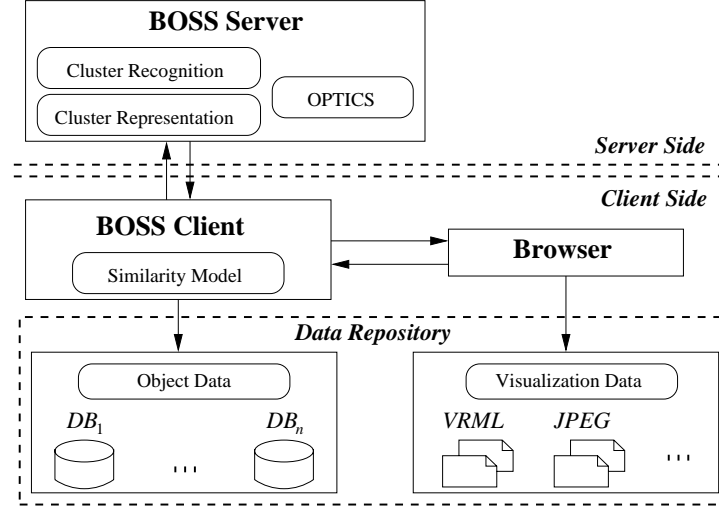


Figure 170: BOSS distributed architecture.

In Figure 169, the SIR-values of some objects of the depicted successor graph for a cluster of seven objects are computed. Since D has no successors, $\text{SIR}_C(D)$ is zero. In fact object B has the highest SIR-value indicating the central role of B in the cluster: B has three successors with relatively low reachability distance values. Our third strategy would select object B as representative for the cluster.

Let us note that there is no additional overhead to compute the reachability distances $\text{reach-dist}(\text{Suc}(o), o)$ for each $o \in CO$ since these values have been computed by OPTICS during the generation of CO and $\text{reach-dist}(\text{Suc}(o), o) = \text{Suc}(o).R$.

If we want to select k representatives for C we simply have to choose the k objects with the maximum SIR_C values.

11.5 System Architecture

The development of the industrial prototype BOSS is an important step towards developing a comprehensive, scalable and distributed computing solution designed to make the effectiveness of OPTICS and the proposed cluster recognition and representation algorithms available to a broader audience. BOSS is a client/server system allowing users to provide their own data locally, along with an appropriate similarity model (cf. Figure 170).

The data provided by the user will be comprised of the objects to be clustered, as well as a data set to visualize these objects, e.g. VRML files for CAD data

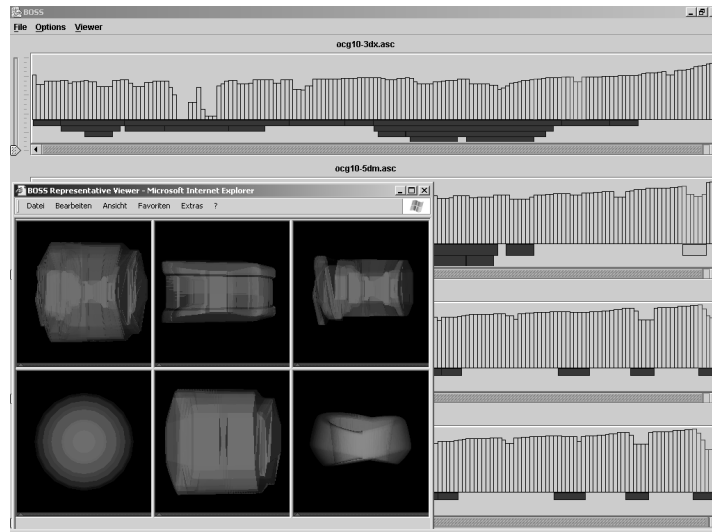


Figure 171: BOSS screenshot.

(cf. Figure 171) or JPEG images for multi-media data. Since this data resides on the user's local computer and is not transmitted to the server heavy network traffic can be avoided. In order for BOSS to be able to interpret this data, the user must supply his own similarity model with which the reachability data can be calculated.

The independence of the data processing and the data specification enables maximum flexibility. Further flexibility is introduced through the support of external visual representation. As long as the user is capable of displaying the visualization data in a browser, e.g. by means of a suitable plug-in, the browser will then load web pages generated by BOSS displaying the appropriate data. Thus, multimedia data such as images or VRML files can easily be displayed (cf. Figure 171). By externalizing the visualization procedure we can resort to approved software components, which have been specifically developed for displaying objects which are of the same type as the objects within our clusters.

11.6 Evaluation

We evaluated both the effectiveness and efficiency of our approaches using two real-world test data sets CAR and PLANE. The first one contains approximately 200 CAD objects from a German car manufacturer, the second one 5000 CAD objects

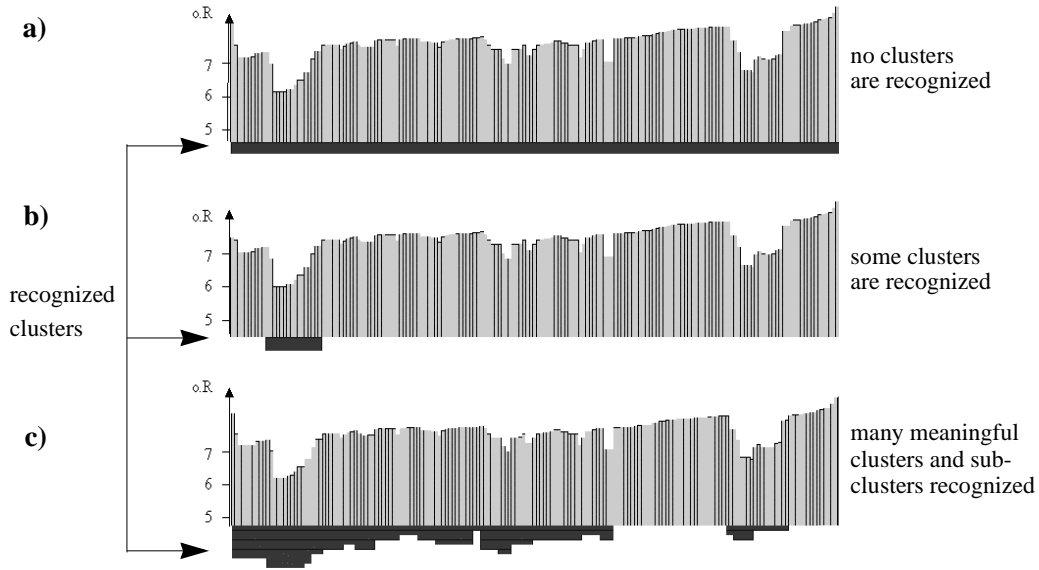


Figure 172: Cluster recognition of CAR parts.

(Vector set model with 7 covers)

a) *Cluster_tree*, b) ξ -clustering, and c) *Gradient-clustering*

from an American aircraft producer (cf. Section 9.2.1). We tested on a workstation with a 1.7 GHz CPU and 2 GB RAM.

In the following, three cluster recognition algorithms will vie among themselves, after which the three approaches for generating representatives will be evaluated.

11.6.1 Cluster Recognition

Automatic cluster recognition is clearly very desirable when analyzing large sets of data. In this section, we will first discuss the quality of our three cluster recognition algorithms. For this evaluation we use the CAR data set. Secondly, we discuss the efficiency by using the CAR and the PLANE data set.

Effectivity. The CAR data set exhibit the commonly seen quality of unpronounced but nevertheless to the observer clearly visible clusters. The corresponding reachability plots of the different cluster recognition algorithms are depicted in Figure 172.

Figure 172a shows that the *cluster_tree* algorithm does not find any clusters at all in the CAR data set, with the suggested default ratio-parameter of 75% [SQL+ 03]. In order to detect clusters in the CAR data set, we had to adjust the ratio-input parameter to 95%. In this case the *cluster_tree* algorithm detected some clusters but missed

out on some other important clusters and did not detect any cluster hierarchies at all. If we have rather high reachability values, e.g. values between 5-7 as in Figure 172, the ratio-parameter for the *cluster_tree* algorithm should be higher than for smaller values. Let us note that we made similar observations for the PLANE data set. Generally, in cases where a reachability graph consists of rather high reachability values or does not present spikes at all, but clusters are formed by smooth troughs in the waveform, this cluster recognition algorithm is unsuitable. Furthermore, it is inherently unable to detect narrowing clusters where a cluster has one sub-cluster of increased density (cf. Figure 165).

On the other hand, the ξ -clustering approach successfully recognizes some clusters while also missing out on significant subclusters (cf. Figure 172b). This algorithm has some trouble recognizing cluster structures with a significant differential of “steepness”. For instance, in Figure 165 it does not detect the narrowing cluster *B* inside of cluster *A* because it tries to create steep down-areas containing as many points as possible. Thus, it will merge the two steep edges if their steepness exceeds the threshold ξ .

Finally, we look at our new *gradient-clustering* algorithm. Figure 172c shows that the recognized cluster structure is close to the intuitive one, which an experienced user would manually derive. Clusters which are clearly distinguishable and contain more than *MinPts* elements are detected by this algorithm. Not only does it detect a lot of clusters, but it also detects a lot of meaningful cluster hierarchies, consisting of narrowing subclusters.

To sum up, in all our tests the *gradient-clustering* algorithm detected much more clusters than the other two approaches, without producing any redundant and unnecessary cluster information.

Efficiency. In all tests, we first created the reachability plots and then applied the algorithms for cluster recognition and representation. Let us note that we could also have integrated the *gradient-clustering* into the OPTICS run without causing any noteworthy overhead.

The overall runtimes for the three different cluster recognition algorithms are depicted in Figure 173. Our new *gradient-clustering* algorithm does not only produce the most meaningful results, but also in sufficiently short time.

	CAR (200 parts)	PLANE (5000 parts)
<i>cluster_tree</i>	0.06 s	1.93 s
ξ -clustering	0.22 s	5.06 s
<i>gradient-clustering</i>	0.31 s	3.57 s

Figure 173: CPU time for cluster recognition.

11.6.2 Cluster Representation

After a cluster recognition algorithm has analyzed the data, algorithms for cluster representation can help to get a quick visual overview of the data. With the help of representatives, large sets of objects may be characterized through a single object of the data set. We extract sample clusters from the CAR data set in order to evaluate the different approaches for cluster representatives. In our first tests, we set the number of representatives to $k = 1$.

The objects of one cluster from the car data set are displayed in Figure 174. The annotated objects are the representatives computed by the respective algorithms. Both the *Maximum Successor* and the *Minimum Core Distance* approaches yield good results. Despite the slight inhomogeneity of the cluster, both representatives sum up the majority of the elements within the cluster. This cannot be said of the representative computed by the commonly used medoid method, which selects an object from the trailing end of the cluster. This cluster and the corresponding representatives are no isolated cases, but reflect our general observations. Nevertheless, there have been some rare cases where the medoid approach yielded the more intuitive representative than the other two approaches.

If we allow a higher number of representatives, for instance $k = 3$, it might be better to display the representatives of all three approaches to reflect the content of the cluster, instead of displaying the three best representatives of one single approach. If we want to confine ourselves to only one representative per cluster, the best possible choice is to use the representative of the *Maximum Successor* approach.

11.6.3 Summary

The results of our experiments show, that our new approaches for the automatic cluster extraction and for the determination of representative objects outperform existing methods. It theoretically and empirically turned out, that our *gradient-cluster-*

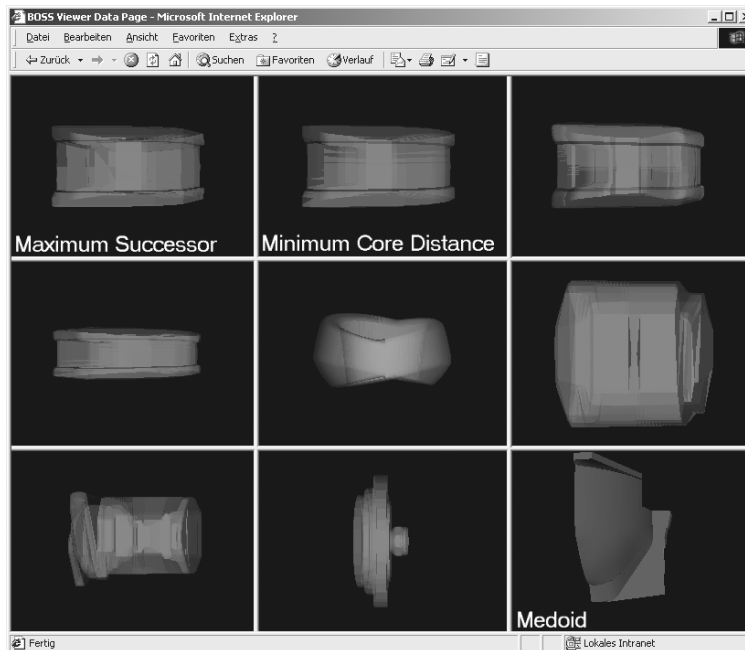


Figure 174: Representatives displayed by the BOSS object viewer.

ing algorithm seems to be more practical than recent work for automatic cluster extraction from hierarchical cluster representations. We also empirically showed that our approaches for the determination of cluster representatives is in general more suitable than the simple (extended) medoid approach. We refer the reader to [Vie 03] and [BK KP 04] for a more detailed evaluation which confirm the observations made in this chapter.

11.7 Summary

In this chapter, we proposed hierarchical clustering combined with automatic cluster recognition and selection of representatives as a promising visualization technique. Its areas of application include visual data mining, similarity search and evaluation of similarity models. We surveyed three approaches for automatic extraction of clusters. The first method, ξ -clustering, fails to detect some clusters present in the clustering structure and suffers from the sensitivity concerning the choice of its input parameter. The algorithm *cluster_tree* is by design unsuitable in the presence of narrowing clusters. To overcome these shortcomings, we proposed a new method, called

gradient-clustering. The experimental evaluation showed that this algorithm is able to extract narrowing clusters. Furthermore, it can easily be integrated into the hierarchical clustering algorithm OPTICS producing no noteworthy overhead. The cluster hierarchies produced by the *gradient-clustering* are similar to the clustering structures which an experienced user would manually extract.

Furthermore, we presented three different approaches to determine representative objects for clusters. The commonly known medoid approach is shown to be questionable for real-world data, while the approaches minimizing the core-distance and maximizing the successors both deliver good results.

Finally, we described our industrial prototype, called BOSS, that implements the algorithms presented in this chapter.

Chapter 12

Conclusions

In this thesis about “*spatial database support for virtual engineering*” we have presented and evaluated techniques for the effective, efficient and seamless integration of collision queries and similarity queries into standard object-relational database management systems. In this last chapter, we conclude our work with a short summary of the main theoretical and practical results (cf. Section 12.1). Furthermore, we give indications for possible future research directions (cf. Section 12.2).

12.1 Summary and Contributions

This thesis has presented and evaluated techniques for the seamless integration of spatial data management into standard object-relational databases. In this section, we summarize the main theoretical and practical contributions of our work.

12.1.1 Virtual Engineering (Part I)

In the first part of this thesis, we discussed different application ranges of virtual engineering, with a special emphasis on digital mockup and similarity search. We introduced collision queries and similarity queries based on voxelized CAD objects which form the foundation of modern engineering information systems. These two query types can efficiently be carried out by using the paradigm of multi-step query processing, which was also shortly sketched in this introductory part.

12.1.2 Database Support for Digital Mockup (Part II)

In the second part of this thesis, we concentrated on the efficient detection of colliding CAD objects on top of off-the-shelf object-relational database management systems. In order to get both industrial-strength and efficiency, relational access methods are required. Relational indexing is an interesting and important research area which attracted an increasing attention in the last years. We presented a detailed survey and classification of the various and most commonly used relational access methods. We illustrated the presented concepts by discussing the RR-tree, the RQ-tree and the RI-tree. In an industrial case study, we integrated geometry based search into the productive data management system of the Volkswagen AG. Thereby, we motivated the need of exact selectivity estimation in order to find the most suitable query execution plans.

To complete the object-relational integration of spatial index structures, we developed a cost model for spatial intersection queries on RI-trees. We showed the suitability of a quantile-based selectivity estimation which served as a foundation for the presented cost model. According to our experimental evaluation, the average relative error of the estimated selectivity to the actual one was in all tests beneath 30%. Based on this rather accurate selectivity estimation, the average relative error of the estimated I/O cost to the actual I/O cost of index scans ranged from 0% to 32%.

Not only did we use the quantile-based statistics to estimate the costs, but we also used it to improve the access methods themselves. We explained how we can apply the statistics maintained by the cost models to accelerate the query process for space partitioning index structures. The main idea was to reduce the number of generated join partners by grouping different join partners together according to a statistic driven grouping algorithm. For hierarchical data partitioning index structures, we proposed to reduce the navigational index traversal cost by using extended index range scans. In our experiments, we applied the presented techniques to accelerate the RR-tree, the RQ-tree and the RI-tree. Thereby, we achieved an average speed-up between 20% and 10,000% for spatial collision queries.

Furthermore, we presented a cost based decomposition approach for complex spatial objects into gray containers which are stored in a compressed way within an ORDBMS. The gray containers have been created by using a cost-based decomposition algorithm which takes the access probability and the decompression cost of the gray containers into account. The experimental evaluation on real-world test data points out that our new concept accelerates collision queries on the RR-tree, the RQ-tree, and the RI-tree as well as spatial join processing by up to two orders of magnitude.

12.1.3 Database Support for Similarity Search (Part III)

In the third part of this thesis, we turned our attention to the effective and efficient retrieval of similar objects. After shortly sketching various aspects and techniques from the literature, we concentrated on suitable similarity models for voxelized CAD objects. We introduced three specific space partitioning similarity models, namely the volume model, the solid-angle model and our new eigen-value model. Each of these models is based on a complete partitioning of the data space into disjoint cells. The volume model compares two objects based on the number of the object voxels in each cell of the space partitioning, the solid-angle model measures the concavity and the convexity of geometric surfaces, and the eigen-value model uses the scattering of the voxel sets to distinguish the objects by computing the minimum bounding ellipsoid of the voxel set in each cell of the given space partitioning. By means of a new evaluation method based on density-based hierarchical clustering, we showed that the eigen-value model yields the most meaningful results of the three space partitioning models. Unfortunately, it is almost three times slower than the other two

models because of the three times higher dimensionality of the corresponding feature vectors.

Besides the space partitioning models, we also investigated data partitioning similarity models, starting with the cover-sequence model. The basic idea of this model is to find large covers which approximate the object as good as possible. These covers are organized in a cover sequence which provides a sequential description of the object. The cover sequence is organized in a single feature vector, so that the distance between two objects can easily be determined by the corresponding feature distance. This model served as a starting point for our new vector set model which is based on a new paradigm in similarity search. In contrast to all the other four models, the vector set model uses sets of feature vectors for representing an object instead of single feature vectors. After introducing an appropriate distance measure on sets of feature vectors, i.e. the minimal matching distance, we introduced three different filter steps, the centroid approach, the Euclidean norm approach and the closest pair approach, helping to accelerate similarity queries. We showed that the combination of the centroid approach and the Euclidean norm approach is especially suitable for efficient similarity search on vector set data, as it can be computed efficiently and the information of each vector and each dimension is taken into consideration. Furthermore, we introduced suitable optimizations for the Relational M-tree, i.e. the Scanning M-tree, the Filtering M-tree and the Caching M-tree. We showed that although the Relational M-tree itself is clearly outperformed by the presented filters, the optimized Relational M-tree is the method of choice for efficiently carrying out similarity range queries.

To sum up, if a high quality similarity model is needed, the best possible choice is to use the vector set model. By means of the optimized Relational M-tree, which is based on the introduced filter techniques, the overall response time of the vector set model is also acceptable. Nevertheless, similarity queries based on the high quality vector set model are still slower than similarity queries on the single feature vector models. Therefore, the cover-sequence model and the eigen-value model might be the right choice, if the emphasis is on efficiency rather than on effectiveness.

Finally, we illustrated how an appropriate visualization of the hierarchical clustering structure can aid the user in his time consuming task of finding similar objects. We introduced a new approach for automatically extracting significant clusters out of a hierarchical cluster representation, called gradient clustering. Furthermore, we pre-

sented three different approaches for determining suitable cluster representatives, i.e. the medoid approach, the core-distance approach and the maximum successor approach. Based on these techniques, we developed a new data mining tool, called BOSS, which allows the user to get an overview over a large collection of CAD objects. Thus, even if the user has no specific part in mind, he or she might gain important insight and extract new knowledge from the hierarchical cluster representation of the data set.

12.2 Potentials for Future Work

We believe that this work can serve as a starting point for many new research activities. In the following, we will present different directions of ongoing research projects which have been motivated by this thesis. In Section 12.2.1, we shortly sketch how Part II of this thesis might be useful for haptic simulations. In Section 12.2.2, we describe how the patterns introduced in Chapter 6 can lead to a new similarity model for voxelized CAD data. In Section 12.2.3, we discuss two different approaches for accelerating density-based clustering algorithms. In Section 12.2.4, we introduce a new approach for navigating through massive data sets, which creates clusters of spherical shapes. We close this thesis in Section 12.2.5 with a short discussion about a browsing tool for distributed data sets.

12.2.1 Efficient Haptic Simulation

Many approaches have been developed to 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 (cf. Section 1.1.3). In order to provide real-time haptic rendering, intelligent pre-fetching strategies have to be combined with efficient spatial index structures. We think that the cost-based decomposition algorithm of spatial objects as presented in Chapter 6 together with the statistic-driven query optimization approach of Chapter 5 form a good starting point for developing new algorithms suitable for efficient haptic rendering [Had 04].

12.2.2 A Similarity Model based on Patterns

In Chapter 6, we showed that space filling curves enumerate the data space in a structured way, and that we can find these “structures” in the resulting voxel sequence describing our spatial object. We used these patterns for the efficient storage of our decomposed gray containers. As patterns describe the interesting, differentiating area of an object, we suggest to build a new similarity distance function based on patterns. As the concrete form of a pattern depends on the location of the object within the data space, we try to extract information from the patterns which are little susceptible to translation, rotation, and reflection. A good approach might be to use aggregated information of the various patterns, i.e. a pattern histogram. This histogram contains information about the number of the patterns, their length and their density. By means of a suitable distance function between different pattern histograms, we try to get a similarity model which is inherently invariant w.r.t. rotation, reflection and translation so that no further normalization steps are required [Bra 04].

12.2.3 Efficient Density-Based Clustering

In this section, we introduce two different approaches for accelerating density-based clustering algorithms which form the foundation of BOSS.

Filter based Clustering. For all complex objects where filters are available, we can also cluster based on the filter information instead of the exact information. This would result in a much faster clustering algorithm which, on the other hand, might produce worse results. *Approximated clustering* seems to be a very promising research area for many application ranges, e.g. distributed clustering (cf. Section 12.2.5) and clustering of mobile objects [Tam 04], where a good trade off between quality and runtime is desired. The quality of the produced results could be measured as shown in [JKP 04a][JKP 04b].

When we are interested in the exact result, we can use an approach which is quite similar to the one used by the Filtering M-tree but which requires a little bit more bookkeeping. The main idea is as follows. We can accelerate clustering algorithms as for instance OPTICS [ABKS 99] by carrying out the range queries on the filters instead of computing the exact object representations. The thereby retrieved objects are inserted into the priority queue according to the filter distances. For the first element of the queue, we always compute the exact distance and insert it again into the priority queue. If for the first element of the queue the exact distance has already

been computed, we carry out a range query around this element on the filter database. This is a slightly simplified presentation of a more complex algorithm which produces exactly the same result as the density-based clustering algorithms DBSCAN and OPTICS but with less exact distance computations. Similar to the Filtering M-tree, the speedup of this approach heavily depends on the quality of the filters. Our first experiments showed that we can accelerate the OPTICS algorithm by up to one order of magnitude when using the centroid filter as lower bounding filter for the minimal matching distance.

Let us note that we can apply the idea of “*filtering clustering*” also to other clustering algorithms, e.g. k-means [McQ 67]. As filter for the feature vectors to be clustered, we might use the Euclidean norm of these vectors (cf. Section 10.1). For more details we refer the interested reader to [BKP 04b].

Index based Clustering. The above mentioned approach has two drawbacks. First, we need suitable filters. Second, the approach results in a rather high main memory footprint. In this section, we sketch a new approach based on metric index structures without the mentioned drawbacks. For each database object which is within an ε -range of a given query object, we have to perform an exact distance computation even if we use appropriate index structures. In this new approach, we do not compute all of these distances. Instead, we compute only the distances necessary to determine the core-distance of the query object. Furthermore, we manage subtrees within the priority queues of our density-based clustering algorithms exploiting the pre-clustering of the metric index structures. We assign two values to each subtree T indicating its minimal and maximal distance from a possible predecessor object. If a predecessor o_a of a subtree T has a minimum distance to T which is higher than the maximum distance of a predecessor o_b , we do not have to compute any distances between o_a and the objects covered by T . For more details we refer the interested reader to [BKP 04b].

12.2.4 Navigating through Massive Multimedia Data Sets

BOSS is based on the density-based hierarchical clustering algorithm OPTICS and on suitable cluster recognition and representation algorithms. In this section we suggest a different approach which helps to navigate through massive data sets which is not based on the concept of *density-connectivity*. For each object o , we carry out a range query around the object o , and determine the core-level of this object. Then we

order all objects according to a quality criterion, walk through the sorted list, and determine thereby a set of representatives. Each representative represents objects which are in a certain range. For the set of representatives we carry out range queries with a higher ϵ -value, order it again according to the quality criterion, and determine a even smaller subset of representatives. By recursively invoking this approach, we can create a browsing-tree, containing suitable representatives. For more details we refer the reader to [KKP 04].

As the representatives are computed while computing the tree, we do not need any additional algorithms for cluster recognition and representation. The main advantage of this approach is that the hierarchical clustering structure consists of clusters of spherical shape which seem more appropriate for similarity search than the density-connected clusters created by OPTICS. The resulting tree could be regarded as a browsing tool but also as a new metric index structure which was created by a bulk-loading operation. In order to make the approach dynamic, we carry out update operations similar to update operations on Slim-trees [TTSF 00], i.e. we propose to use a variant of the slim-down algorithm trying to keep the tree tight. The quality of the resulting dynamic browsing tool could be measured by means of numerical values reflecting the degree of overlapping nodes (cf. the fat-factor and the bloat-factor presented in [TTSF 00]). A decreasing quality might trigger a complete reclustering.

12.2.5 Browsing Distributed Data Sets

Nowadays, large amounts of heterogeneous, complex data reside on different, independently working computers which are connected to each other via local or wide area networks (LANs or WANs). Examples comprise distributed mobile networks, sensor networks or supermarket chains where check-out scanners, located at different stores, gather data unremittingly. Furthermore, international companies such as DaimlerChrysler have some data which is located in Europe and some data in the US. Those companies have various reasons why the data cannot be transmitted to a central site, e.g. limited bandwidth or security aspects.

In [JKP 03][JKP 04b] a distributed clustering algorithm based on DBSCAN was introduced which determines local representatives quite similar to the approach presented in the foregoing section. This distributed clustering algorithm together with the approaches outlined in Section 12.2.3 and Section 12.2.4 allow us to create an efficient and effective distributed browsing tool for complex multimedia objects.

List of Figures

1 Introduction

Fig. 1	Partial virtual prototype of a car.....	4
Fig. 2	Digital mock-up (DMU) [IWB 01]	5
Fig. 3	Spatial queries on CAD data.	6
Fig. 4	Browsing through a hierarchy of CAD objects.	7
Fig. 5	Sample scenario for haptic rendering.....	8
Fig. 6	Virtual environment of the International Space Station.....	8
Fig. 7	Spatial referencing of engineering documents.....	9

2 Spatial Engineering Databases

Fig. 8	Scan conversion on a triangulated surface.	18
Fig. 9	Filling a closed voxelized surface.	19
Fig. 10	Examples of space-filling curves in the two-dimensional case.....	19
Fig. 11	Object decompositions	20
Fig. 12	Feature transformation.	21
Fig. 13	Conservative and progressive approximations	23
Fig. 14	Multi-step query processing.....	24

3 Object Relational Indexing

Fig. 15	Classification of data models.....	31
Fig. 16	Methods for extensible index definition and manipulation.....	33
Fig. 17	Object-relational DDL statements for CAD data	34
Fig. 18	A custom index CADINDEX for CAD objects.....	35
Fig. 19	SQL-statements for intersection queries	35
Fig. 20	Methods for extensible query optimization.....	36
Fig. 21	Approaches to implement custom access methods	37

Fig. 22	B-tree routine <code>next_node</code> for different data types	39
Fig. 23	User-defined data type <code>FracNum</code>	40
Fig. 24	Paradigms and characteristics of access methods	45
Fig. 25	The MBR-List, a simple example for a relational access method.	46
Fig. 26	Box queries on CAD data	50
Fig. 27	Relational mapping of an R-tree directory	52
Fig. 28	Cursor-driven window query on a Relational R-tree.	53
Fig. 29	Relational mapping of a Linear Quadtree	56
Fig. 30	Cursor-driven window query for the RQ-tree	57
Fig. 31	The Relational Interval Tree	58
Fig. 32	SQL statement for a single query interval with bind variables.	59
Fig. 33	Naive query processing for an interval sequence.	60
Fig. 34	Different execution plans.	64
Fig. 35	Optimal access paths	65
Fig. 36	Processing a query on the DIVE system.	66

4 A Cost Model for Spatial Intersection Queries

Fig. 37	Extensible indexing / optimization frameworks.	69
Fig. 38	Selectivity estimation on an interval histogram.	73
Fig. 39	Selectivity estimation on node quantiles.	76
Fig. 40	Touched leaf blocks and query gaps for an intersection query τ	78
Fig. 41	Additional I/O due to block gaps g between range queries.	80
Fig. 42	Approximation C of a fine-grained interval sequence F	82
Fig. 43	Extended RI-tree cost model.	85
Fig. 44	Real and estimated gap distribution.	86
Fig. 45	Histograms of interval distributions	87
Fig. 46	Computation cost of histogram-based and quantile-based statistics.	88
Fig. 47	Relative error of selectivity estimation for histograms and quantiles	89
Fig. 48	Relative error of selectivity estimation for varying statistic resolutions.	90
Fig. 49	Relative error of selectivity estimation	90
Fig. 50	Relative error for cost estimation	91
Fig. 51	Output cost and join overhead for queries evaluated	92
Fig. 52	Output cost and join cost for queries using interval sequences.	92

5 Statistic-Driven Acceleration of Relational Index Structures

Fig. 53	Accelerated query processing	100
Fig. 54	Cost-based tile grouping	103
Fig. 55	Accelerated window query on a Relational Quadtree	103
Fig. 56	Grouping algorithm <code>Decompose</code>	105

Fig. 57	An example for an hierarchical index structure (the RR-tree)	106
Fig. 58	SQL box intersection query on a RR-tree (Oracle syntax)	107
Fig. 59	Acceleration of hierarchical index structures	108
Fig. 60	Determination of the overlap-factor σ (RR-tree)	111
Fig. 61	RI-tree histograms	113
Fig. 62	Used index levels.	113
Fig. 63	RI-tree optimizations without using statistics	114
Fig. 64	Statistic based acceleration for two variants of the RI-tree.	115
Fig. 65	Statistic based accelerated RQ-tree on the CAR data set	116
Fig. 66	Box queries for the RI-tree on the PLANE data set	117
Fig. 67	RR-tree for varying selectivity.	118

6 Cost-based Decompositioning of Complex Spatial Objects

Fig. 68	Gray containers	126
Fig. 69	Patterns	129
Fig. 70	Flow diagram of QSDC compression algorithm.	133
Fig. 71	Query distribution functions $P_i(x,y)$	134
Fig. 72	Computation of average access probabilities of gray containers	136
Fig. 73	Grouping algorithm GroupCon	137
Fig. 74	SQL intersection-statements based on gray containers	139
Fig. 75	Spatial join procedure.	144
Fig. 76	Decompositioning algorithm JoinGroupCon.	148
Fig. 77	Nested-Loop join algorithm.	149
Fig. 78	Intervals stemming from R and the corresponding histograms.	151
Fig. 79	Two-phase sort-merge join.	152
Fig. 80	Interval length depending on the MAXGAP parameter	155
Fig. 81	Storage requirements for the RI-tree (PLANE).	155
Fig. 82	Update operations for the RI-tree (CAR)	156
Fig. 83	Boolean intersection queries for MaxGap(DC) (RI-tree (PLANE))	157
Fig. 84	Intersection queries (CAR)	158
Fig. 85	Boolean intersection queries for MaxGap(QSDC).	158
Fig. 86	Tested candidate pairs of gray query and database intervals.	159
Fig. 87	Candidate and result sets	160
Fig. 88	Response time	161
Fig. 89	GRP(DC) evaluated for the nested loop join (CAR data set)	163
Fig. 90	Overall nested-loop join performance for different packers	164
Fig. 91	Sort-merge join performance.	165
Fig. 92	Overall sort-merge join performance	166

7 Foundations of Similarity Search

Fig. 93	Similarity range query	173
Fig. 94	Similarity k-nearest neighbor query	174
Fig. 95	Examples of a q-ranking for two query points q' and q''	175
Fig. 96	SQL-Statement for a range query	176
Fig. 97	SQL-Statement for a k-nn query	176
Fig. 98	SQL-Statement for a ranked k-nn query	177
Fig. 99	SQL-Statement for a ranked k-nn query with additional constraints	177
Fig. 100	The R*-tree architecture	179
Fig. 101	Range query processing on R-trees	186
Fig. 102	Incremental ranking query processing on R-trees	188
Fig. 103	Similarity range query processor	190
Fig. 104	k-nearest neighbor query processor according to [KSF+ 96]	191
Fig. 105	Multi-step query processor for optimal k-nearest neighbor search	192
Fig. 106	Multi-step ranking queries	193
Fig. 106	Multi-step ranking queries	193
Fig. 107	The Section Coding feature transformation	194
Fig. 108	Section coding of 2D regions	196
Fig. 109	Shells and sections as basic models for shape histograms	197
Fig. 110	Classification of complex similarity models	198

8 Similarity Models for Voxelized CAD Data

Fig. 111	2D space partitioning with 4 cells	205
Fig. 112	The Solid-Angle model	206
Fig. 113	A 2D example for the eigen-value model	207
Fig. 114	The eigen-value model with the principal axis of a sample object	209
Fig. 115	The cover sequence model	210
Fig. 116	Advantages of free permutations	213
Fig. 117	An example of a feasible vertex labeling and an equality subgraph	220

9 Effectiveness of Similarity Models

Fig. 118	Results of 5-nn queries for a “good” and “bad” similarity model	225
Fig. 119	Density-reachability and density-connectivity	227
Fig. 120	Nested density-based clusters	228
Fig. 121	Illustration of core-distance and reachability-distance	230
Fig. 122	Reachability plot (right) computed by OPTICS for a 2D data set (left)	231
Fig. 123	Effects of parameter settings on the cluster ordering of OPTICS	232
Fig. 124	Reachability plots for the space partitioning similarity models	234

Fig. 125 Objects found by the solid-angle model (cf. Figure 124c)	235
Fig. 126 Objects found by the eigen-value model (cf. Figure 124e)	236
Fig. 127 Reachability plots for the cover sequence model with 7 covers	238
Fig. 128 The minimum Euclidean distance under permutation.	238
Fig. 129 Reachability plots for the vector set model	239
Fig. 130 Objects found by the cover sequence model (cf. Figure 127a)	240
Fig. 131 Objects found by the vector set model (cf. Figure 129c)	241

10 Efficiency of Similarity Models

Fig. 132 An ϵ -range query on an NB-tree	245
Fig. 133 2-dimensional example for the centroid filter.	248
Fig. 134 The Euclidean norm vector	249
Fig. 135 2-dimensional example for the Euclidean norm filter.	252
Fig. 136 2-dimensional example for the closest pair filter.	254
Fig. 137 Similarity range search on M-trees.	257
Fig. 138 Relational mapping of an M-tree directory	259
Fig. 139 SQL range query	259
Fig. 140 The overlap-factor σ for range queries on the Relational M-tree	261
Fig. 141 Positive Pruning for the M-tree.	262
Fig. 142 Positive Pruning on M-trees.	263
Fig. 143 Similarity range query based on the filtering M-tree.	264
Fig. 144 Filtering M-tree.	265
Fig. 145 FindSubTree-function for an M-tree.	266
Fig. 146 NodeSplit-function for an M-tree.	267
Fig. 147 Caching M-tree.	269
Fig. 148 SQL-Statement for range queries	270
Fig. 149 Query times of range queries on the single vector models	272
Fig. 150 Number of candidates of range queries on the single vector models	273
Fig. 151 Query times of k-nn queries on the single vector models	274
Fig. 152 Number of candidates of k-nn queries on the single vector models	275
Fig. 153 Number of candidates of range queries on the vector set model.	278
Fig. 154 Query times of range queries on the vector set model	279
Fig. 155 Number of candidates of k-nn queries on the vector set model	280
Fig. 156 Query times of k-nn queries on the vector set model	281
Fig. 157 k-nn queries on the vector set model (RM-tree)	283
Fig. 158 Range queries on the vector set model (RM-tree).	284
Fig. 159 Range queries on the vector set model (7 covers) (optimized RM-tree)	285
Fig. 160 Range queries on the vector set model (optimized RM-tree)	286

Fig. 161 Creation of a Relational M-tree	288
--	-----

11 BOSS: Browsing Optics-Plots for Similarity Search

Fig. 162 Different approaches to visual data mining	293
Fig. 163 Browsing through reachability plots with different thresholds ϵ_{cut} . . .	294
Fig. 164 Hierarchically ordered representatives	296
Fig. 165 Sample narrowing clusters.	297
Fig. 166 Gradient vector	299
Fig. 167 Inflexion points	300
Fig. 168 Pseudo code of the gradient-clustering algorithm	302
Fig. 169 Successor graph.	305
Fig. 170 BOSS distributed architecture	306
Fig. 171 BOSS screenshot	307
Fig. 172 Cluster recognition of CAR parts	308
Fig. 173 CPU time for cluster recognition.	310
Fig. 174 Representatives displayed by the BOSS object viewer	311

12 Conclusions

List of Definitions

Def. 1 (Relational Access Method)	46
Def. 2 (Cursor-Bound Operation)..	48
Def. 3 (Cursor-Driven Operation)	49
Def. 4 (Navigational Scheme)	51
Def. 5 (Direct Scheme)	55
Def. 6 (Interval Histogram).	72
Def. 7 (Histogram-based Selectivity Estimate).	72
Def. 8 (Quantile Vector)..	73
Def. 9 (Node Quantiles).	73
Def. 10 (Average Node Distances)..	74
Def. 11 (Span of Touched Nodes).	74
Def. 12 (Quantile-based Selectivity Estimate)..	75
Def. 13 (Intersection Ranking Function).	82
Def. 14 (Aggregates on Interval Sequences).	82
Def. 15 (Voxelized Object)	125
Def. 16 (Gray Container, Gray Container Sequence)	125
Def. 17 (Non-Overlapping Gray Containers)	127
Def. 18 (Object Intersection).	138
Def. 19 (Similarity Range Query)	172
Def. 20 (Similarity k-Nearest Neighbor Query)	173
Def. 21 (Similarity Ranking Query)	175
Def. 22 (Lower-Bounding Property)	189
Def. 23 (Metric).	200
Def. 24 (Metric Object Similarity)	200
Def. 25 (Feature-Based Object Similarity)	200

Def. 26 (Invariance)	201
Def. 27 (Extended Metric Object Similarity)	201
Def. 28 (Minimum Euclidian Distance under Permutation)	214
Def. 29 (Weighted Complete Bipartite Graph)	216
Def. 30 (Perfect Matching)	216
Def. 31 (Minimum Weight Perfect Matching)	217
Def. 32 (Enumeration of a Set)	217
Def. 33 (Minimal Matching Distance)	217
Def. 34 (Dummy Vectors)	218
Def. 35 (M-Alternating Path)	219
Def. 36 (Feasible Vertex Labeling)	219
Def. 37 (Directly Density-Reachable)	226
Def. 38 (Density-Reachable)	226
Def. 39 (Density-Connected)	227
Def. 40 (Cluster and Noise)	227
Def. 41 (Core-Distance)	229
Def. 42 (Reachability-Distance)	229
Def. 43 (Cluster Ordering)	230
Def. 44 (Extended Centroid)	247
Def. 45 (Euclidean Norm Vector)	249
Def. 46 (The Closest Pair Distance)	252
Def. 47 (Inflexion Index)	299
Def. 48 (Inflexion Point)	300
Def. 49 (Gradient Determinant)	300
Def. 50 (Predecessor)	304
Def. 51 (Successor)	304

References

- [ABKS 99] Ankerst M., Breunig M., Kriegel H.-P., Sander J.: *OPTICS: Ordering Points To Identify the Clustering Structure*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 49-60, 1999.
- [AFS 93] Agrawal R., Faloutsos C., Swami A. *Efficient Similarity Search in Sequence Databases*. Proc. 4th. Int. Conf. on Foundations of Data Organization and Algorithms (FODO '93), LNCS 730, 69-84, 1993.
- [AKKS 99] Ankerst M., Kastenmüller G., Kriegel H.-P., Seidl T.: *3D Shape Histograms for Similarity Search and Classification in Spatial Databases*. Proc. Int. Symposium on Large Spatial Databases (SSD), 207-226, 1999.
- [ALSS 95] Agrawal R., Lin K.-I., Sawhney H., Shim K.: *Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases*. Proc. 21th Int. Conf. on Very Large Databases (VLDB), 490-501, 1995.
- [Ank 00] Ankerst M.: *Visual Data Mining*. Ph.D. Thesis, Institute for Computer Science, University of Munich, 2000.
- [Aok 98] Aoki P. M.: *Generalizing "Search" in Generalized Search Trees*. Proc. 14th Int. Conf. on Data Engineering (ICDE), 380-389, 1998.
- [APR+ 98] Arge L., Procopiu O., Ramaswamy S., Suel T., Vitter J.S.: *Scalable Sweeping-Based Spatial Join*. Proc. 24th Int. Conf. on Very Large Databases (VLDB), 570-581, 1998.
- [APR 99] Acharya S., Poosala V., Ramaswamy S.: *Selectivity Estimation in Spatial Databases*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 13-24, 1999.
- [AV 96] Arge L., Vitter J. S.: *Optimal Dynamic Interval Management in External Memory*. Proc. 37th Annual Symp. on Foundations of Computer Science, 560-569, 1996.
- [Bay 96] Bayer R.: *The Universal B-Tree for multidimensional Indexing*. Technical University of Munich, TUM-I9637, 1996.

- [BBB+ 97] Berchtold S., Böhm C., Braunmüller B., Keim D. A., Kriegel H.-P.: *Fast Parallel Similarity Search in Multimedia Databases*. Proc. ACM SIGMOD Int. Conf. on Management of Data, Best Paper Award, 1-12, 1997.
- [BBD+ 01] Bercken J., Blohsfeld B., Dittrich J.-P., Krämer J., Schäfer T., Schneider M., Seeger B.: *XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries*. Proc. 27th Int. Conf. on Very Large Databases (VLDB), 39-48, 2001.
- [BBG+ 90] Batory D. S., Barnett J. R., Garza J. F., Smith K. P., Tsukuda K., Twichell B. C., Wise T. E.: *GENESIS: An Extensible Database Management System*. In: Zdonik S. B., Maier D. (eds.): *Readings in Object-Oriented Database Systems*. 500-518, Morgan Kaufman, 1990.
- [BBJ+ 00] Berchtold S., Böhm C., Jagadish H. V., Kriegel H.-P., Sander J.: *Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces*. Proc. Int. Conf. on Data Engineering (ICDE), 577-588, 2000.
- [BBK 98] Berchtold S., Böhm C., Kriegel H.-P.: *The Pyramid-Tree: Breaking the Curse of Dimensionality*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 142-153, 1998.
- [BBK 01] Böhm C., Berchtold S., Keim D.: *Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases*. ACM Computing Surveys, 2001.
- [BBK+ 00] Berchtold S., Böhm C., Keim D., Kriegel H.-P., Xu X.: *Optimal Multidimensional Query Processing Using Tree Striping*. Proc. 2nd. Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK'00), 244-257, 2000.
- [BBKK 97] Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space*. Proc. Int. Symposium on Principles of Database Systems (PODS), 78-86, 1997.
- [BBKM 99] Berchtold S., Böhm C., Kriegel H.-P., Michel U.: *Implementation of Multidimensional Index Structures for Knowledge Discovery in Relational Databases*. Proc. 1st Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK), LNCS 1676, 261-270, 1999.
- [BBKM 00] Berchtold S., Böhm C., Kriegel H.-P., Michel U.: *Multidimensional Index Structures in Relational Databases*. Journal of Intelligent Information Systems (JIIS), Vol. 15, No. 1, 51-70, 2000.
- [BDS 00] Bercken J., Dittrich J.-P., Seeger B.: *javax.XXL: A Prototype for a Library of Query Processing Algorithms*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 588, 2000.
- [BEK+ 98] Berchtold S., Ertl B., Keim D. A., Kriegel H.-P., Seidl T.: *Fast Nearest Neighbor Search in High-dimensional Space*. Proc. 14th Int. Conf. on Data Engineering (ICDE), 209-218, 1998.
- [BEKS 00] Braunmüller B., Ester M., Kriegel H.-P., Sander J.: *Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases*. Proc. 16th Int. Conf. on Data Engineering (ICDE), 256-267, 2000.
- [BF 95] Belussi A., Faloutsos C.: *Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension*. Proc. 21st Int. Conf. on Very Large Databases (VLDB), 299-310, 1995.

- [BK 97] Berchtold S., Kriegel H.-P.: *S3: Similarity Search in CAD Database Systems*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 564-567, 1997.
- [BKK+ 03] Brecheisen S., Kriegel H.-P., Kröger P., Pfeifle M., Viermetz M.: *Representatives for Visually Analyzing Cluster Hierarchies*. Proc. 4th Int. Workshop on Multimedia Data Mining MDM/KDD, 64-71, 2003.
- [BKK+ 04] Brecheisen S., Kriegel H.-P., Kröger P., Pfeifle M., Pötke M., Viermetz M.: *BOSS: Browsing OPTICS-Plots for Similarity Search*. Proc. 20th Int. Conf. on Data Engineering (ICDE), 2004.
- [BKK 96] Berchtold S., Keim D. A., Kriegel H.-P.: *The X-tree: An Index Structure for High-Dimensional Data*. Proc. 22nd Int. Conf. on Very Large Databases (VLDB), 28-39, 1996.
- [BKK 97a] Berchtold S., Keim D. A., Kriegel H.-P.: *Using Extended Feature Objects for Partial Similarity Retrieval*. VLDB Journal, 6(4): 333-348, 1997.
- [BKK 97b] Berchtold S., Keim D. A., Kriegel H.-P.: *Section Coding: Ein Verfahren zur Ähnlichkeitssuche in CAD-Datenbanken*. Proc. 7. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 152-171, 1997 (in German).
- [BKKP 04] Brecheisen S., Kriegel H.-P., Kröger P., Pfeifle M.: *Visually Mining Through Cluster Hierarchies*. Proc. SIAM Int. Conf. on Data Mining (SIAMDM'04), 2004.
- [BKKS 01] Breuning M., Kriegel H.-P., Kröger P., Sander J.: *Data Bubbles: Quality Preserving Performance Boosting*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 79-90, 2001.
- [BKKS 00] Berchtold S., Keim D. A., Kriegel H.-P., Seidl T.: *Indexing the Solution Space: A New Technique for Nearest Neighbor Search in High-Dimensional Space*. IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol. 12, No. 1, 45-57, 2000.
- [BKP 98] Berchtold S., Kriegel H.-P., Pötke M.: *Database Support for Concurrent Digital Mock-Up*. Proc. IFIP Int. Conf. PROLAMAT, Globalization of Manufacturing in the Digital Communications Era of the 21st Century, Kluwer Academic Publishers, 499-509, 1998.
- [BKP 04a] Brecheisen S., Kriegel H.-P., Pfeifle M.: *Efficient Query Processing on Sets of Feature Vectors*. Submitted for publication.
- [BKP 04b] Brecheisen S., Kriegel H.-P., Pfeifle M.: *Efficient Density-Based Clustering of Complex Objects*. University of Munich, Technical Report, 2004.
- [BKS 93a] Brinkhoff T., Kriegel H.-P., Schneider R.: *Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems*. Proc. 9th Int. Conf. on Data Engineering (ICDE), 40-49, 1993.
- [BKS 93b] Brinkhoff T., Kriegel H.-P., Seeger B.: *Efficient Processing of Spatial Joins Using R-trees*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 237-246, 1993.
- [BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *The R*-tree: An Efficient and Robust Access Method for Points and Rectangles*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 322-331, 1990.

- [BKSS 94] Brinkhoff T., Kriegel H.-P., Schneider R., Seeger B.: *Multi-Step Processing of Spatial Joins*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 197-208, 1994.
- [BMH 92] Badel A., Mornon J. P., Hazout S.: *Searching for Geometric Molecular Shape Complementarity using Bidimensional Surface Profiles*. Journal of Molecular Graphics, Vol. 10, 205-211, 1992.
- [BO 99] Boulos J., Ono K.: *Cost Estimation of User-Defined Methods in Object-Relational Database Systems*. ACM SIGMOD Record, 28(3), 22-28, 1999.
- [Bra 04] Braun D.: *Pattern based Similarity Search*. Diploma Thesis, University of Munich, 2004 (in German).
- [Bro 01] Brown P.: *Object-Relational Database Development – A Plumber’s Guide*. Informix Press, Menlo Park, CA, 2001.
- [BSSJ 99] Bliujute R., Saltenis S., Slivinskas G., Jensen C.S.: *Developing a DataBlade for a New Index*. Proc. 15th Int. Conf. on Data Engineering (ICDE), 314-323, 1999.
- [BW 94] Burrows M., Wheeler D. J.: *A Block-sorting Lossless Data Compression Algorithm*. Digital Systems Research Center Research Report 124, 1994.
- [CBM 97] Carey R., Bell G., Marrin C.: *ISO/IEC 14772-1:1997 Virtual Reality Modeling Language*. VRML Consortium, 1997.
- [CCF+ 99] Chen W., Chow J.-H., Fuh Y.-C., Grandbois J., Jou M., Mattos N., Tran B., Wang Y.: *High Level Indexing of User-Defined Types*. Proc. 25th Int. Conf. on Very Large Databases (VLDB), 554-564, 1999.
- [CCN+ 99] Carey M. J., Chamberlin D. D., Narayanan S., Vance B., Doole D., Rielau S., Swagerman R., Mattos N.: *O-O, What Have They Done to DB2?* Proc. 25th Int. Conf. on Very Large Databases (VLDB), 542-553, 1999.
- [CD 98] Chen F.-C. F., Dunham M. H.: *Common Subexpression Processing in Multiple-Query Processing*. IEEE Trans. on Knowledge and Data Engineering, 10(3), 493-499, 1998.
- [CDF+ 91] Carey M. J., DeWitt D. J., Frank D., Graefe G., Richardson J. E., Shekita E. J., Muralikrishna M.: *The Architecture of the EXODUS Extensible DBMS*. In: Dittrich K. R., Dayal U., Buchmann A. P. (eds.): *On Object-Oriented Database Systems*. 231-256, Topics in Information Systems, Springer, 1991.
- [CDG+ 90] Carey M. J., DeWitt D. J., Graefe G., Haight D. M., Richardson J. E., Schuh D. T., Shekita E. J., Vandenberg S. L.: *The EXODUS Extensible DBMS Project: An Overview*. In: Zdonik S. B., Maier D. (eds.): *Readings in Object-Oriented Database Systems*. 474-499, Morgan Kaufman, 1990.
- [CF 91] Clark K. B., Fujimoto T.: *Product Development Performance – Strategy, Organization, and Management in the World Auto Industry*. Harvard Business Scholl Press, Boston, MA, 1991.
- [CNBM 01] Chávez E., Navarro G., Baeza-Yates R., Marroquín J.: *Searching in Metric Spaces*. ACM Computing Surveys 33(3): 273-321, 2001.
- [Con 86] Connolly M.: *Shape Complementarity at the Hemoglobin *α*1b1 Subunit Interface*. Biopolymers, 25:1229-1247, 1986.

- [CPZ 97] Ciaccia P., Patella M., Zezula P.: *M-tree: An Efficient Access Method for Similarity Search in Metric Spaces*. Proc. Int. Conf. on Very Large Data Bases (VLDB), Athens, 426-435, 1997.
- [CS 97] Carey M. J., Kossmann D.: *On Saying "Enough Already!" in SQL*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 219-230, 1997.
- [CW 00] Cannane A., Williams H.: *A Compression Scheme for Large Databases*. Australian Database Conference (ADC), 6-11, 2000.
- [CWZ 99] Cannane A., Williams H., Zobel J.: *A General-Purpose Compression Scheme for Databases*. Data Compression Conference (DCC), 519-520, 1999.
- [CZ 01] Chaudhri A. B., Zicari R.: *Succeeding with Object Databases*. Wiley, New York, NY, 2001.
- [Dat 99] Date C. J.: *An Introduction to Database Systems*. Addison Wesley Longman, Boston, MA, 1999.
- [DDSS 95] DeFazio S., Daoud A., Smith L. A., Srinivasan J.: *Integrating IR and RDBMS Using Cooperative Indexing*. Proc. 18th ACM SIGIR Conference on Research and Development in Information Retrieval, 84-92, 1995.
- [Deu 96] Deutsch P.: RFC1951, *DEFLATE Compressed Data Format Specification*. <http://rfc.net/rfc1951.html>, 1996.
- [DK 02] Döller M., Kosch H.: *Enhancement of Oracle's Indexing Capabilities through GiST-implemented access methods*. University of Klagenfurt, Technical Report, No TR/ITEC/02/2.09, 2002.
- [DKD+ 02] Döller M., Kosch H., Dörflinger B., Bachlechner A., Blaschke G.: *Demonstration of an MPEG-7 Multimedia Data Cartridge*. Proc. ACM Multimedia Int. Conf. on Multimedia, 85 - 86, 2002.
- [DKO+ 85] DeWitt D. J., Katz R. H., Olken F., Shapiro L. D., Stonebraker M., Wood D. A.: *Implementation Techniques for Main Memory Database Systems*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 1-8, 1984.
- [Doh 98] Doherty C. G.: *Database Systems Management and Oracle8*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 510-511, 1998.
- [DS 00] Dittrich J.-P., Seeger B.: *Data Redundancy and Duplicate Detection in Spatial Join Processing*. In Proc. ICDE, 2000, 535-546.
- [DSTW 02] Dittrich J.-P., Seeger B., Taylor D. S., Widmayer P.: *Progressive Merge Join: A Generic and NON-Blocking Sort-Based Join Algorithm*. In Proc. VLDB, 2002, 299-310.
- [Ede 80] Edelsbrunner H.: *Dynamic Rectangle Intersection Searching*. Institute for Information Processing Report 47, Technical University of Graz, Austria, 1980.
- [EKSX 96] Ester M., Kriegel H.-P., Sander J., Xu X.: *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96), 226-231, 1996.
- [EM 97] Eiter T., Manilla H.: *Distance Measures for Point Sets and their Computation*. Acta Informatica, 34(2): 103-133, 1997.
- [EM 99] Eisenberg A., Melton J.: *SQL:1999, formerly known as SQL3*. ACM SIGMOD Record, 28(1), 131-138, 1999.

- [FDFH 00] Foley J. D., van Dam A., Feiner S. K., Hughes J. F.: *Computer Graphics: Principles and Practice*. Addison Wesley Longman, Boston, MA, 2000.
- [FEF+ 94] Faloutsos C., Equitz M., Flickner M., Niblack W., Petkovic D., Barber R.: *Efficient and Effective Querying by Image Content*. Journal of Intelligent Information Systems, 3:231-262, 1994.
- [FFS 00] Freytag J.-C., Flaszka M., Stillger M.: *Implementing Geospatial Operations in an Object-Relational Database System*. Proc. 12th Int. Conf. on Scientific and Statistical Database Management (SSDBM), 209-219, 2000.
- [FJ 03] Fonseca M. J., Jorge J. A.: *Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases*. Database Systems for Advanced Applications (DASFAA), 267-274, 2003.
- [FJM 97] Faloutsos C., Jagadish H. V., Manolopoulos Y.: *Analysis of the n-Dimensional Quadtree Decomposition for Arbitrary Hyperrectangles*. IEEE Trans. on Knowledge and Data Engineering 9(3), 373-383, 1997.
- [FR 89] Faloutsos C., Roseman S.: *Fractals for Secondary Key Retrieval*. Proc. ACM Symposium on Principles of Database Systems (PODS), 247-252, 1989.
- [FRM 94] Faloutsos C., Ranganathan M., Manolopoulos Y.: *Fast Subsequence Matching in Time-Series Databases*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 419-429, 1994.
- [Gae 95] Gaede V.: *Optimal Redundancy in Spatial Database Systems*. Proc. 4th Int. Symp. on Large Spatial Databases (SSD), LNCS 951, 96-116, 1995.
- [GG 98] Gaede V., Günther O.: *Multidimensional Access Methods*. ACM Computing Surveys 30(2), 170-231, 1998.
- [GHJV 95] Gamma E., Helm R., Johnson R., Vlissides J.: *Design Patterns*. Addison Wesley Longman, Boston, MA, 1995.
- [GLM 96] Gottschalk S., Lin M. C., Manocha D.: *OBBTree: A Hierarchical Structure for Rapid Interference Detection*. Proc. ACM SIGGRAPH Int. Conf. on Computer Graphics and Interactive Techniques, 171-180, 1996.
- [GM 93] Gary J. E., Mehrotra R.: *Similar Shape Retrieval using a Structural Feature Index*. Information Systems, Vol. 18, No. 7, 525-537, 1993.
- [GR 94] Gaede V., Riekert W.-F.: *Spatial Access Methods and Query Processing in the Object-Oriented GIS GODOT*. Proc. AGDM Workshop, Geodetic Commission, 1994.
- [Gri 92] Grieger I.: *Graphische Datenverarbeitung*. Springer Verlag, 1992.
- [GS 92] Garcia-Molina H., Salem K.: *Main Memory Database Systems: An Overview*. IEEE Trans. on Knowledge and Data Engineering 4(6), 509-516, 1992.
- [Gud 95] Gudivada V. N.: *Spatial Similarity Measures for Multimedia Applications*. Proc. Int. Conf. on Storage and Retrieval for Image and Video Databases (SPIE), 363-372, 1995.
- [Gün 99] Günther O.: *Looking Both Ways: SSD 1999 ± 10* . Proc. 6th Int. Symp. on Large Spatial Databases (SSD), LNCS 1651, 12-15, 1999.
- [Güt 94] Güting R. H.: *An Introduction to Spatial Database Systems*. VLDB Journal, 3(4), 357-399, 1994.

- [Gut 84] Guttman A.: *R-trees: A Dynamic Index Structure for Spatial Searching*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 47-57, 1984.
- [Had 04] Haddad A.: *Database Support for Haptic Simulation*. Diploma Thesis, 2004 (in German).
- [Hig 90] Higgins W. E.: *Automatic Analysis of 3D and 4D Radiological Images*. Grant application to the Department of Health and Human Services, Dec 1990.
- [HJR 97] Huang Y.-W., Jing N., Rundensteiner E. A.: *A Cost Model for Estimating the Performance of Spatial Joins Using R-trees*. Proc. 9th Int. Conf. on Scientific and Statistical Database Management (SSDBM), 30-38, 1997.
- [HNP 95] Hellerstein J. M., Naughton J. F., Pfeffer A.: *Generalized Search Trees for Database Systems*. Proc. 21st Int. Conf. on Very Large Databases (VLDB), 562-573, 1995.
- [HNSS 95] Haas P. J., Naughton J. F., Seshadri S., Stokes L.: *Sampling-Based Estimation of the Number of Distinct Values of an Attribute*. Proc. 21st Int. Conf. on Very Large Databases (VLDB), 311-322, 1995.
- [Hof 01] Hofmann R.: *Integration of Spatial Search into PDM Systems*. Diploma Thesis, University of Munich, 2001 (in German).
- [HS 93] Hellerstein J., Stonebraker M.: *Predicate Migration: Optimizing Queries with Expensive Predicates*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 267-276, 1993.
- [HS 95] Hjaltason G. R., Samet H.: *Ranking in Spatial Databases*. Proc. 4th Int. Symp. on Large Spatial Databases (SSD), LNCS 951, 83-95, 1995.
- [HSE+ 95] Hafner J., Sawhney H. S., Equitz W., Flickner M., Niblack W.: *Efficient Color Histogram indexing for Quadratic Form Distance Functions*. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 17, No. 7, 729-736, 1995.
- [Huf 52] Huffman D. A.: *A Method for the Construction of Minimum-Redundancy Codes*. Proc. Inst. Radio Engineers, 40(9), 1098-1101, 1952.
- [HYFK 98] Huang J., Yagel R., Filippov V., Kurzion Y.: *An Accurate Method for Voxelizing Polygon Meshes*. Proc. IEEE Symp. on Volume Visualization, 119-126, 1998.
- [IBM 98] IBM Corp.: *IBM DB2 Spatial Extender Administration Guide and Reference, Version 2.1.1*. Armonk, NY, 1998.
- [IBM 99] IBM Corp.: *IBM DB2 Universal Database Application Development Guide, Version 6*. Armonk, NY, 1999.
- [IGES 96] American National Standards Institute: *ANSI/US PRO/IPO 100-1996 (Initial Graphics Exchange Specification IGES 5.3)*. New York, NY, 1996.
- [Inf 98] Informix Software, Inc.: *DataBlade Developers Kit User's Guide, Version 3.4*. Menlo Park, CA, 1998.
- [Inf 99] Informix Software, Inc.: *Informix R-Tree Index User's Guide, Version 9.2*. Menlo Park, CA, 1999.
- [IWB 01] Digital Mock-up Process Simulation For Product Conception and Downstream Processes (Brite-Euram Project BRPR-CT95-0066), <http://www.iwb.tum.de/projekte/dmu-ps>.

- [Jag 90] Jagadish H. V.: *Linear Clustering of Objects with Multiple Attributes*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 332-342, 1990.
- [Jag 91] Jagadish H. V.: *A Retrieval Technique for Similar Shapes*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 208-217, 1991.
- [JB 91] Jagadish H. V., Bruckstein A. M.: *On sequential shape descriptions*. Pattern Recognition, 1991.
- [JD 88] Jain A. K., Dubes R. C.: *Algorithms for Clustering Data*. Prentice-Hall Inc., 1988.
- [JKP 03] Januzaj E., Kriegel H.-P., Pfeifle M.: *Towards Effective and Efficient Distributed Clustering*. International ICDM Workshop on Clustering Large Data Sets, 49-58, 2003.
- [JKP 04a] Januzaj E., Kriegel H.-P., Pfeifle M.: *A Quality Measure for Distributed Clustering*. Proc. International Conference on Databases and Applications (DBA), 2004.
- [JKP 04b] Januzaj E., Kriegel H.-P., Pfeifle M.: *DBDC: Density Based Distributed Clustering*. Proc. 9th International Conference on Extending Database Technology (EDBT), 2004.
- [JS 99] Jensen C. S., Snodgrass R. T.: *Temporal Data Management*. IEEE Trans. on Knowledge and Data Engineering 11(1), 36-44, 1999.
- [Kau 87] Kaufman A.: *An Algorithm for 3D Scan-Conversion of Polygons*. Proc. Eurographics, 197-208, 1987.
- [KB 95] Kornacker M., Banks D.: *High-Concurrency Locking in R-Trees*. Proc. 21st Int. Conf. on Very Large Databases (VLDB), 134-145, 1995.
- [KBJ+ 03] Kriegel H.-P., Brecheisen S., Januzaj E., Kröger P., Pfeifle M.: *Visual Mining of Cluster Hierarchies*. Proc. 3rd International ICDM Workshop on Visual Data Mining VDM@ICDM2003, 151-165, 2003.
- [KBK+ 03] Kriegel H.-P., Brecheisen S., Kröger P., Pfeifle M., Schubert M.: *Using Sets of Feature Vectors for Similarity Search on Voxelized CAD Objects*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 587-598, 2003.
- [KC 98] Kim K., Cha S. K.: *Sibling Clustering of Tree-based Spatial Indexes for Efficient Spatial Query Processing*. Proc. ACM CIKM Int. Conf. on Information and Knowledge Management, 398-405, 1998.
- [Kei 99] Keim D.: *Efficient Geometry-based Similarity Search of 3D Spatial Databases*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 419-430, 1999.
- [KF 92] Kamel I., Faloutsos C.: *Parallel R-trees*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 195-204, 1992.
- [KF 93] Kamel I., Faloutsos C.: *On Packing R-trees*. Proc. ACM CIKM Int. Conf. on Information and Knowledge Management, 490-499, 1993.
- [KKG 03] Kriegel H.-P., Kröger P., Gotlibovich I.: *Incremental OPTICS: Efficient Computation of Updates in a Hierarchical Cluster Ordering*. Proc. 5th Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK), 224-233, 2003.

- [KKM+ 03] Kriegel H.-P., Kröger P., Mashael Z., Pfeifle M., Pötke M., Seidl T.: *Effective Similarity Search on Voxelized CAD Objects*. Proc. 8th Int. Conf. on Database Systems for Advanced Applications (DASFAA), 27-36, 2003.
- [KKP 04] Kriegel H.-P., Kröger P., Pfeifle M.: *Efficient and Effective Computation of Hierarchical Clusters and their Representatives*. University of Munich, Technical Report, 2004.
- [KKPR 03a] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Acceleration of Relational Index Structures Based on Statistics*. Proc. 15th Int. Conf. on Scientific and Statistical Database Management (SSDBM), 2003.
- [KKPR 03b] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Efficient Query Processing on the Relational Quadtree*. V Simpósio Brasileiro de Geoinformática, 2003.
- [KKPR 04a] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Cost based Decompositioning of Complex Spatial Objects for Efficient Relational Indexing*. University of Munich, Technical Report, 2004.
- [KKPR 04b] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Statistic Driven Acceleration of Object-Relational Space-Partitioning Index Structures*. Proc. 9th Int. Conf. on Database Systems for Advanced Applications (DASFAA), 2004.
- [KKPR 04c] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Effective Decompositioning of Complex Spatial Objects into Intervals*. Proc. International Conference on Databases and Applications (DBA), 2004.
- [KKPR 04d] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Efficient Query Processing on Relational Data-Partitioning Index Structures*. Proc. 16th Int. Conf. on Scientific and Statistical Database Management (SSDBM), 2004.
- [KKPR 04e] Kriegel H.-P., Kunath P., Pfeifle M., Renz M.: *Spatial Join Processing for High-Resolution Objects*. Proc. 16th Int. Conf. on Scientific and Statistical Database Management (SSDBM), 2004.
- [KKPS 04] Kailing K., Kriegel H.-P., Pfeifle M., Schönauer S.: *Indexing of Complex Objects for Efficient Density-Based Clustering*. Submitted for publication.
- [KKP+ 03] Kriegel H.-P., Kunath P., Pfeifle M., Pötke M., Renz M., Strauß P.-M.: *Stochastic Driven Relational R-Tree*. V Simpósio Brasileiro de Geoinformática, 2003.
- [KKS 98] Kastenmüller G., Kriegel H.-P., Seidl T.: *Similarity Search in 3D Protein Databases*. Proc. German Conf. on Bioinformatics (GCB), Köln, Germany, 1998.
- [KKSS 04] Kailing K., Kriegel H.-P., Schönauer S., Seidl T.: *Efficient Similarity Search in Large Databases of Tree Structured Objects*. Proc. 20th International Conference on Data Engineering (ICDE), 2004.
- [Klu 04] Kluger M.: *The Filtering M-Tree*. Diploma Thesis, University of Munich, 2004 (in German).
- [KMH 97] Kornacker M., Mohan C., Hellerstein J. M.: *Concurrency Control in Generalized Search Trees*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 62-72, 1997.

- [KMPS 01a] Kriegel H.-P., Müller A., Pötke M., Seidl T.: *DIVE: Database Integration for Virtual Engineering* (Demo). Demo Proc. 17th Int. Conf. on Data Engineering (ICDE), 15-16, 2001.
- [KMPS 01b] Kriegel H.-P., Müller A., Pötke M., Seidl T.: *Spatial Data Management for Computer Aided Design* (Demo). Proc. ACM SIGMOD Int. Conf. on Management of Data, 614, 2001.
- [Kor 99] Kornacker M.: *High-Performance Extensible Indexing*. Proc. 25th Int. Conf. on Very Large Databases (VLDB), 699-708, 1999.
- [KP 92] Keim D. A., Prawirohardjo E. S.: *Datenbankmaschinen – Performanz durch Parallelität*. Reihe Informatik 86, BI Wissenschaftsverlag, Mannheim, 1992.
- [KPP+ 03] Kriegel H.-P., Pfeifle M., Pötke M., Renz M., Seidl T.: *Spatial Data Management for Virtual Product*, Lecture Notes in Computer Science, Springer, Vol. 2598, 216-230, 2003.
- [KPP+ 04] Kriegel H.-P., Pfeifle M., Pötke M., Seidl S., Enderle J.: *Object-Relational Spatial Indexing*. In: Manolopoulos Y., Papadopoulos A., Vassilakopoulos M. (eds.): *Spatial Databases: Technologies, Techniques and Trends*. Idea Group Inc., 2004.
- [KPPS 02] Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *A Cost Model for Interval Intersection Queries on RI-Trees*. Proc. 14th Int. Conf. on Scientific and Statistical Database Management, 131-141, 2002.
- [KPPS 03a] Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *Spatial Query Processing for High Resolutions*. Proc. 8th Int. Conf. on Database Systems for Advanced Applications (DASFAA), 17-26, 2003.
- [KPPS 03b] Kriegel H.-P., Pfeifle M., Pötke M., Seidl S.: *The Paradigm of Relational Indexing: a Survey*. Proc. 10. GI-Fachtagung Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 285-304, 2003.
- [KPPS 04] Kriegel H.-P., Pfeifle M., Pötke M., Seidl S.: *A Cost Model for Spatial Intersection Queries on RI-Trees*. Proc. 9th Int. Conf. on Database Systems for Advanced Applications (DASFAA), 2004.
- [KPS 00] Kriegel H.-P., Pötke M., Seidl T.: *Managing Intervals Efficiently in Object-Relational Databases*. Proc. 26th Int. Conf. on Very Large Databases (VLDB), 407-418, 2000.
- [KPS 01] Kriegel H.-P., Pötke M., Seidl T.: *Interval Sequences: An Object-Relational Approach to Manage Spatial Data*. Proc. 7th Int. Symposium on Spatial and Temporal Databases (SSTD), LNCS 2121, 481-501, 2001.
- [KS 97] Katayama N., Satoh S.: *The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 369-380, 1997.
- [KS 03] Kriegel H.-P., Schönauer S.: *Similarity Search in Structured Data*, Proc. 5th Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK), 309-319, 2003.
- [KSF+ 96] Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z.: *Fast Nearest Neighbor Search in Medical Image Databases*. Proc. 22nd Int. Conf. on Very Large Databases (VLDB), 215-226, 1996.

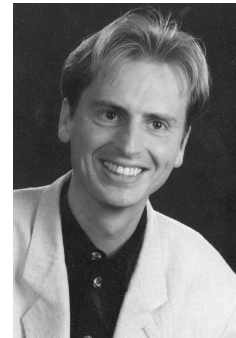
- [KSS 97] Kriegel H.-P., Schmidt T., Seidl T.: *3D Similarity Search by Shape Approximation*. Proc. Fifth Int. Symposium on Large Spatial Databases (SSD'97), LNCS 1262, 11-28, 1997.
- [Kuh 55] Kuhn H.W.: *The Hungarian Method for the Assignment Problem*. Naval Research Logistics Quarterly, 2:83-97, 1955.
- [Kun 02] Kunath P.: *Compression of CAD data*. Diploma Thesis, University of Munich, 2002.
- [Lei 04] Leis F.: *Efficient Similarity Search on Voxelized CAD Objects*. Diploma Thesis, University of Munich, 2003.
- [Lib 01] Libkin L.: *Expressive Power of SQL*. Proc. 8th Int. Conf. on Database Theory (ICDT), 1-21, 2001.
- [LJF 94] Lin K., Jagadish H. V., Faloutsos C.: *The TV-Tree: An Index Structure for High-Dimensional Data*. Int. Journal on Very Large Data Bases (VLDB Journal), Vol. 3, No. 4, 517-542, 1994.
- [LL 98] Leutenegger S. T., Lopez M. A.: *The Effect of Buffering on the Performance of R-Trees*. Proc. 14th Int. Conf. on Data Engineering (ICDE), 164-171, 1998.
- [LNS 90] Lipton R. J., Naughton J. F., Schneider A. D.: *Practical Selectivity Estimation through Adaptive Sampling*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 1-11, 1990.
- [Lom 98] Lomet D.: *B-tree Page Size When Caching is Considered*. ACM SIGMOD Record, 27(3), 28-32, 1998.
- [LR 94] Lo M-L., Ravishankar C.V.: *Spatial Joins Using Seeded Trees*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 209-220, 1994.
- [LR 96] Lo M-L., Ravishankar C.V.: *Spatial Hash-Joins*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 247-258, 1996.
- [LS 02] Lang C., Singh A.: *Accelerating High-Dimensional Nearest Neighbor Queries*. Proc. 14th Int. Conf. on Scientific and Statistical Database Management (SSDBM), 109-119, 2002.
- [LSW 99] Lennerz C., Schömer E., Warken T.: *A Framework for Collision Detection and Response*. Proc. 11th European Simulation Symposium (ESS), 309-314, 1999.
- [LZ 77] Lempel A., Ziv J.: *A Universal Algorithm for Sequential Data Compression*. IEEE Transactions on Information Theory, Vol. IT-23, No. 3, 337-343, 1977.
- [Mai 03] Maier H.: *Statistic-Driven Acceleration of the Relational Interval Tree*. Diploma Thesis, 2003 (in German).
- [McQ 67] McQueen J.: *Some Methods for Classification and Analysis of Multivariate Observations*. 5th Berkeley Symp. Math. Statist. Prob., volume 1, 281-297, 1967.
- [MG 93] Mehrotra R., Gray J.: *Feature-Based Retrieval of Similar Shapes*. Proc. 9th Int. Conf. on Data Engineering (ICDE), 108-115, 1993.
- [MG 95] Mehrotra R., Gary J. E.: *Feature-Index-Based Similar Shape Retrieval*. Proc. 3rd Working Conf. on Visual Database Systems, 1995.
- [MH 99] Möller T., Haines E.: *Real-Time Rendering*. A K Peters, Natick, MA, 1999.

- [MJFS 96] Moon B., Jagadish H. V., Faloutsos C., Saltz J. H.: *Analysis of the Clustering Properties of Hilbert Space-filling Curve*. Techn. Report CS-TR-3611, University of Maryland, 1996.
- [MP 94] Medeiros C. B., Pires F.: *Databases for GIS*. ACM SIGMOD Record, 23(1), 107-115, 1994.
- [MPT 99] McNeely W. A., Puterbaugh K. D., Troy J. J.: *Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling*. Proc. ACM SIGGRAPH Int. Conf. on Computer Graphics and Interactive Techniques, 401-408, 1999.
- [MTT 00] Manolopoulos Y., Theodoridis Y., Tsotras V. J.: *Advanced Database Indexing*. Kluwer, Boston, MA, 2000.
- [Müh 03] Mühlbauer W.: *Statistic-Driven Acceleration of the Relational Quadtree*. Diploma Thesis, 2003 (in German).
- [Mun 57] Munkres J.: *Algorithms for the Assignment and Transportation Problems*. Journal of the SIAM, 6:32-38, 1957.
- [NBE+ 93] Niblack W., Barber R., Equitz W., Flickner M., Glasmann E., Petkovic D., Yanker P., Faloutsos C., Taubin G.: *The QBIC Project: Querying Images by Content Using Color, Texture, and Shape*. SPIE 1993 Int. Symposium on Electronic Imaging: Science and Technology Conference 1908, Storage and Retrieval for Image and Video Databases, 1993.
- [NS 86] Newmann W. M., Sproull R. F.: *Grundzüge der interaktiven Computergraphik*. McGraw-Hill Book Company GmbH Hamburg, 1986.
- [NS 87] Nelson R.C., Samet H.: *A population analysis for hierarchical data structures*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 270-277, 1987.
- [NW 97] Nevill-Manning C., Witten I.: *Compression and Explanation using Hierarchical Grammars*. The Computer Journal 40(2/3): 103-116, 1997.
- [OM 84] Orenstein J. A., Merrett T. H.: *A Class of Data Structures for Associative Searching*. Proc. 3rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS), 181-190, 1984.
- [OM 88] Orenstein J. A., Manola F. A.: *PROBE Spatial Data Modeling and Query Processing in an Image Database Application*. IEEE Transactions on Software Engineering, 14(5), 611-629, 1988.
- [Ora 99a] Oracle Corp.: *Oracle8i Data Cartridge Developer's Guide, Release 2 (8.1.6)*. Redwood Shores, CA, 1999.
- [Ora 99b] Oracle Corp.: *Oracle Spatial User's Guide and Reference, Release 8.1.6*. Redwood Shores, CA, 1999.
- [Ora 99c] Oracle Corp.: *Oracle8i Concepts, Release 8.1.6*. Redwood Shores, CA, 1999.
- [Ora 00] Oracle Corp.: *Oracle8i Appliance – An Oracle White Paper*. Redwood Shores, CA, 2000.
- [Ore 86] Orenstein J.A.: *Spatial Query Processing in an Object-Oriented Database System*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 326-336, 1986.
- [Ore 89] Orenstein J. A.: *Redundancy in Spatial Databases*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 294-305, 1989.

- [OTYB 00] Ooi B. C., Tan K.-L., Yu C., Bressan S.: *Indexing the Edges - A simple and yet efficient approach to high-dimensional indexing*. ACM Symposium on Principles of Database Systems (PODS), 166-174, 2000.
- [PD 96] Patel J.M., DeWitt D.J.: *Partition Based Spatial-Merge Join*. Proc. of the ACM SIGMOD Conference, 259-270, 1996.
- [PF 99] Proietti G., Faloutsos C.: *I/O Complexity for Range Queries on Region Data Stored Using an R-tree*. Proc. 15th Int. Conf. on Data Engineering (ICDE), 628-635, 1999.
- [Pfe 01] Pfeifle M.: *Object-Relational Management of High-Resolution CAD Databases*. Diploma Thesis, University of Munich, 2001.
- [Pöt 01] Pötke M.: *Spatial Indexing for Object-Relational Databases*. Ph.D. Thesis, Institute for Computer Science, University of Munich, 2001.
- [PRS 99] Papadopoulos A.N., Rigaux P., Scholl M.: *A Performance Evaluation of Spatial Join Processing Strategies*. Proc. of the Symposium on Large Spatial Databases (SSD), 286-307, 1999.
- [PS 93] Preparata F. P., Shamos M. I.: *Computational Geometry: An Introduction*. 5th ed., Springer, 1993.
- [Ram 97] Ramaswamy S.: *Efficient Indexing for Constraint and Temporal Databases*. Proc. 6th Int. Conf. on Database Theory, LNCS 1186, 419-431, 1997.
- [RB 00] Ramon J., Bruynooghe M.: *A polynomial time computable metric between point sets*. Technical Report CW 301, Katholieke Universiteit Leuven, 2001.
- [RG 02] Roelofs G., Gailly J.: *zlib, Technical Details*. http://www.gzip.org/zlib/zlib_tech.html, 2002.
- [RH 93] Roth M., Van Horn S.: *Database Compression*. SIGMOD Record 22(3): 31-39, 1993.
- [RKV 95] Roussopoulos N., Kelley S., Vincent F.: *Nearest Neighbor Queries*. Proc. Int. Conf. on Management of Data (SIGMOD), 71-79, 1995.
- [RMF+ 00] Ramsak F., Markl V., Fenk R., Zirkel M., Elhardt K., Bayer R.: *Integrating the UB-Tree into a Database System Kernel*. Proc. 26th Int. Conf. on Very Large Databases (VLDB), 263-272, 2000.
- [RRSB 99] Ravi Kanth K. V., Ravada S., Sharma J., Banerjee J.: *Indexing Medium-dimensionality Data in Oracle*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 521-522, 1999.
- [RS 99] Ravada S., Sharma J.: *Oracle8i Spatial: Experiences with Extensible Databases*. Proc. 6th Int. Symp. on Large Spatial Databases (SSD), LNCS 1651, 355-359, 1999.
- [SAC+ 79] Selinger P. G., Astrahan M. M., Chamberlin D. D., Lorie R. A., Price T. G.: *Access Path Selection in a Relational Database Management System*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 23-34, 1979.
- [Sam 90a] Samet H.: *The Design and Analysis of Spatial Data Structures*. Addison Wesley Longman, Boston, MA, 1990.
- [Sam 90b] Samet H.: *Applications of Spatial Data Structures*. Addison Wesley Longman, Boston, MA, 1990.

- [SB 98] Stonebraker M., Brown P.: *Object-relational DBMSs – Tracking the Next Great Wave*. Morgan Kaufmann, San Francisco, CA, 1998.
- [Sch 04] Schmitt O.: *The Relational Filtering M-tree*. Diploma Thesis, University of Munich, 2004 (In German).
- [SDF+ 00] Jagannathan Srinivasan, Souripriya Das, Chuck Freiwald, Eugene Inseok Chong, Mahesh Jagannath, Aravind Yalamanchi, Ramkumar Krishnan, Anh-Tuan Tran, Samuel DeFazio, Jayanta Banerjee: *Oracle8i Index-Organized Table and Its Application to New Domains*. Proc. 26th Int. Conf. on Very Large Databases (VLDB), 285-296, 2000.
- [Sel 88] Sellis T. K.: *Multiple-Query Optimization*. ACM Transactions on Database Systems (TODS), 13(1), 23-52, 1988.
- [SH 94] Sawhney H., Hafner J.: *Efficient Color Histogram Indexing*. Proc. Int. Conf. on Image Processing, 66-70, 1994.
- [SJS 01] Slivinskas G., Jensen C. S., Snodgrass R. T.: *Adaptable Query Optimization and Evaluation in Temporal Middleware*. Proc. ACM SIGMOD ACM SIGMOD Int. Conf. on Management of Data, 127-138, 2001.
- [SK 93] Schiwietz M., Kriegel H.-P.: *Query Processing of Spatial Objects: Complexity versus Redundancy*. Proc. 3rd Int. Symp. on Large Spatial Databases (SSD), LNCS 692, 377-396, 1993.
- [SK 98] Seidl T., Kriegel H.-P.: *Optimal Multi-Step k-Nearest Neighbor Search*. Proc. ACM SIGMOD Int. Conf. on Management of Data, 154-165, 1998.
- [SKSH 89] Schneider R., Kriegel H.-P., Seeger B., Heep S.: *Geometry-based Similarity Retrieval of Rotational Parts*. Proc. Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering, Gaithersburg, MD, 150-160, 1989.
- [SLF 93] Seeger B., Larson P., McFadyen R.: *Reading a Set of Disk Pages*. Proc. 19th Int. Conf. on Very Large Databases (VLDB): 592-603, 1993.
- [SMS+ 00] Srinivasan J., Murthy R., Sundara S., Agarwal N., DeFazio S.: *Extensible Indexing: A Framework for Integrating Domain-Specific Indexing Schemes into Oracle8i*. Proc. 16th Int. Conf. on Data Engineering (ICDE), 91-100, 2000.
- [SN 02] Steinmetz R., Nahrstedt K.: *Multimedia Fundamentals, Volume 1: Media Coding and Content Processing*. Second Edition. Prentice Hall, 110-119, 2002.
- [Sno 00] Snodgrass R. T.: *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, San Francisco, CA, 2000.
- [SQL 92] American National Standards Institute: *ANSI X3.135-1992/ISO 9075-1992 (SQL-92)*. New York, NY, 1992.
- [SQL 99] American National Standards Institute: *ANSI/ISO/IEC 9075-1999 (SQL:1999, Parts 1-5)*. New York, NY, 1999.
- [SQL+ 03] Sander J., Qin X., Lu Z., Niu N., Kovarsky A.: *Automatic Extraction of Clusters from Hierarchical Clustering Representations*. Proc. 7th Pacic-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2003.

- [STEP 94+] International Organization for Standardization: *ISO 10303, Industrial automation systems and integration – Product data representation and exchange (STEP, Parts 1-520)*. Geneva, 1994-2001.
- [Sto 86] Stonebraker M.: *Inclusion of New Types in Relational Data Base Systems*. Proc. 2nd Int. Conf. on Data Engineering (ICDE), 262-269, 1986.
- [Str 04] Strauß P.-M.: *Statistic-Driven Acceleration of the Relational R-tree*. Diploma Thesis, University of Munich, 2003.
- [Tam 04] Tamecke L.: *Clustering Moving Objects*. Diploma Thesis, University of Munich, 2004 (in German).
- [TC 91] Taubin G., Cooper D. B.: *Recognition and Positioning of Rigid Objects Using Algebraic Moment Invariants*. Geometric Methods in Computer Vision Vol. 1570, SPIE, 175-186, 1991.
- [TCG+ 93] Tansel A. U., Clifford J., Gadia S., Jajodia S., Segev A., Snodgrass R.: *Temporal Databases: Theory, Design and Implementation*. Redwood City, CA, 1993.
- [TH 81] Tropf H., Herzog H.: *Multidimensional Range Search in Dynamically Balanced Trees*. Angewandte Informatik, 81(2), 71-77, 1981.
- [TSS 00] Theodoridis Y., Stefanakis E., Sellis T.: *Efficient Cost Models for Spatial Queries Using R-Trees*. IEEE Transactions on Knowledge and Data Engineering, 12(1), 19-32, 2000.
- [TTSF 00] Traina C. Jr., Traina A., Seeger B., Faloutsos C.: *Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes*. Int. Conf. on Extending Database Technology (EDBT), 51-65, 2000.
- [VDA 87] Verband der Automobilindustrie: *VDA-Flächenschnittstelle VDAFS, Version 2.0*. Frankfurt a. M., 1987 (in German).
- [Vei 03] Veith M.: *Decompositioning of Voxalized CAD Objects*. Diploma Thesis, University of Munich, 2003 (in German).
- [Vie 03] Viermetz M.: *BOSS: Browsing Optics-Plots for Similarity Search*. Diploma Thesis, University of Munich, 2003.
- [Vin 91] Vincent L.: *New Trends in Morphological Algorithms*. SPIE Proceedings on Non-linear Image Processing II, San Jose, CA, February 1991.
- [Wan 91] Wang F.: *Relational-Linear Quadtree Approach for Two-Dimensional Spatial Representation and Manipulation*. IEEE Trans. on Knowledge and Data Engineering 3(1), 118-122, 1991.
- [WJ 96] White D. A., Jain R.: *Similarity Indexing: Algorithms and Performance*. Proc. of Storage and Retrieval for Image and Video Databases IV, SPIE Proc. 2670, 62-73, 1996.
- [WSB 98] Weber R., Schek H.-J., Blott S.: *A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces*. Proc. Int. Conf. on Very Large Data Bases (VLDB), New York City, 194-205, 1998.
- [YOTJ 01] Yu C., Ooi B. C., Tan K.-L., Jagadish H. V.: *Indexing the Distance: An Efficient Method to KNN Processing*. Proc. 27th Int. Conf. on Very Large Databases (VLDB), 421-430, 2001.



Curriculum Vitae

Martin Pfeifle was born on December 15, 1967 in Freudenstadt, Germany. After attending primary school from 1974 to 1978, and high-school from 1978 to 1987, he served in the military service from 1987 to 1988 in Marburg.

From 1988 to 1991 he attended the University of Cooperative Education in Stuttgart, studying Electrical Engineering. During this time he attended several internships at the Bosch GmbH, Stuttgart, culminating in his diploma thesis on “Model based Measuring of Ignition Variables” (written in German).

From 1991 to 2004, he continuously worked as a software engineer and a consultant for different companies. For more than seven years he has been working for a company, called iff, which produces software for the wood-cutting industry.

From 1990 to 2001, he studied computer science with a minor in mathematics at the Fernuniversität Hagen. His diploma thesis was on “*High Resolution Indexing for CAD Databases*”, supervised by Prof. Dr. Ralf Hartmut Güting, Prof. Dr. Hans-Peter Kriegel, Prof. Dr. Thomas Seidl and Dr. Marco Pötke.

In 2001, 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 systems at the Institute for Computer Science. The research interests of Martin Pfeifle include database support for virtual engineering, with a strong emphasis on spatial index structures and similarity search in spatial databases. Furthermore, he is interested in the area of knowledge discovery in databases, especially in density-based clustering.

