Cost-based query-adaptive clustering for multidimensional objects with spatial extents

Cristian-Augustin Saita

THÈSE / PhD THESIS

présentée à

L'UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité

Informatique

soutenue par

Cristian-Augustin SAITA

le 13 Janvier 2006

Titre

Groupement d'objets multidimensionnels étendus avec un modèle de coût adaptatif aux requêtes

(Cost-based query-adaptive clustering for multidimensional objects with spatial extents)

Directeur de thèse: **Philippe Pucheral**
Encadrant de thèse: **François Llirbat**

Jury

Michel Scholl	Professeur des universités, CNAM-Paris	*rapporteur*
Patrick Valduriez	Directeur de recherche, INRIA-Rennes	*rapporteur*
Jean-Marc Saglio	Directeur d'études, ENST-Paris	*examinateur*
Ioana Manolescu	Chercheur, INRIA-Futurs Orsay	*examinateur*
Philippe Pucheral	Professeur, Université de Versailles	*directeur de thèse*
François Llirbat	Chercheur, INRIA-Rocquencourt	*encadrant de thèse*

To my dear parents, and to my wonderful sister Eliza

Acknowledgements

I wish to express my most sincere gratitude and appreciation to many people who made this PhD thesis possible.

I am highly indebted to Doctor François Llirbat, my supervisor, for accepting me to carry out research in his area, for his guidance, thought-provoking ideas, encouragement and support at all levels. Special thanks are due to Professor Philippe Pucheral, my PhD director, who offered me much-appreciated advice and support throughout all my work.

I would like to thank the members of my PhD committee who took effort in reading my dissertation and provided me with valuable comments and suggestions: Professor Michel Scholl, Professor Patrick Valduriez, Professor Jean-Marc Saglio, Doctor Ioana Manolescu, Professor Philippe Pucheral and Doctor François Llirbat.

Many thanks are due to the members of the Caravel Project at INRIA-Rocquencourt: Eric Simon, François Llirbat, Françoise Fabret, Helena Galhardas, Ioana Manolescu, Joao Pereira, and Daniela Florescu. They welcomed me in a friendly scientific environment; they motivated and supported me to pursue my studies and to go on with my PhD thesis. Special thanks to the senior members of the SMIS INRIA Project, where I had the opportunity to finish my PhD work: Philippe Pucheral, Beatrice Finance, and Luc Bouganim; and to Elisabeth Baque, our project assistant, who helped me with many administrative issues and friendly thoughts.

I wish to express my appreciation to my colleagues at INRIA, dear and wonderful friends: Sophie Giraud, François Dang-Ngoc, Nicolas Dieu, Cosmin Cremarenco, Adrian Dragusanu, Lucian Precup, and Aurelian Lavric; and to my trustful and wonderful friends from the University of Versailles: Tao Wan and Mathieu Decore, Xiaohui Xue and Sha Fei, Veronika Peralta, and Clement Jamard. Thank you so much for your help and for the great moments we had together.

I would like to thank all my teachers. More particularly, I express my sincere gratitude to Professor Irina Athanasiu, from the Polytechnic University of Bucharest.

I dedicate this thesis to my family to whom I owe a lot: my parents, Petre and Augustina Saita, my sister, Eliza, and his husband, Dragos Georgescu.

Last, I wish to mention the many people and dear friends I had the chance to meet since I came in France: Lucian Precup and Celine Schawann, Adrian Dragusanu and Magda Ciortan, Cosmin Cremarenco and Diana Radu, Lilan Wu and Raoul, Farida Ibersiene, Liliana Cucu and Samuel, Minel Ferecatu and Ioana, Radu Pop, Florin Dragan, Corina Ferdean, Nicoleta Preda, Daniel Calegari, Saida Medjdoub, Sonia Guehis, Cristophe Schmitz, Cristophe Salperwyck, François Boisson, Adi Citu and Loredada, Dan and Flori Olteanu, Gabriel Kaltman, Iulia Comsa, Iulian Mihai, Sorana Turcu, Khaled Yagoub, Jean-Pierre Matsumoto, Mokrane Amzal, Dinial Bensallah, Didier Lemaire, and Sebastien Combaz.

Cristian Saita

April 3rd, 2006
Versailles, France

Contents

1 Introduction **1**

1.1 Motivating applications 2

1.2 Challenge for the multidimensional indexing 4

1.3 Contributions 8

1.4 Organization 10

2 Problem specification and related work **11**

2.1 Problem specification 11

2.1.1 Data representation model 11

2.1.2 Problem definition 12

2.1.3 Requirements 13

2.2 State-of-the-art 14

2.2.1 Classification of multidimensional indexing methods 14

2.2.2 Space partitioning methods 15

2.2.3 Region bounding methods 19

2.2.4 Dynamic R-tree versions 24

2.2.5 Static R-tree versions 27

2.2.6 Cost-based R-tree extensions 27

2.2.7 Limitations of region bounding methods 34

2.2.8 Conclusions 38

2.3 Principles and characteristics of our clustering solution 40

2.4 Conclusions 44

3 Cost-based query-adaptive clustering **46**

3.1 Preliminary definitions 46

3.2 Clustering strategy 48

3.2.1 Outline of the clustering strategy 48

3.2.2 Application of the clustering strategy 49

3.3 Object grouping criterion . . . 51
3.3.1 Multidimensional space data representation model . . . 51
3.3.2 Generic grouping model for multidimensional spatial objects . . . 52
3.3.3 Cluster signature and cluster subsignature . . . 54
3.3.4 Object grouping methods . . . 56
3.3.5 Clustering function . . . 60
3.4 Cost model and benefit functions . . . 63
3.4.1 System performance parameters . . . 63
3.4.2 Cluster performance indicators . . . 64
3.4.3 Cost model and database storage scenarios . . . 65
3.4.4 Benefit functions supporting the clustering strategy . . . 68
3.5 Conclusions . . . 70

4 Database clustering and manipulation algorithms 73
4.1 Notations . . . 73
4.2 Algorithms for database clustering operations . . . 74
4.2.1 Cluster restructuring invocation . . . 75
4.2.2 Cluster merge procedure . . . 78
4.2.3 Cluster split procedure . . . 80
4.3 Algorithms for database manipulation operations . . . 83
4.3.1 Spatial query execution . . . 83
4.3.2 Data object insertion . . . 86
4.3.3 Data object deletion . . . 87
4.4 Conclusions . . . 88

5 Implementation and performance evaluation 91
5.1 Implementation considerations . . . 91
5.1.1 Database storage management . . . 91
5.1.2 Data structures supporting our clustering solution . . . 93
5.2 Performance evaluation . . . 97
5.2.1 Experimental setup . . . 98
5.2.2 Search performance evaluation . . . 102
5.2.3 Behavior and adaptability study . . . 112
5.3 Conclusions . . . 118

6 Conclusions and perspectives 120
6.1 Summary . . . 120

6.2 Perspectives 122

7 Resumé en Français – French Summary **124**

7.1 Introduction 124

7.1.1 Motivation du travail 125

7.1.2 Exigences de performance 126

7.1.3 Formulation du problème 127

7.1.4 Contributions 128

7.1.5 Organisation de la thèse 129

7.2 Problème et état de l'art 130

7.2.1 Spécification du problème 130

7.2.2 Etat de l'art 131

7.3 Groupement en clusters avec un modèle de coût adaptatif aux requêtes . . 134

7.3.1 Définition du cluster 135

7.3.2 Processus de groupement en clusters 136

7.3.3 Fonctions supportant la stratégie de groupement 137

7.4 Algorithmes d'organisation et de manipulation de la base d'objets 138

7.4.1 Restructuration des clusters 138

7.4.2 Exécution des requêtes spatiales 142

7.4.3 Opérations de mise à jour d'objets 144

7.5 Implémentation et évaluation expérimentale 146

7.5.1 Gestion de la mémoire et de l'espace de stockage 146

7.5.2 Evaluation expérimentale 147

7.6 Conclusions et perspectives de recherche 147

List of Figures

2.1 Representation of a range subscription as a multidimensional spatial object 12
2.2 An R-tree example 21
2.3 The resulting R-tree after insertion of $R10$ 21
2.4 $MaxO$ threshold value with increasing disk page size 37
2.5 Overlap between MBRs of nodes resulting from directory node splits . . . 38
2.6 Spatial access methods: characteristics and limitations 39
2.7 Example illustrating the rationales of our clustering approach 43

3.1 Multidimensional data space representation model 52
3.2 Similar intervals 53
3.3 Similar multidimensional spatial objects 54
3.4 An example to illustrate some possible clusters of spatial objects 55
3.5 Object grouping based on space division using all dimensions 57
3.6 Object grouping based on space division using one dimension 59

4.1 Cluster restructuring invocation 75
4.2 Recursive cluster merges 76
4.3 Recursive cluster splits 77
4.4 Cluster merge procedure 79
4.5 Cluster split procedure 81
4.6 Spatial query execution algorithm 84
4.7 Recursively determine the clusters to be explored 85
4.8 Object insertion procedure 86
4.9 Object deletion procedure 88

5.1 Data structures enabling the implementation of our clustering approach . 93
5.2 Possible representations for a cluster signature 94
5.3 Search performance in Memory for different query selectivities (uniform data and query objects) 103

5.4 Search performance on Disk for different query selectivities (uniform data and query objects) . . . 104
5.5 Search performance in Memory for different space dimensionalities (skewed data) . . . 107
5.6 Search performance on Disk for different space dimensionalities (skewed data) . . . 108
5.7 Search performance in Memory with regions of interest (spatial data and query skewness) . . . 110
5.8 Search performance on Disk with regions of interest (spatial data and query skewness) . . . 111
5.9 Performance evolution in Memory when dynamically changing query selectivity . . . 113
5.10 Performance evolution on Disk when dynamically changing query selectivity 114
5.11 Behavior of Adaptive Clustering in Memory when dynamically changing data distribution . . . 116
5.12 Behavior of Adaptive Clustering on Disk when dynamically changing data distribution . . . 117

7.1 Restructuration des clusters . . . 139
7.2 Fusion de cluster . . . 140
7.3 Eclatement de cluster . . . 141
7.4 Exécution des requêtes spatiales . . . 143
7.5 Insertion d'objet . . . 144
7.6 Suppression d'objet . . . 146

List of Tables

2.1 Average performance parameters of our execution platform 35
2.2 System parameters and $MaxO$ with increasing disk page size 36

4.1 Notations 75

5.1 Average I/O and CPU operations costs 99
5.2 Overlap degree between regions of interest 109
5.3 Cost comparison for different data organization methods 115

Chapter 1

Introduction

The nowadays availability of high-speed, low-priced Internet access has globally led to an explosion in the number of users and applications on-line. About 15 out of every 100 persons around the world use the Internet today, and more than half a million people each week are choosing broadband technology to gain permanent, high-speed Internet connection [Int05]. With the Internet expansion, the number of on-line information sources is permanently increasing. The World Wide Web (Web) has become a huge repository of information in practically all domains of activity. In addition to enormous quantities of static and long lasting data, larger and larger amounts of dynamic information are nowadays deployed on the Web. Thanks to the large-scale high-speed connectivity, the Web is now particularly well-suited for volatile, changing, and time-varying information. With the accelerated penetration of the broadband technology, such information can be instantly made available to millions of users and applications on-line. In this context, monitoring the Web to track the new developments in a particular domain of interest turns to be, for common users, an uneasy, time consuming task. The manifesting need to facilitate user access to relevant information has become a critical concern. This concern has triggered an accelerated development of a new class of applications focusing on Selective Dissemination of Information (SDI applications) [AF00, FJL+01, Per02, LJ04].

Information dissemination applications involve timely delivery of relevant information from and to large sets of registered clients such as users or applications. The entities producing information are referred to as producers or publishers. The information is delivered in packets called publications. The publications are emitted by publishers at different time moments referred to as publication events. The entities interested in the disseminated information are referred to as consumers or subscribers. The subscribers express their interests in certain publications by means of subscriptions. The interaction between publishers and subscribers is assured by the application system referred to as "publish and subscribe system". The role of the system is to manage the collection of subscriptions, and to react to the incoming publication events by retrieving the matching subscriptions and by delivering the relevant data to the concerned subscribers, in a timely manner. Depending on the client needs and requirements, the data can be delivered either periodically, or according to some delivery schedules, as fast as possible, or even instantly in critical and real-time applications. Common examples of SDI applications are related to digital libraries of publications, on-line stock trading and exchange, auctions, news, forums, small ads and advertising. More complex SDI applications are

deployed in distributed network management systems, alerting, alarm, awareness and notification systems, Enterprise Application Integration systems (EAI), financial security and exchange systems, search engine marketing and service delivery systems.

Many of the new emerging information dissemination applications involve large quantities of multidimensional data objects such as subscriptions and publications. The characteristics of the data objects manipulated in these applications, and the performance requirements that should be met, bring out new real challenges for the multidimensional data indexing. These challenges have become the main motivation of our work.

1.1 Motivating applications

To illustrate the motivation of our work, we next present two application examples from the SDI domain, showing some specific data characteristics and requirements that should be met by our target applications.

Small ads notification system

The first application consists of a publish and subscribe notification system dealing with small ads. For simplicity, we only consider demands and offers concerning apartments for rent in the area of Newark. Here is an example of subscription corresponding to a small ads demand: "Notify me of all new apartments within 30 miles from Newark, with a rent price between 400$ and 700$, having between 3 and 5 rooms, and 2 bathrooms". In this subscription most attributes specify range intervals instead of single values: 0-30 as distance from Newark, 400-700 as rent price, 3-5 as number of rooms.

In general, range subscriptions are more suitable for publish and subscribe systems: First, because it is neither always possible, nor appropriate, to provide exact matching values for all the subscription attributes. Looking for an apartment with an exactly-matching rent price would not make too much sense. Second, because the probability of finding offers that match exactly and simultaneously all the attribute values is very low. Rather than having no matching offer, the subscribers commonly prefer to consult alternative offers, not exactly matching, but close enough to their wishes. By means of range intervals, the subscribers can express much more flexible matching criteria, being enabled to establish acceptable ranges of variation for the values of the subscription attributes, according to their expectations. Therefore, range subscriptions are commonly employed in many similar application cases [LJ04].

In the small ads application, potentially high rates of small ads offers, regarded as incoming publication events emitted by publishers, need to be verified against the subscription database. The publication objects can specify either single values, or range intervals, for their attributes. An example of single-valued small ads publication is: "Rent apartment, 10 miles from Newark, 3 rooms, 1 bathroom, 650$ a month". An example of range publication is: "Apartments for rent in Newark: 3 to 5 rooms, 1 or 2 bathrooms, 600-900 a month". The system's job is to retrieve the subscribers interested in the incoming offers in order to notify them in a timely manner.

Data characteristics and performance requirements: Both subscription and publication objects can be represented as sets of *<attribute, interval of values>* associations. The subscriptions matching an incoming publication are those whose intervals of values overlap the publication's intervals for all the corresponding attributes. Retrieving the qualifying subscriptions is equivalent to answering a multi-interval intersection query between the incoming publication object and the subscription database.

Considering the number of potential clients such as Internet users or application agents, this type of application could involve a large number of range subscriptions with many attributes. Insertions of new subscriptions and removals of expired ones could frequently occur. Therefore the application should be prepared to cope with frequent subscription changes and updates. The number of publishers can be large, thus the rate of emitted publication events can also be important. The application should be ready to deal with medium to high rates of publication events. In such a context, the system's performance is an important consideration. In addition, in this type of application the interests of the subscribers can vary in time, as well as the characteristics of the publications emitted by publishers. The application should take into account the distribution of the subscription and publication objects to improve the system's performance, and to adapt it to important changes that might occur over time in the subscription and publication patterns.

Image classification application

The second motivating example consists of an image classification application dealing with large sets of image classes. Comparing and classifying images is a complex task: On the one hand, relevant image similarity criteria have to be defined. On the other hand, the high dimensionality involved in images requires high computation efforts. In practice, different image features, such as distribution of colors, textures, and local shapes, are extracted and represented as multidimensional image descriptors. The image descriptors are further used, instead of images, for comparison and classification purposes [AG01, GS04]. An example of a color-based image descriptor can be a three-dimensional histogram vector recording the amounts of red, green and blue from the corresponding image. In general one expects that two similar images should have spatially-close image descriptors. Choosing good image descriptors is an important consideration, often application dependent. In practice, to better capture the human perception of image similarity, many image features are considered, resulting in high dimensional image descriptors with tens to hundreds of dimensions.

In the image classification domain, a possible image class definition consists of associating an image class with a multidimensional *range* descriptor where each dimension is represented by an interval of values. With respect to such a definition, a given image belongs to a given class if the image descriptor has all its values falling in the intervals of variation of the class descriptor. For instance, we could consider that all the images having between 10% and 15% of red and between 35% and 45% of blue in their color histograms belong to the same image class. An image classification application should be able to manage large sets of image classes in order to quickly classify high rates of incoming images represented by multidimensional image descriptors. For each occurring image, all the qualifying image classes need to be retrieved. As part of the same application, image class descriptors may also be compared to the image class collection in order

to answer queries like "Retrieve all the image classes overlapping, contained in, enclosing, or having a similar shape with a given image class".

Data characteristics and performance requirements: An image classification application using multidimensional range descriptors has to answer multidimensional point and spatial range queries (i.e., intersection, containment, enclosure queries) over a database of multidimensional objects with spatial extents (hyper-intervals or hyper-rectangles) representing image classes. In this case, the application objects involve many attributes corresponding to different image descriptor dimensions. The collection of image classes is rather static, but the rate of images submitted for classification can be important (i.e., classification of images from image streams). The multidimensional aspect and the system's performance are two important considerations in such application cases.

1.2 Challenge for the multidimensional indexing

As shown in the two motivating examples, advanced publish and subscribe applications use range intervals instead of single values for the subscription and publication attributes [LJ04]. Such applications have to manage large collections of range subscriptions with possibly many attributes (dimensions), to cope with high rates of incoming publication events, to support frequent subscription updates (insertions and deletions), and to accommodate data (subscriptions) and query (publication events) distribution changes. An efficient multidimensional indexing solution is needed to meet the performance requirements of such applications.

Problem formulation

The subscription and the publication objects are commonly represented as sets of <*attribute, interval of values*> associations. This format is widely used due to its simplicity and flexibility in representing a large range of different types of information. Considering each attribute as a different dimension, the subscription and the publication objects can be represented in a multidimensional space where each dimension stands for a different space axis. Under such a representation, range subscriptions and range publications can be regarded as multidimensional objects with axes-parallel spatial extents. Such objects are known as hyper-intervals or hyper-rectangles, and also referred to as multidimensional spatial objects or multidimensional extended objects.

Using this multidimensional space representation model, the collection of subscriptions forms a database of multidimensional extended objects. Retrieving the subscriptions matching the incoming publications is equivalent to answering spatial range queries over the database of multidimensional extended objects representing the set of subscriptions. In this context, the main problem we address is how to efficiently index a large and dynamic collection of multidimensional extended objects in order to provide fast answers to the spatial range queries corresponding to the publication events.

Requirements

The indexing solution should meet a number of requirements specific to information dissemination applications:

- *Scalability*: Manage large collections of multidimensional extended objects with possibly many dimensions and with possibly large extents over dimensions.

  Data dissemination applications could involve large collections of subscriptions (up to several millions) and significant numbers of subscription/publication attributes (up to tens of dimensions). The subscriptions could involve attributes with large acceptable ranges of values. Therefore the supporting multidimensional access method should scale with the number of objects, cope with possibly high numbers of dimensions and also support the presence of spatial objects with possibly large extents over dimensions.

- *Search performance*: Cope with high rates of spatial range queries.

  In a number of applications the reactivity of the system is crucial (e.g., alerting, alarm, awareness and notification systems). Some applications need to handle hundreds to thousands of publication events per second (e.g, financial security and exchange systems, stock trading, auctions). To support high rates of publication events, the backing multidimensional indexing method should efficiently (quickly) answer spatial range queries. Spatial range queries such as intersection, containment, or enclosure queries are particularly expensive because they need to explore large portions of the data space.

- *Update performance*: Support frequent data object updates.

  In many data dissemination applications, the collection of subscriptions is highly dynamic, insertions of new subscriptions and removals of expired subscriptions frequently occur. In such contexts, insertions of new objects and removals of existing objects need to be fast, not to significantly affect the system availability.

- *Adaptability*: Take into consideration the spatial distribution of the data objects (subscriptions) and of the query objects (publications) and dynamically accommodate important distribution changes that might occur over time.

  In many publish and subscribe applications, the subscriptions tend to follow some regions of interest, while the publications do not necessary listen to these interests. In such contexts, taking into account the spatial distribution of the data objects and of the query objects could considerably improve the search performance. The interests of subscribers might vary in time, as well as the characteristics of the publications emitted by publishers. Important changes occurring in the data distribution (i.e., subscription insertions or removals) or in the query distribution (i.e., changes in publication patterns) could seriously degrade the search performance. Therefore, dynamically accommodating changes in the data and in the query distribution is an important consideration for our target applications.

At once meeting all these requirements represents a great challenge for the multidimensional data indexing.

Limitations of existing solutions

Although the motivation of our work is introduced in a new setting (in the context of the new emerging applications from the SDI domain), the problem of indexing multidimensional objects with spatial extents is already known in literature. Numerous indexing techniques have been proposed to improve the search performance over large collections of multidimensional objects. For the special case of multidimensional objects with spatial extents, an entire family of solutions originated and evolved from the R-tree technique.

The R-tree technique [Gut84] relies on minimum bounding to delimit space regions that enclose completely and tightly spatial data objects or smaller bounding regions, in a recursive manner. The space regions, commonly hyper-rectangles, are referred to as minimum bounding regions or MBRs. The MBRs are hierarchically arranged in a height-balanced tree. Each node of the tree corresponds to one page of secondary storage. The data objects are stored or referenced in the leaf nodes of the tree. An entry in a leaf node consists of an object identifier, and a MBR minimally enclosing the corresponding data object. An entry in a non-leaf node consists of a pointer to a child node, and a MBR minimally enclosing the MBRs from the lower levels of the corresponding child node subtree. In order to enable a number of nice properties for the indexing structure, such as tree height balance and minimal node/page storage utilization, the MBRs are allowed to overlap, including at node level.

However, the overlap between MBRs is not a desirable feature because, during searches, it determines exploration of multiple tree paths, with negative impact on the search performance. All types of queries are affected, notably the spatial range queries that need to explore larger portions of the data space. The presence of multidimensional spatial objects, with possibly large extents over dimensions, accentuates the global overlap between MBRs. In addition, the overlap probability is increasing with the number of dimensions, phenomenon known as "dimensional curse" [BKK96, BBK98b, BBK01]. As a result, during spatial selections an important number of nodes/pages need to be accessed, notably in a random manner. Random disk access requires disk head repositioning, inflicting high I/O costs. This leads to serious degradation of the search performance, especially in high dimensions. For these reasons, the usability of the R-tree method is limited to applications involving only few dimensions and small volume spatial objects (e.g., close to point objects).

Different tree construction strategies and node split heuristics have been proposed to minimize the overlap between MBRs (e.g., Packed R-tree [RL85], R^+-tree [SRF87], R*-tree [BKSS90], Hilbert R-tree [FB93]). Despite many efforts, in high dimensions the MBRs overlap remains too important, and the degradation of the search performance can not be avoided. In spaces with more than 5-6 dimensions, too many nodes/pages are accessed in a random manner and R-trees fail to beat the sequential scan which, in turn, benefits of sustained data transfer between disk and memory [BBK98b, BK00]. Based on this observation, structural changes have been proposed to R-trees as trade-off solutions between random access and sequential scan: X-tree [BKK96], DABS-tree [BK00]. The basic idea is to avoid splitting the nodes that induce high overlaps, and rather enlarge their capacity, in order to replace several random accesses with sequential scan. Cost models embedding the performance characteristics of the execution platform like I/O and CPU parameters are used to decide whether nodes should be split or extended. In general, the cost models take into account the data distribution and assume the query

distribution follows the data distribution. In practice, this assumption is in general not true, especially in data dissemination applications.

Taking into account the query distribution and regarding the node access probability when deciding node splits or extensions could help to improve the search performance. Following this consideration, [TP02] proposes a general framework for converting traditional indexing structures to adaptive versions, exploiting the query distribution in addition to the data distribution. The framework is employed to construct and maintain Adaptive B-trees. A generalization is also proposed for Adaptive R-trees. According to this method, optimal node size is determined based on statistical information associated with the data space covered by the given node. Statistics concerning both data and query distributions are maintained in a global histogram dividing the data space into bins/cells of equal extent/volume. These statistics are used in a cost model, together with system performance parameters, to estimate the node access cost. The suggested method is impractical for high-dimensional R-trees and is not adapted for multidimensional objects with spatial extents. First, because the number of histogram bins grows exponentially with the number of dimensions and the maintenance cost becomes too important. Second, because the deployed histogram is not suitable for spatial objects that could lay over numerous bins.

Because SDI applications commonly involve frequent data object insertions and deletions, another important aspect that needs to be considered in addition to the query performance is the maintenance cost of the supporting indexing structure. It has been noticed that the order in which the data objects are inserted in R-trees affects the search performance [RL85, BKSS90]. Significant changes in the data distribution could also degrade the search performance. To alleviate these problems, different insertion strategies have been proposed aiming to better adapt the tree structure to subsequent data insertions: forced reinsertion of data objects or object redistribution between sibling nodes. However, this is achieved at price of more complex and more expensive insertion and node split procedures. Regarding the data deletion, when the occupancy level of a given node decreases below a minimal usage threshold, the corresponding node is removed and the remaining objects are reinserted in the tree structure or redistributed between sibling nodes. The reinsertion of the remaining objects could trigger additional tree restructuring operations. As a result, the deletion procedure can be quite expensive. In the context of the target applications from the SDI domain, frequent object insertions and deletions could trigger complex tree restructuring operations, resulting in high maintenance costs.

Considering the evolution and the limitations of the existing multidimensional access methods, we can conclude that a new cost-based query-adaptive indexing method is required to handle spatial objects with possibly many dimensions and with possibly large extents over dimensions and to cope with spatial range queries. This indexing solution should be able to gather suitable data and query statistics and to integrate them in a cost model meant to decide optimal node page sizes with respect to the performance characteristics of the execution platform and so to avoid the degradation of the search performance below a naive solution such as sequential scan. The indexing method should support fast data object insertions and deletions, and also adapt to important changes in data and query patterns.

Considerations

As the main objective of our work was to develop a multidimensional indexing solution suitable for applications from the SDI domain, we first investigated the characteristics of this type of applications. We noticed that in practice some attributes are more selective and more discriminatory than others in certain space regions. In such contexts, taking into account the spatial distribution of the query objects could help to improve the object grouping and thus the search performance. To emphasize this idea, we next consider an example from the small ads application.

Example 1 It is reasonable to assume that the small ads database contains an important number of subscriptions interested in low price apartments. Nevertheless, most of the offers occurring in practice are likely to propose moderate and high price apartments. Therefore the subscriptions interested in low price apartments will not be matched most of the time. Based on this observation, we could extract and group in a separate cluster all the subscriptions interested in cheap apartments. Such an action would help to improve the system's average performance by avoiding to check a large fraction of the subscription database every time that moderate and high price offers are processed.

Following this rationale, an important goal of our work was to develop a multidimensional indexing method able to exploit the spatial distribution of the query objects, notably the difference in access probability among dimensions and space regions, as means to assist the object grouping and thus to improve the average performance of spatial queries.

Another goal of our work was to have an indexing method assisted by a cost model taking into account the actual performance characteristics of the execution platform. Such a cost model is essential to monitor the query search performance and to ensure for spatial range queries better average performance than sequential scan.

With the recent proliferation of large-memory systems, efficient indexing solutions working in main memory are also of great interest to applications from the SDI domain, notably in connection with critical and real-time systems. Therefore, a cost model is also necessary in memory to decide optimal node page sizes with respect to the corresponding performance parameters. For this reason, we wanted a cost model flexible and easy to adapt to different storage scenarios such as disk-based storage and main memory storage.

1.3 Contributions

As an alternative to classical multidimensional indexing methods, we propose a cost-based query-adaptive clustering solution meant to improve the average performance of spatial range queries over large and dynamic collections of multidimensional objects with spatial extents. Our clustering approach consists of dynamically grouping the spatial objects in clusters with respect to both data and query distributions and taking into account the performance characteristics of the execution platform.

Our clustering solution is based on the following principles:

- A grouping criterion suitable for multidimensional objects with spatial extents is used to support the object clustering

A cluster regroups spatial objects with similar intervals (i.e., similar locations and similar extents) in a reduced subset of dimensions, namely the most selective and discriminatory dimensions and domain regions relative to the query objects. To identify the best grouping dimensions and domain regions, the grouping criterion is used to deterministically partition each cluster into a number of candidate subclusters representing future cluster candidates. Data and query statistics are maintained for all the candidate subclusters and employed in a cost model to estimate the search performance of the candidate subclusters. The candidate subclusters promising the best profits with respect to the average search performance are chosen to become new clusters.

- A cost model is used to support clustering decisions such as creation of new profitable clusters and removal of older inefficient clusters

  The cost model embeds a number of system parameters affecting the query execution such as disk access/seek time, disk transfer rate, memory access time, and object check rate, and makes use of data and query statistics associated with clusters and with candidate subclusters to evaluate the search performance of existing clusters and to estimate it for candidate subclusters. The cost model is employed to support decision for the following clustering operations:

  1. *Creation of new profitable clusters*
     New clusters are chosen from among the candidate subclusters and are only created (materialized) if they are expected to improve the average performance of spatial queries.
  2. *Removal of inefficient clusters*
     Clusters having lost their profitability as result of changes in the data or in the query distribution are removed.

  The cost model is flexible and can be adapted according to different storage scenarios such as disk-based storage and main memory storage.

Our clustering solution is meant to provide the following features:

- Ensure for spatial range queries better average performance than sequential scan in virtually every case (i.e., in spaces with many dimensions, for large collections of spatial objects, in the presence of data objects with possibly large extents over dimensions)
- Take into account the spatial query distribution and dynamically adapt the object clustering to important changes that might occur over time in the data or in the query distribution
- Adapt the object clustering to the performance characteristics of the execution platform according to the storage scenario
- Support fast database update operations (i.e., data object insertions and deletions)

1.4 Organization

The rest of this dissertation is organized in five chapters as follows.

Chapter 2 presents the problem that we address and reviews the related work. We show that our problem belongs to the multidimensional data indexing domain. The characteristics and the requirements of our target applications represent a great challenge for this domain. In this context, we review the existing multidimensional indexing methods, presenting the evolution, the features and the limitations of the indexing techniques applicable in our case. We finally present the driving principles and outline the main characteristics of the clustering method that we propose as an alternative solution.

Chapter 3 presents the main elements of our clustering solution, namely the clustering strategy, the object grouping criterion, and the cost model supporting the clustering decisions. We first explain the clustering strategy and describe the clustering process. Then we present the object grouping criterion enabling the implementation of our clustering strategy. Finally, we detail the cost model used to assist clustering decisions like creation of new profitable clusters and removal of inefficient clusters.

Chapter 4 provides algorithms and execution procedures for object clustering operations and for standard database manipulation operations. Implementation, cost and complexity aspects are addressed. We first consider the cluster restructuring operations used to accomplish the database clustering: cluster restructuring invocation, cluster split, and cluster merge. Then we provide and discuss algorithms for standard database manipulation operations: spatial query execution, data object insertion and data object deletion.

Chapter 5 considers implementation related aspects and proceeds to an advanced performance evaluation study. We first provide details on the database storage management and on the memory management. Then we present a series of experiments meant to evaluate the performance and the efficiency of our clustering method, and to compare it to alternative solutions. We experimentally show that our method is efficient and practical for different datasets, workloads and storage scenarios.

Chapter 6 concludes this thesis with a summary of our contributions and some remarks about future work.

Chapter 2

Problem specification and related work

In the context of the new emerging publish and subscribe applications, retrieving the subscriptions matching the publication events is equivalent to answering spatial range queries over a dynamic collection of multidimensional objects with spatial extents. The problem of indexing multidimensional objects with spatial extents is already known in literature. Numerous indexing techniques have been proposed to improve the search performance over large collections of multidimensional objects. In this chapter we first present the problem that we address. Then we review the existing multidimensional indexing solutions, with emphasis on the features and the limitations of the indexing solutions suitable for this problem. We finally present the driving principles and the main characteristics of the clustering solution that we propose as an alternative to the existing multidimensional indexing methods.

Chapter organization Section 2.1 presents the problem that we address. Section 2.2 reviews the state-of-the-art. Section 2.3 emphasizes the principles and the characteristics of our clustering solution.

2.1 Problem specification

We first introduce the generic data model that we use to represent the data objects manipulated in advanced information dissemination applications (Section 2.1.1). Then we define the problem that we address and present the main characteristics (Section 2.1.2) and requirements of this problem (Section 2.1.3).

2.1.1 Data representation model

As shown in the introduction, advanced publish and subscribe applications use range intervals instead of single values for the subscription and publication attributes. The subscription and the publication objects are commonly represented as sets of $<$*attribute, interval of values*$>$ associations. This format is widely used due to its simplicity and flexibility in representing a large range of different types of information.

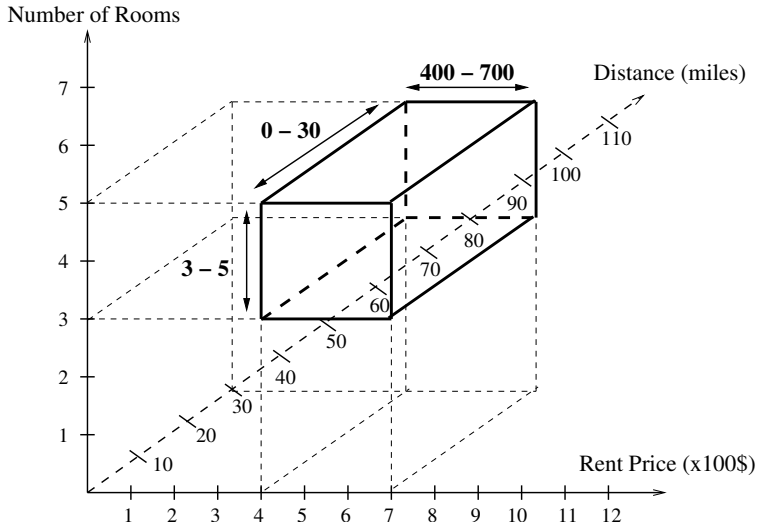


Figure 2.1: Representation of a range subscription as a multidimensional spatial object

Considering each attribute as a different dimension, the subscription and the publication objects can be represented in a multidimensional space where each dimension stands for a different space axis. Under such a representation, range subscriptions and range publications can be regarded as multidimensional objects with axes-parallel spatial extents. Multidimensional objects with axes-parallel spatial extents are also known as hyper-intervals or hyper-rectangles. We also refer to them as multidimensional spatial objects or multidimensional extended objects.

Example 2 The usage of a multidimensional space data representation model in a publish and subscribe application is illustrated in Figure 2.1. The figure depicts a three-dimensional spatial object corresponding to the following small ads subscription: "Looking for an apartment within 30 miles from Newark, with a rent price between 400$ and 700$, and having between 3 and 5 rooms". The three axes from the figure correspond to the three subscription attributes: the distance from Newark (where the given subscription ranges between 0 and 30), the rent price (where the subscription ranges between 400 and 700), and the number of rooms (where the subscription ranges between 3 and 5).

Single values for subscription and publication attributes can be also represented in the multidimensional space. They can be simply modeled as zero-length intervals. In particular, multidimensional points, with single values in all dimensions, can be regarded as multidimensional spatial objects with zero-length extents in all dimensions. Such objects can be used to represent subscriptions or publications with exact values for all the attributes. However, the main target of our applications are spatial objects with real extents over dimensions such as range subscription and range publication objects.

2.1.2 Problem definition

Using the multidimensional space data representation model, the collection of subscriptions forms a database of multidimensional extended objects. The publication objects are also multidimensional objects with spatial extents. Thus retrieving the subscriptions

matching the incoming publication events is equivalent to answering spatial range queries over the database of multidimensional extended objects representing the set of subscriptions. Several types of spatial range queries are of special interest for data dissemination applications:

- *Intersection Queries*

  An intersection query aims to find all the data objects whose intervals intersect the intervals of the query object in all the dimensions.

- *Containment Queries*

  A containment query aims to find all the data objects whose intervals are enclosed by the intervals of the query object in all the dimensions.

- *Enclosure Queries*

  An enclosure query aims to find all the data objects whose intervals enclose the intervals of the query object in all the dimensions.

- *Similar-shape Queries*

  A similar-shape query aims to find all the data objects whose intervals are similar to the intervals of the query object in all the dimensions. By similar intervals we understand intervals with close spatial locations and extensions. Maximum acceptable variation values with respect to the interval bounds of the query object need to be provided as part of the selection criterion.

In this context, the main problem that we address is how to efficiently index a large and dynamic collection of multidimensional extended objects in order to provide fast answers to spatial range queries. This problem belongs to the multidimensional data indexing domain. The needs to manage multidimensional objects with spatial extents and to cope with spatial range queries are the main characteristics of our multidimensional indexing problem.

2.1.3 Requirements

As shown in the introduction, the multidimensional indexing solution to our problem should meet a number of general application requirements regarding the following aspects:

- Scalability

  – Scale with the number of spatial objects, and also cope with many dimensions and with spatial objects with possibly large extents over dimensions.

- Search performance

  – Support high rates of spatial range queries such as intersection, containment, enclosure and similar-shape queries.

- Update performance

  – Support frequent data object insertions and deletions without significantly affecting the system availability.

- Adaptability
  - Take into consideration the spatial distribution of both data objects and query objects and also dynamically accommodate important distribution changes that might occur over time in data and query patterns.

At once meeting all these requirements represents a great challenge for the multidimensional data indexing.

Because the problem that we address belongs to the multidimensional data indexing domain, we need to refer to the related work from this domain. The state-of-the-art is presented in the following section.

2.2 State-of-the-art

The problem of indexing multidimensional objects with spatial extents is already known and has been widely addressed in literature. During the last three decades, numerous indexing techniques have been proposed to improve the search performance over large collections of multidimensional objects. Several surveys [GG98, BBK01, RSV01, Yu02, MNPT03] review and compare most of the existing multidimensional access methods with respect to different environments or applications.

In this section, we review the existing multidimensional indexing solutions, with emphasis on those relevant to our work, notably supporting multidimensional objects with spatial extents. A classification of the existing multidimensional access methods is provided in Section 2.2.1. Following this classification, we first review the multidimensional indexing solutions from the space partitioning family (Section 2.2.2). Then we focus on the indexing approaches based on the R-tree technique, also referred to as region bounding methods. The R-tree technique and the evolution of the region bounding family are presented in Section 2.2.3. Following this evolution, we further detail the R-tree based solutions relevant to our work: dynamic R-tree versions (Section 2.2.4), static R-tree versions (Section 2.2.5), and cost-based R-tree extensions (Section 2.2.6). The limitations of the region bounding indexing methods are revised and analyzed in Section 2.2.7. Conclusions regarding the general requirements of an efficient indexing solution for our problem are presented in Section 2.2.8.

2.2.1 Classification of multidimensional indexing methods

A first classification of the multidimensional access methods was made in [GG98] based on the data types that they have been designed to support:

- *Point Access Methods* (PAMs)
- *Spatial Access Methods* (SAMs)

While the *point access methods* are designed to index collections on multidimensional points, the *spatial access methods* are designed to work with multidimensional objects with spatial extents. Since multidimensional points can be represented as spatial objects

with zero-length extents, all SAMs can function as PAMs. However, the data objects manipulated in our target applications are multidimensional objects with spatial extents. Therefore only the spatial access methods are of special interest for us.

Another classification of the multidimensional indexing techniques can be made based on the way that they partition and organize the collection of data objects. According to this criterion, two major families of solutions can be distinguished:

1. *Space Partitioning Methods*

   The indexing solutions from the *space partitioning family* are derived from K-D-tree [Ben75, BF79] and quad-tree [FB74, Sam84] and rely on recursive partitioning of the multidimensional data space in mutually disjoint regions, so to obtain a limited number of data objects in each space subregion. Most indexing solutions based on space partitioning are PAMs. They primarily apply to collections of multidimensional points, which are easily separable in non-overlapping regions. Some approaches and extensions have been also proposed to support spatial objects.

2. *Region Bounding Methods*

   The indexing methods from the *region bounding family* are based on the R-tree technique [Gut84] which relies on minimum bounding to delimit space regions that enclose completely and tightly spatial data objects or smaller bounding regions, in a recursive manner. The indexing space regions are referred to as *minimum bounding regions* or MBRs. They are hierarchically organized in a height-balanced tree. The indexing solutions based on region bounding are primarily designed to support multidimensional objects with spatial extents (SAMs). For this reason, they are of special interest for us.

We next present the main characteristics and limitations of both families of solutions, and also discuss several extensions and hybrid approaches, focusing on the indexing methods relevant to our work.

2.2.2 Space partitioning methods

The multidimensional access methods from the space partitioning family originated and evolved from K-D-tree [Ben75, BF79] and quad-tree [FB74, Sam84]. They are based on recursive partitioning of the multidimensional data space in disjoint (non-overlapping) regions. The space regions are separated by one (e.g., K-D-tree) or several (e.g., quad-tree) iso-oriented hyperplanes. Each separating hyperplane is perpendicular to one dimension referred to as "split dimension". The split positions in the domains of the split dimensions corresponding to the separating hyperplanes are chosen so that the data objects are distributed evenly between the resulting space subregions. The purpose of the space partitioning is to obtain a limited number of data objects in each space region. When the number of data objects from a space region exceeds the maximum limit, one or several split dimensions are chosen and split positions are established in order to obtain a balanced distribution of the data objects between the resulting subregions.

A tree structure, further referred to as "division tree", is used to record the space partitioning information. The split dimensions and the split positions are stored in the

internal nodes (directory nodes) of the division tree. The data objects are referenced or directly stored in the leaf nodes of the tree. During insertions of new data objects, as well as when performing spatial searches, the split dimensions and the split positions are used to indicate the tree paths that need to be followed. At each level of the division tree, the space subregions that need to be visited are determined by comparing the split positions corresponding to the split dimensions with the coordinates of the query objects.

An important property of space partitioning methods is that point queries follow only one path, corresponding to only one space subregion at each level of the division tree. A common drawback is that the division tree can be quite height-unbalanced, notably in the presence of non-uniform data. This can affect the performance on disk, where special paging techniques need to be used to alleviate this problem.

Space partitioning methods with disk paging capabilities. Several indexing solutions with disk paging capabilities have been proposed: K-D-B-tree [Rob81], Grid file [NHS84], LSD-tree [HSW89], hB-tree [LS90].

The K-D-B-tree [Rob81] combines the properties of the Adaptive K-D-tree [BF79] and the B-tree [Com79] to handle multidimensional points. This results in a balanced tree where each internal node corresponds to a hyper-rectangular region of the data space. This region is partitioned into several mutually disjoint subregions in the manner of an Adaptive K-D-tree. The partitioning information consisting of the split dimensions and split positions is recorded at the node level. This structure adapts well to the distribution of the data points, but no minimum space utilization can be guaranteed because splits of internal nodes have to be propagated down the tree causing fragmentation of lower level nodes.

LSD-tree [HSW89] is another point access method based on K-D-tree. The directory of the LSD-tree is organized as an Adaptive K-D-tree[BF79], partitioning the data space into disjoint cells of various sizes. LSD-tree uses a special paging algorithm which consists of identifying subtrees that can be paged out such as to preserve external balancing.

hB-tree [LS90] is also a K-D-tree based indexing structure with disk paging capabilities. This method proposes node splitting using multiple attributes (dimensions). This way, it guarantees a worst-case data distribution of $1/3 : 2/3$ between the two nodes resulting from a split. Nodes no longer correspond to hyper-rectangular space regions, but to hyper-rectangular space regions from which smaller subregions are excised. Such regions with cavities are called "holey bricks". By using "holey bricks", node splits are local and do not have to be propagated downwards.

The Grid file [NHS84] can be seen as generalization of the quad-tree [FB74]. It divides the data space using a d-dimensional orthogonal grid. The grid is not necessarily regular, thus the resulting cells may be of different sizes. A grid directory associates one or more of these cells with data buckets, which are stored on one disk page each. The purpose of this structure is to have a limited number of data points in each cell. The data points are placed and can be retrieved based on the partitioning information from the grid directory. A known problem of the grid file structure is that cell splitting is not a local operation. All the cells need to be also split, which are intersected by the hyper-plane used to divide one cell. This causes a superlinear directory growth.

Hybrid indexing methods based on space partitioning. Hybrid multidimensional indexing solutions have been also developed combining space partitioning with space transformation: Pyramid-tree [BBK98b], VA-file [WSB98] and iDistance method [JOT$^+$05].

The Pyramid technique [BBK98b] transforms d-dimensional points into 1-dimensional values. These last are stored and accessed using a conventional index such as the B$^+$-tree [Com79]. To achieve this, the Pyramid technique partitions the multidimensional space into a number of $2d$ disjoint pyramids sharing the center point of the data space. Each pyramid is divided into several disjoint partitions such as that each partition fits into one page of the B$^+$-tree.

The VA-file [WB97, WSB98] uses approximated data representation and takes advantage of sequential scan to perform faster searches over large collections of multidimensional points. The VA-file method maps the coordinates of the data objects to some approximated values that reduce the storage requirement, in order to lower the I/O cost of range searching. For this purpose, the VA-file technique divides the data space into 2^b hyper-rectangular cells where $b = \sum_{i=0}^{d-1} b_i$ represents the number of bits used in the data representation. A data point is approximated by the bit string of the cell that it falls into. The object approximations are stored in a flat file on disk. The file is sequentially scanned during spatial searches and the object approximations are used to filter and discard an important number of data objects, with respect to the selection criterion.

In the iDistance method [JOT$^+$05], high-dimensional points are transformed into points in a single dimensional space, further indexed using a conventional B$^+$-tree. To achieve this, the data space is divided into a set of partitions, where each partition is determined by a reference point. A data point is represented by an index key computed based on the distance from the nearest reference point, and used to determine the placement of the corresponding data object in the B$^+$-tree.

Most of the multidimensional indexing methods based on space partitioning are PAMs. They apply to collections of multidimensional points, which are easily separable in non-overlapping regions. Such a separation is not possible for data objects with spatial extents because such objects commonly overlap each others and balanced distribution in mutually disjoint regions is often impossible. For this reason, the classical indexing methods based on space partitioning can not be used directly to manage collections of spatial objects. However, some compromise solutions have been developed in order to enable space partitioning methods to support spatial objects. These solutions are reviewed next.

Space partitioning methods supporting multidimensional spatial objects. To support objects with spatial extents, several approaches have been proposed. The first approach consists of clipping the spatial data objects that intersect the split hyperplanes and insert the resulting object fragments in the corresponding subregions. The Extended K-D-tree [MHN84] is one of the first indexing structures adopting the object fragmentation approach. Another example is the cell tree [Gün89] designed to manage spatial objects of arbitrary shapes. All the structures based on clipping have to cope with the object fragmentation which is becoming increasingly problematic as more objects are inserted into the tree. After some time, most new objects will be split into fragments during insertion. Such a solution is not suitable for indexing large collections of spatial

objects, as the overhead of redundant storage can be very high, notably when having data objects with large extents over dimensions.

The second approach relaxes the non-overlapping property and allows regions to overlap. The SKD-tree (Spatial K-D-tree) [OSDM87] is one of the indexing solutions that adopts this approach: It basically indexes the centroid points of the spatial objects, in a classical K-D-tree based manner, but keeps additional information concerning the maximal extents of the data objects in the two subregions resulting from a split along the split dimension. This approach looses an important property of the classical space partitioning methods: Multiple paths of the indexing tree can now be explored even when performing point queries. According to the authors, the SKD-tree is competitive to the R-tree (a region bounding indexing method) in both storage utilization and search efficiency. However, this technique can only work for spatial objects of small volume. Data objects with large spatial extents induce high overlaps between subregions along the split dimensions. The spatial selections are highly probable to explore both subregions, leading to poor query performance.

A third approach is based on space transformation methods. A first possible space transformation method consists of representing a d-dimensional spatial object with n vertexes as an nd-dimensional point, then using a point access method to index such points. A structure that was designed explicitly to be used in connection with this transformation technique is LSD-tree [HSW89], which appear to adapt well to non-uniform distributions. However, there are major disadvantages of this scheme [GG98]. First, the number of dimensions is much higher than in the original space. Second, objects close in the original space can be far apart in the new space. Third, point and range query formulation is much more complicated than in the original space. Forth, the point distribution in the new space can be highly non-uniform, even though the original data is uniformly distributed.

A second possible space transformation method consists of partitioning the data space with a grid, then using a space-filling curve to enumerate all the cells of the grid [Sam89]. The space-filling curves [Sag94] provide a total ordering of the space, allowing one to use one-dimensional access methods, like B^+-tree, to index the data. The spatial objects can be represented by a list of grid cells, or, equivalently, a list of one-dimensional intervals that define the positions of the grid cells concerned. Such an object decomposition can become complicated, notably in high dimensions. An indexing method that combines the two transformation techniques is proposed in [FR91]. The transformation techniques are only practical for 2- or 3-dimensional spaces.

Conclusion on space partitioning indexing methods. The multidimensional indexing solutions based on space partitioning are primarily designed for collections of multidimensional points. They are not appropriate for data objects with spatial extents. Some special extensions and space transformation methods have been proposed to support spatial objects. However, these extensions complicate the query processing and are impractical notably when dealing with large spatial objects with many dimensions.

For the special case of multidimensional objects with spatial extents, another family of solutions has been developed and used in practice, based on minimum bounding regions.

2.2.3 Region bounding methods

For multidimensional objects with spatial extents, an entire family of solutions originated and evolved from the R-tree technique. The R-tree was introduced in [Gut84] as an access method for extended objects, initially applied to 2d-rectangles. Its generalization to higher dimensions is straightforward. The R-tree relies on minimum bounding regions, commonly hyper-rectangles, to hierarchically organize the spatial objects in a height-balanced tree. To illustrate the principles of this indexing method we next present the original R-tree.

Original R-tree The R-tree [Gut84] was thought of as a multidimensional generalization of B^+-tree [BM72, Com79], designed to store multidimensional rectangles without clipping or transforming them into higher dimensional points. The spatial objects are stored or referenced in the leaf nodes of a height-balanced tree. The tree hierarchy is based on minimum bounding rectangles that enclose spatial objects or smaller bounding rectangles in a recursive manner. A minimum bounding rectangle (MBR) represents the smallest (hyper-)rectangle that encloses completely and tightly one or several multidimensional spatial objects.

The tree structure has two types of nodes: leaf nodes and non-leaf nodes. The non-leaf nodes are also called directory nodes or internal nodes. An entry in a leaf node consists of an object identifier and a minimum bounding rectangle enclosing or representing the data object referenced by the corresponding object identifier. An entry in a directory node consists of a pointer to a child node and a minimum bounding rectangle enclosing the minimum bounding rectangles from the child node referenced by the corresponding pointer.

Every node of the tree corresponds to one page of external support, with the size of one or several I/O blocks. Let M be the maximum number of entries that will fit in one node and let $m \leq \frac{M}{2}$ be a parameter specifying the minimum number of entries in a node. R-tree satisfies the following properties:

1. The root has at least two children unless it is a leaf
2. Every non-leaf node has between m and M entries unless it is the root
3. Every leaf node has between m and M entries unless it is the root
4. All leaves appear on the same level

These construction rules are meant to ensure height balance for the tree hierarchy and minimal node page storage utilization.

An important characteristic of the R-tree method is that the minimum bounding rectangles are allowed to overlap, including those belonging to the same node. The overlap between minimum bounding rectangles is necessary to make possible the balanced partitioning of a set of MBRs into two subsets, notably during node splits. This is essential to enable the nice properties of the R-tree structure such as tree height balance and minimal node storage utilization.

Nevertheless, the overlap between minimum bounding rectangles is not a desirable feature because it induces the exploration of multiple tree paths during spatial selections,

with negative impact on the search performance. Therefore the overlap between minimum bounding rectangles should be minimized. This is also the objective of the data insertion and node split procedures presented next.

Data object insertion. An important aspect of the R-tree method is that redundant storage is not allowed for data objects. A data object falling in a region covered by multiple minimum bounding rectangles is stored or referenced only once in the indexing tree. Insertions of new data objects start by the root and are directed to leaf nodes. To insert a new data object, at each level of the tree, the node that will be least enlarged to accommodate the new object is chosen. In case of a tie, the node with the smallest area is selected. When the last level of tree is reached, the data object is inserted in the leaf node that best fits the new inserted object. If all the entries in the selected node are occupied (node overflow), a node split takes place. For each node that is traversed during the object insertion, the minimum bounding rectangle in the corresponding parent node needs to be readjusted (enlarged) in order to fit the new inserted object.

Node split. The driving criterion for the node split is the minimization of the sum of the areas of the two resulting nodes. When performing a node split, the MBR entry from the parent node corresponding to the split node is replaced with the minimum bounding rectangle that encloses all the entries from one of the two resulting nodes. A new entry with a minimum bounding rectangle enclosing all the entries from the second resulting node is also inserted into the parent node. If the parent node overflows, then the parent node is split. This process may propagate up to the root node. When the root node gets split, a new root is created and the height of the tree increases by one level.

Example 3 Figures 2.2 and 2.3 illustrate two R-tree examples. The R-tree depicted in Figure 2.2 stores nine 2-dimensional spatial objects represented by the minimum bounding rectangles $R1, R2, ..., R9$. The data objects are grouped in three leaf nodes, direct children of the root node. Ra, Rb and Rc represent the minimum bounding rectangles of the three leaf nodes. The maximum number of entries per node is 3.

Figure 2.3 depicts the same R-tree after the insertion of an additional data object represented by $R10$. According to the R-tree from the first figure, the best place to store $R10$ is the leaf node pointed by Rc, which needs the least area enlargement to accommodate the new data object. However, this node is already full, and therefore it has to be split. Its set of member rectangles $\{R7, R8, R9, R10\}$ is partitioned into two subsets chosen to minimize the sum of the areas of the resulting minimum bounding rectangles. The minimum bounding rectangles of the new resulting leaf nodes, Rd and Re, take the place of Rc in the parent node. The parent node is, in this example, the root node of the tree. As the root node is full, it has to be also split. A new root is created storing the minimum bounding rectangles RI and RII resulting from the partitioning of the set of minimum bounding rectangles $\{Ra, Rb, Rd, Re\}$.

Node split algorithms. An essential element of the node split procedure is the algorithm used to partition a set of MBR entries into two subsets aiming to minimize the overlap between the MBRs of the two resulting subsets. [Gut84] proposes three alternative algorithms to handle node splits, which are of linear, quadratic and exponential

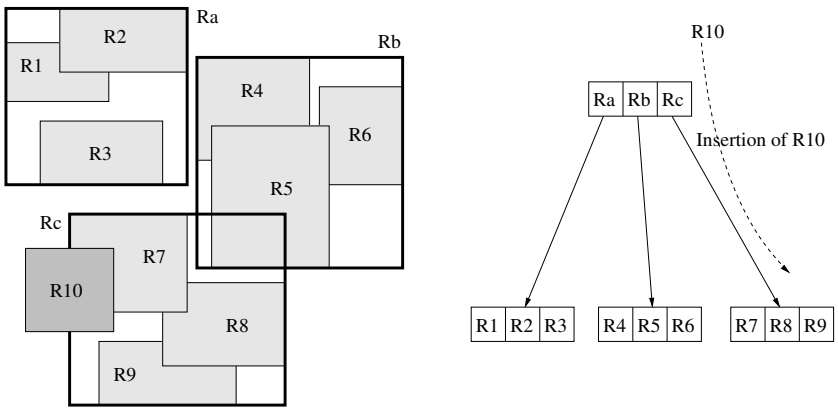


Figure 2.2: An R-tree example

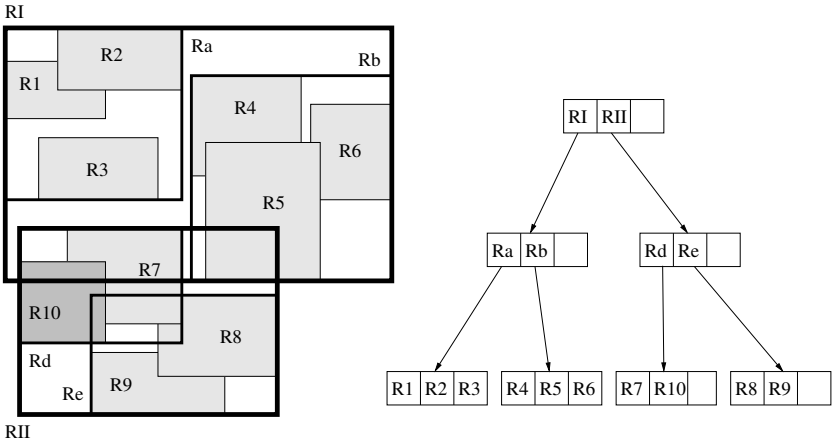


Figure 2.3: The resulting R-tree after insertion of $R10$

complexity. The linear algorithm chooses as seeds, for the two new nodes, the two rectangles that in one of all possible dimensions present the largest separation between the highest low side of the first selected rectangle and the lowest high side of the second selected rectangle. Then each remaining rectangle, in a random order, is assigned to one of the two resulting nodes, namely to the one requiring the least MBR enlargement. The quadratic algorithm considers all the possible pairs of rectangles and chooses as seeds the pair of rectangles that create the largest dead space when put together. The remaining rectangles are assigned to one of the two nodes, in the order that maximizes the difference of dead space obtained if the selected rectangle is assigned to each of the two MBRs. The exponential algorithm considers all possible groupings of rectangles and the best is chosen with respect to the minimization of the MBR enlargement. [Gut84] suggested using the quadratic algorithm as a good compromise to achieve reasonable retrieval performance.

Spatial range queries. In order to find all the data objects which intersect a given query window (query rectangle), the search descends the tree, starting by the root, and recursively traverses the subtrees whose minimum bounding rectangle intersects the query rectangle. When a leaf node is reached, the minimum bounding rectangles corresponding

to the data objects are tested against the query window and the qualifying objects are reported. Since the minimum bounding regions might overlap at node levels, multiple paths could be followed, even for point queries.

Data object deletion. To remove a data object, the tree is traversed and the nodes whose minimum bounding rectangle encloses the given object are explored in a recursive manner. Once located in one of the leaf nodes, the data object is deleted. The object deletion may cause the leaf node to underflow. In this case, the node is removed and the remaining entries are reinserted from the root. The deletion of a leaf node may cause further deletion of nodes in upper levels of the tree. The entries belonging to a node removed from an upper level must be reinserted at the same level of the tree in order to preserve the tree's height balance. The reinsertion of the remaining entries may cause additional node splits. The deletion of a data object may change the minimum bounding rectangles of the nodes situated on the path from the root to the corresponding leaf node. Hence, readjustments of the corresponding entries are necessary.

Performance considerations Although essential to ensure the nice properties of the R-tree structure like tree height balance and minimal node storage utilization, the overlap between minimum bounding rectangles is not a desirable feature because it induces exploration of multiple tree paths during spatial selections, with negative impact on the search performance. When the overlap between minimum bounding rectangles is important, a significant number of nodes/pages need to be accessed, notably in a random manner. Random disk accesses require disk head repositioning, inflicting high I/O costs. This can cause serious degradation of the search performance. All types of queries are affected, notably the spatial range queries which are particularly expensive because they need to explore larger portions of the data space.

In this context, the presence of data objects with (possibly large) spatial extents also contributes to the deterioration of the search performance because it accentuates the global overlap between minimum bounding rectangles.

Another known problem is that the overlapping probability is increasing with the number of dimensions. This phenomenon is referred to as "dimensional curse" [BKK96, BBK98b]. Some insights of this phenomenon are provided in [BBK01], covering the following effects specific to high-dimensional spaces:

- Geometric effects concerning the volume and surface of (hyper-)cubes:
  - the volume of a cube grows exponentially with increasing dimension for constant edge length
  - most of the volume of a cube is very close to the $(d-1)$-dimensional surface of the cube
- Effects on the shape and location of the index space regions:
  - a typical index space region in high-dimensional spaces will span the majority of the data space in most dimensions and only be split in a few dimensions
  - a typical index space region will not be cubic, rather it will look like a rectangle

  - a typical index space region touches the boundary of the data space in most dimensions
  - the space partitioning gets coarser the higher the dimension

- Effects arising in a database environment:
  - assuming uniformity, a reasonably selective range query corresponds to a hyper-cube having a huge extension in each dimension

Because of the effects of the "dimensional curse", the higher the number of dimensions, the higher the overlap between minimum bounding rectangles at node level [BKK96]. This limits the R-tree applicability to only low-dimensional spaces.

A high overlap between minimum bounding rectangles can deteriorate the search performance so much that a naive approach such as sequential scan can become more efficient. To avoid such a behavior and to extend the applicability of the R-tree method, different tree construction strategies, data insertion criteria and node split heuristics have been proposed, aiming to minimize the overlap between minimum bounding regions and trying to alleviate the effects of the "dimensional curse". The evolution of the region bounding methods is outlined next.

Evolution of region bounding methods Since 1984, when the original R-tree structure has been proposed in [Gut84], an impressive number of indexing methods have been developed aiming to improve this technique. A recent survey on R-trees is provided in [MNPT03].

Following the evolution of the R-tree based methods, we have identified the next classes of solutions:

1. Dynamic R-tree versions

   Like the original R-tree, these solutions are designed to support dynamic insertions and deletions of data objects. They preserve the original tree construction approach based on subsequent data insertions, but bring forward better data insertion criteria and node split heuristics meant to minimize the overlap between minimum bounding regions as means to improve the retrieval performance.

2. Static R-tree versions

   These solutions are designed for static collections of data objects. They rely on a priori knowledge of the complete data set to produce better structured trees. The overlap between minimum bounding regions is globally minimized and the general retrieval performance improves.

3. Point-oriented R-tree versions

   Although designed as an indexing method for multidimensional objects with spatial extents, R-tree technique showed good behavior for multidimensional points too, notably in low dimensions. In fact, low-volume objects are preferable because the overlap between data objects accentuates the overlap between minimum bounding

regions, with negative impact on the search performance. Since points can be handled as zero-volume objects, they are well suited for R-trees. Motivated by a wide range of applications (e.g., image retrieval, time series, document indexing), a new class of R-tree-based methods evolved, aiming at working with collections of multidimensional points and focusing on improving performance of nearest neighbor queries: SS-tree [KS97], SR-tree [WJ96], A-tree [SYUK00]. SS-tree [KS97] considers that spheres are more suitable than rectangles for distance-based queries and uses minimum bounding hyper-spheres as support for the tree hierarchy instead of minimum bounding hyper-rectangles. SR-tree [WJ96] uses both minimum bounding hyper-spheres and minimum bounding hyper-rectangles, trying to take advantage of both representations. To assist insertion of new data points, both SS-tree and SR-tree rely on minimizing the distance to the centroids of the candidate subtrees. A-tree [SYUK00] also uses the distance to the centroids of the candidate subtrees for data insertion. Such structural and technical changes helped the new R-tree variants to outperform classical R-trees at nearest neighbor queries on large collections of multidimensional points. However, these extensions do not support multidimensional objects with spatial extents and do not apply in our case. We will not further discuss these extensions.

4. Cost-based R-tree extensions

   These extensions make use of cost models integrating the performance characteristics of the execution platform such as I/O and CPU parameters in order to alleviate the deterioration of the search performance. In general, the cost models take into account the spatial distribution of the data objects and assume that the query distribution follows the data distribution. A new method has been recently proposed, that, in addition to the data distribution, also exploits the query distribution to further improve the average query performance.

We next present and discuss the following classes of region-based indexing solutions relevant to our work: dynamic R-tree versions (Section 2.2.4), static R-tree versions (Section 2.2.5), and cost-based R-tree extensions (Section 2.2.6).

2.2.4 Dynamic R-tree versions

The R-tree was originally designed as a completely dynamic structure: Insertions and deletions of data objects can be intermixed with queries and no global reorganization is required. The indexing tree is meant to grow in a gradual manner, by means of subsequent data object insertions. Many R-tree based methods preserved this construction approach, while focusing on reducing the overlap between minimum bounding regions in order to obtain better structured trees. The following methods are representative for this class: R^+-tree [SRF87], R*-tree [BKSS90] and Hilbert R-tree [KF94]. We next overview these methods.

R^+-tree R^+-tree [SRF87] uses the clipping technique to completely avoid overlap between minimum bounding rectangles situated at the same tree level. This approach aims to improve the retrieval performance by ensuring single search paths for point queries. To achieve this, inserted objects that overlap several MBR partitions are divided in two or

more fragments. The resulting fragments are inserted into the corresponding partitions, carrying with them the original object identifier. Thus an object may be fragmented and its reference redundantly stored in several leaf nodes.

When a node overflows, its minimum bounding rectangle is divided into two disjoint MBRs at a suitable position. The entries overlapping both MBRs are clipped and the resulting fragments are inserted into the qualifying partitions. Unlike in the original R-tree, where splitting propagates only up the tree, here it can propagate down the tree, causing further object fragmentation and low storage utilization.

The object deletions also become more expensive because all the fragments of a deleted object need to be located in order to be removed from the corresponding leaf nodes. When a node underflows, the reinsertion of the remaining entries may cause additional fragmentation.

Overlap avoiding is achieved at the expense of space, increasing the height of the tree and lowering the storage utilization. R^+-tree claims better performance in low dimensions, but it is impractical in high dimensions, where the increased overlap causes significant object fragmentation and the overhead of redundant storage can be very high.

R*-tree R*-tree [BKSS90] preserves the original R-tree model, but brings forward improved insertion and node split heuristics, intended to reduce the overlap between minimum bounding rectangles. Several optimization criteria are combined to assist object insertions and node splits as follows:

- Minimize the overlap between MBRs;
- Minimize the surface covered by MBRs, notably the dead space covered by bounding rectangles and not covered by enclosed rectangles – This criterion was also used in the original R-tree;
- Minimize the margins (the perimeters) of MBRs – Squarish MBRs are preferred because they are more compact;
- Maximize the storage utilization – This is achieved by forced data object reinsertions.

Performance studies have demonstrated that R-trees can behave differently for the same data set if the sequence of insertions is different. The search performance is sensitive to the order of the insertion of data. Deletions and reinsertions of data objects can lead to better structured trees with significantly improved search performance [BKSS90].

Aiming to better adapt the tree structure to subsequent data insertions, R*-tree introduces the concept of "forced reinsert": When a node overflows, a defined percentage of entries with the highest distances from the center of the corresponding MBR are extracted from the node and reinserted in the tree. The object reinsertion helps to increase the storage utilization, obtaining better structured trees, but it can be expensive when the trees are large.

Node splits need to be performed when the reinsertion ends up placing the objects in the same nodes. The node split algorithm proposed by R*-tree is based on the optimization criteria itemized before and leads to better partitioning than previous methods. The

node split algorithm has a complexity of the form $C(2 * d * (n+1) * log(n+1))$ where n represents the node capacity and d represents the number of dimensions.

Compared to previous methods, R*-tree demonstrated best search performance and became a comparison reference for later multidimensional indexing solutions. The R*-tree insertion and node split algorithms improve the original R-tree structure as far as retrieval is concerned. Evidently, this is not for free: The complexity and the cost of the insertion procedure increase.

Hilbert R-tree A special variant of R-tree is the Hilbert R-tree [FB93, KF94]. This method makes use of a space-filling curve, namely the Hilbert curve [Sag94], to compute the Hilbert values of the centroid points of the MBRs corresponding to data objects and to child nodes. In each entry of a node, together with the minimum bounding rectangle, the largest Hilbert value of the enclosed MBRs is stored.

The Hilbert values associated with MBRs are used as keys to assist data insertion and node splits. At each level of the tree, a new data object is inserted in the node with the smallest Hilbert value larger than the Hilbert value corresponding to the new data object. In case of node overflow, Hilbert R-tree redistributes some entries to sibling nodes, or splits the overflowing node when such a redistribution is not possible. In case of node underflow, Hilbert R-tree borrows some entries from sibling nodes, or merges the underflowing node with its siblings. As a result, Hilbert R-tree better controls and improves the storage utilization.

According to the authors, Hilbert R-tree proved better performance than R*-tree for 2-dimensional data sets. However, this solution is vulnerable performance-wise to large objects, notably in higher dimensions, where the generalization of the Hilbert curve is also more complex.

Other data insertion and node split algorithms aiming to produce better structured trees with less overlap between MBRs and with better storage utilization were proposed in [AT97], [GLL98b], [SC00] and [HLL01].

Performance considerations Thanks to the new data insertion criteria and node split heuristics the search performance of the new R-tree versions has improved compared with the original R-tree. However, these techniques are only efficient in spaces with a few dimensions and for collections of spatial objects whose size is small relative to the size of the data space (i.e., objects with small extents over dimensions). In spaces with many dimensions, due to the "dimensional curse", the overlap between minimum bounding regions may become large enough to render the indexing tree ineffective. One ends up searching most of the index nodes in a random manner, which turns to be more expensive than a simple sequential scan. Performance studies have demonstrated that advanced R-tree variants such as R*-tree and Hilbert R-tree fail to beat the sequential scan in spaces with more than 5-6 dimensions [BBK98b, BK00, BBK01].

2.2.5 Static R-tree versions

R-tree was originally thought as a completely dynamic indexing structure. Insertions and deletions of data objects can be intermixed with queries and no global reorganization is required. However, performance studies showed that R-trees are sensitive to the order of insertion of data and can behave differently for the same data set if the sequence of insertions is different [BKSS90]. Deletions and reinsertions of data objects can lead to better structured trees with significantly improved retrieval performance.

Based on these facts and motivated by a significant number of applications involving static data (applications where data insertions and data deletions are very rare or inexistent), special attention was paid to produce optimal tree structures for sets of data objects known in advance. Several construction methods exploiting a priori knowledge of the complete data set as means to improve the retrieval performance have been proposed. They are known in literature as "packing" or "bulk-loading" methods.

Unlike in the original approach, where the indexing tree is growing in a top-down manner by means of subsequent object insertions, the new construction methods are building the indexing tree from bottom to up: First the data objects are grouped in leaf nodes. Then the leaf nodes are grouped in internal nodes, and so on up to the root.

The first packing method was proposed soon after the publication of the original R-tree: Packed R-tree [RL85]. This method suggests ordering the spatial objects according to some spatial criterion (e.g., according to ascending x-coordinate), and then grouping them in leaf nodes. [KF93] proposes a construction method to obtain Packed Hilbert R-trees. It consists of sorting the spatial objects according to the Hilbert values of their centroids and then build the tree in a bottom-up manner. Other packing algorithms were later proposed in [vSW97], [LEL97], [BBK98a], [GLL98a], [CCR98], [AHVV02], [dMOG02].

An interesting study regarding the optimal clustering of a static collection of spatial objects is presented in [PSW95]. The static clustering problem is solved as a classical optimization problem, but the data and the query distributions need to be known in advance.

Performance considerations The R-tree structures resulting from packing methods show better retrieval performance compared with their dynamic relatives, notably in low dimensions and for low-volume objects. However, in higher dimensions, they suffer as well from the "dimensional curse". In addition, these construction methods only work for static collections of spatial objects, requiring all the data to be available in advance. A major drawback is that when data objects change, the indexing structure has to be rebuilt from scratch, otherwise the gains in retrieval performance are lost.

2.2.6 Cost-based R-tree extensions

As already shown, different tree construction strategies, data insertion criteria and node split heuristics have been proposed to minimize the overlap between minimum bounding regions. Despite these efforts, in high dimensions the overlap between MBRs remains important, and the degradation of the search performance under sequential scan can not be avoided. The bad performance compared to sequential scan is mostly due to the

fact that too many nodes/pages are accessed in a random manner, while the sequential scan benefits of sustained data transfer between disk and memory. According to [BK00], contiguous reading of a large file can be by a factor greater than 12 faster than reading the same amount of data from random positions.

In this context, an obvious drawback of classical R-trees is that they neglect the performance characteristics of the execution platform when building the indexing structure. In order to mitigate this problem, structural changes have been proposed to R-trees as trade-off solutions between random access and sequential scan: X-tree [BKK96], DABS-tree [BK00], and Adaptive R-tree [TP02]. The basic idea is to avoid splitting the nodes that induce high overlaps, and rather enlarge their capacity, in order to replace several random accesses with sequential scan. Cost models embedding the performance characteristics of the execution platform are used to decide whether nodes should be split or extended.

X-tree [BKK96] and DABS-tree [BK00] rely on such cost models to adapt the indexing structure to the spatial distribution of the data objects, while assuming that the query distribution follows the data distribution. We further refer to these two methods as cost-based data-adaptive extensions.

Adaptive R-tree [TP02] uses a cost model that, in addition to the data distribution, also takes the query distribution into account. We further refer to this method as a cost-based query-adaptive extension.

These cost-based R-tree extensions are relevant to our work and will be detailed next.

Cost-based data-adaptive R-tree extensions

X-tree X-tree [BKK96] was proposed as a cost-based extension of the R-tree structure for data spaces with many dimensions. In such spaces, the overlap between MBRs is important, notably in directory nodes. This causes numerous random accesses during spatial searches, inflicting high I/O costs and resulting in worse performance than sequential scan. As a compromise solution between random access and sequential scan, X-tree introduces the concept of "supernode".

Supernodes are created during object insertions by assigning multiple subsequent disk pages to overflowing directory nodes whose split would otherwise result in sibling nodes with highly overlapping MBRs. This approach is based on the consideration that two sibling nodes with highly overlapping MBRs are very probable to be accessed during the same spatial selections. One random access followed by one sequential read of two disk pages is expected to perform faster than two random accesses followed by two disk page reads. Based on this consideration, the directory node is preserved unsplit, and, instead of splitting, its storage capacity is extended by one additional disk page, obtaining a supernode. The node extension solves the overflowing situation.

Storage management aspects. For this method to work, a special storage management strategy is needed to ensure that pages associated with supernodes are contiguously placed on the external support. When there is not enough contiguous space on disk to sequentially store a supernode, the disk manager has to perform a local reorganization. The storage management is more complex because it has to avoid storage fragmentation.

However, this is not considered an important problem.

Search and delete algorithms. The algorithms to search the X-tree (point, range queries) are similar to the algorithms used in classical R-trees since only minor changes are necessary in accessing supernodes. The delete and update operations are also simple modifications of the corresponding R-tree algorithms. In case of an underflow of a supernode, when the supernode has two pages, it is converted into a normal directory node. Otherwise, the size of the supernode is reduced by one page.

Cost model. The most important consideration of this approach is the cost model used to decide the split or the extension of an overflowing directory node. The criterion supporting this decision consists of using a maximal overlap threshold to which the overlap occurring between the MBRs of the two nodes resulting from a directory node split is compared. When the resulting overlap is below the threshold value, the split is performed, otherwise the directory node is extended becoming a supernode.

The threshold value is fixed and determined based on the performance characteristics of the execution platform. According to [BKK96], the maximum overlap value is given by the following formula:

$$MaxO = \frac{T_{Tr} + T_{CPU}}{T_{IO} + T_{Tr} + T_{CPU}} \tag{2.1}$$

where

- T_{IO} represents the time to perform a disk seek operation (disk access time)
- T_{Tr} represents the time to transfer a page from disk to memory
- T_{CPU} represents the time necessary to process a page (i.e., verify the selection criterion against the MBRs stored in the page)

This formula is determined by the balance between reading a supernode of two pages, on the one side, and reading two normal nodes of one page each with a probability of $MaxO$ and one normal node with a probability of $(1 - MaxO)$, on the other side. This estimation is only correct for the simplest case of initially creating a supernode. However, the authors of X-tree consider it as a good estimation for practical purposes.

There are two extreme cases of the X-tree: (1) none of the directory nodes is a supernode and (2) the directory consists of only one large supernode, the root. According to the authors, the first case may occur for low dimensional and non-overlapping data, while the second case is likely to occur for high-dimensional or highly overlapping data. Between the two extremes, we have trees with hybrid directory hierarchies, consisting of both normal nodes and supernodes, determined by the spatial distribution of the data objects.

Performance considerations. An important observation is that the cost model of X-tree only applies to directory nodes. The leaf nodes storing or referencing the data objects

have fixed sizes and continue to be accessed in a random manner. Because the search performance is determined by both fractions of directory nodes and of leaf nodes accessed during spatial selections, this cost model cannot always guarantee better average search performance than sequential scan. According to [BBK01], for small dimensions, the X-tree shows a behavior almost identical to R-trees. In medium-dimensional spaces, the X-tree shows an important performance gain compared to R*-tree for all types of queries. For higher dimensions, the X-tree has to access such a large number of nodes that a sequential scan is less expensive.

Because most of the experimental evaluations were performed for collections of multidimensional points, we conducted a study to determine whether this cost model can be used in practice to support indexing of collections of multidimensional objects with spatial extents. This study is presented in Section 2.2.7. According to our experiments, the average overlap occurring in practice between MBRs resulting from directory node splits is much greater than the threshold value computed according to the formula (2.1) when considering modern I/O and CPU system parameter values. As a result, the directory of X-tree reduces to only one large supernode. In this case, the search performance is mostly determined by the percentage of leaf nodes accessed and can get worse than a sequential scan.

DABS-tree In DABS-tree [BK00], a cost-based approach is also proposed for multidimensional indexing, which consists of dynamically computing optimal node page sizes adapted according to the spatial distribution of the data objects. This indexing method is designed for collections of multidimensional points, but the cost model used to decide node splits and merges is quite generic and could apply to other indexing structures.

Unlike X-tree where only directory nodes could be extended, the DABS-tree rather adapts the page size of the leaf nodes used to store the data objects. The DABS-tree uses a flat directory whose entries consist of an MBR, a pointer to a data page and the size of the data page. Additionally to the linear directory, a K-D-tree is maintained in order to guarantee overlap-free page regions. For this reason, this indexing structure only works for multidimensional points. The K-D-tree is also used to facilitate insertion of new data points, and to assist merging operations between leaf pages. Only two leaf pages with a common parent node in the K-D-tree are eligible for merging.

Storage management aspects. The data pages are always full. This ensure 100% storage utilization and reduces the amount of data read from external support. Whenever a new entry is inserted into a data page, the page is stored at a new position. A garbage collection strategy is applied to avoid storage fragmentation. Such a storage management politics could be however expensive in highly dynamic environments.

Cost model. After a number of inserts or deletions, a cost estimate for the current data page is determined. The data page is checked whether a split is likely to improve the query performance. When this is the case, the data page is split. Otherwise, if a suitable partner can be found, a merging operation is considered. The merging is only performed if it is expected to improve the query performance.

When taking a split or a merge decision, the cost caused by one larger page is com-

pared with the cost caused by two pages with smaller capacities. The following balance is used to support the split or the merge decision:

$$\Delta_T = (T_{IO} + C_1 * T_{TrP}) * p_1 + (T_{IO} + C_2 * T_{TrP}) * p_2 - (T_{IO} + C_0 * T_{TrP}) * p_0 \quad (2.2)$$

where

- T_{IO} represents the time to perform a disk seek operation (disk access time)
- T_{TrP} represents the time to transfer a point from disk to memory
- C_0 and p_0 represent the capacity and the access probability of the larger page
- C_1 and C_2, and respectively p_1 and p_2, represent the capacities and the access probabilities of the two smaller pages

T_{IO} and T_{TrP} are hardware dependent. The page capacities, C_0, C_1 and C_2, are known. The access probabilities, p_0, p_1 and p_2 are not known, but they can be estimated if a suitable access probability model is available. The authors of DABS-tree suggest using an access probability model suitable for nearest-neighbor queries. The access probability associated with a page is estimated based on the volume of the MBR of the page and on the number of data objects stored in the page. The analytical model uses the fractal dimension of the data set [BF95] and the Minkowski sum [BBKK97], assuming that the query distribution follows the data distribution.

According to (2.2), if the cost balance Δ_T is negative, the larger page causes higher cost than the two smaller pages. When a split decision is considered, the split operation can be performed. If a merge decision is considered, the merge operation can be only performed when Δ_T is positive.

Performance Considerations. Experimental studies involving collections of multidimensional points have demonstrated better performance for DABS-tree than both X-tree and sequential scan, even in cases where the X-tree failed to outperform the sequential scan. Thanks to its cost model, DABS-tree claims to outperform the sequential scan in virtually every case. However, this can be only true when the query distribution follows the data distribution. When this is not the case, the cost model can trigger inappropriate page split and merge decisions, causing deterioration of the search performance.

Although DABS-tree is an indexing structure for multidimensional points, the cost model used to support node split and merge decisions is quite generic and can apply to other indexing structures with condition that the page access probability is known or can be estimated. We used a similar cost model to support our clustering solution.

Cost-based query-adaptive R-tree extension

The cost models of both X-tree and DABS-tree rely on the spatial distribution of the data objects to optimize the node page size, while assuming that the query distribution follows the data distribution. In practice, the distribution of the query objects does

not always follow the distribution of the data objects. In such contexts, taking into account the query distribution and the node access probability when deciding node splits or extensions could help to significantly improve the average search performance.

Adaptive R-tree Maintaining histogram-based data and query distribution statistics for effective query optimization received considerable research attention over years [HS92, CR94, IP95, GM98, APR99, GLR00, BCG01, WAE01, WAE02]. Based on this consideration, [TP02] proposes a general framework for converting traditional multidimensional indexing structures to adaptive versions, exploiting the query distribution in addition to the data distribution. The framework is employed to construct and maintain Adaptive B-trees. A generalization is also proposed to obtain Adaptive R-trees.

Like in X-tree, performance gain is obtained by allowing nodes to extend over a variable number of subsequent disk pages. However, in X-tree only directory nodes could be extended. In Adaptive R-tree the size of the leaf nodes can be also adapted. Size of a node is decided when the node is created, and reconsidered when the node incurs over/under-flows. The number of pages associated with a node is determined based on a cost model that takes into account both data and query distributions, as well as the system's performance parameters.

Cost model. The optimal number of disk pages for a node is determined based on statistical information associated with the data space covered by the given node. Statistics concerning both data and query distributions are maintained in a global histogram dividing the data space into bins/cells of equal extent/volume. These statistics are employed in an analytical model together with system performance parameters to estimate the average query cost associated with each bin. The query cost associated with a bin is further used to derive an optimal size for the nodes falling in that bin. This model assumes that nodes have smaller extents than the bins' extents, and that all the nodes falling in the same bin have similar optimal sizes.

The optimal number of disk pages p_{opt} associated with a node falling in a bin i is computed following the next three steps:

Step 1. The node average query time is expressed as a function of the number of disk pages associated with the node and of other parameters as follows:

$$T_q(p) = f(p, n_i, q_i, sum\_q_i, sum\_q_i^2, num_{bin}, b, \xi, T_{IO}, T_{Tr}, T_{CPU})$$

where

- T_q represents the node average query time
- p represents the number of disk pages associated with the node

The other parameters used in the cost model are:

- data and query statistics associated with the bin i enclosing the node
  - n_i represents the number of data objects falling in the bin i
  - q_i represents the number of queries that intersect the extent of the bin i

- sum_q_i represents the range sum of queries that intersect the extent of the bin i
- $sum\_q_i^2$ represents the area sum of queries that intersect the extent of the bin i when having more than one dimension

- system and database parameters
  - num_{bin} represents the number of bins in the histogram
  - b represents the number of entries contained in a page
  - ξ represents the average node storage utilization
- I/O and CPU performance parameters
  - T_{IO} represents the time to perform a disk seek operation (disk access time)
  - T_{Tr} represents the time to transfer one page from disk to memory
  - T_{CPU} represents the time to process one data object

The exact expression of $T_q(p)$ is provided in [TP02].

Step 2. The derivative of $T_q(p)$ with respect to p is calculated

$$T'_q(p) = d\frac{f(p)}{dp}$$

Step 3. The optimal p that minimizes $T_q(p)$ is obtained by solving the equation

$$T'_q(p) = 0 \Rightarrow p_{opt}$$

The solution of the above derivative is easy to compute in one dimension. However, in multidimensional spaces, it requires numerical approaches that are too expensive to compute in real-time. The authors propose to substitute this computation with an algorithm that starts with an initial size $p = p_{guess}$ and then refines it iteratively by modifying p towards minimizing the query time $T_q(p)$.

Performance considerations. This technique seems to be effective for B-trees (in one dimension), but it is highly impractical for high-dimensional R-trees for several reasons: First, the number of histogram bins/cells grows exponentially with the number of dimensions and the maintenance cost increases too much. Considering, for instance, a 20-dimensional space and a space division factor of 4 bins per dimension, more than one million cells need to be managed.

Second, the deployed histogram might be suitable for multidimensional points which necessarily fit the bins, but it is inappropriate for multidimensional extended objects which could expand over numerous bins. In addition, the cost model assumption according to which node extents are smaller than bin extents limits the practical usage of this technique.

Another observation is that the node sizes are only adjusted when over/under-flows situations occur. This could cause performance degradation if changes in the query distribution are not followed by changes in the data distribution.

Even though impractical in high dimensions and inappropriate for spatial objects, the Adaptive R-tree represents the first multidimensional indexing method proposing to take into account the real distribution of the query objects as means to improve the average query performance. We also adopted a similar strategy because it proves to be of special interest for our target applications from the SDI domain.

2.2.7 Limitations of region bounding methods

As shown in our motivation, we require an indexing method able to efficiently answer spatial range queries over large collections of spatial objects with possibly many dimensions and with possibly large extents over dimensions. Among the existing multidimensional indexing solutions, the most likely to meet our application requirements are those belonging to the region bounding family, specifically designed to support multidimensional object with spatial extents. With respect to the characteristics of our target applications, the main causes limiting the practical usage of the region bounding methods are:

1. The need to manage high-dimensional data spaces with more than 5-6 dimensions

   Due to the effects of the "dimensional curse", the overlap between minimum bounding rectangles increases with the number of dimensions, with negative impact on the search performance.

2. The need to handle multidimensional spatial objects with possibly large extents over dimensions

   The presence of data objects with spatial extents accentuates the global overlap between minimum bounding rectangles and contributes to the deterioration of the search performance.

3. The need to cope with spatial range queries like intersection, containment, enclosure and similar-shape queries

   The spatial range queries are in general expensive because they need to explore large portions of the data space. In such cases, an important overlap between minimum bounding rectangles at node level can quickly lead to poor performance.

When the overlap between minimum bounding rectangles is important, the search performance can get worse than a naive solution such as sequential scan due to the excessive number of nodes accessed in a random manner and inflicting high I/O costs.

As explained in the previous section, the only way to alleviate the deterioration of the search performance is to take into account the performance characteristics of the execution platform and to adapt the size of the nodes to the spatial distribution of the data objects (X-tree [BKK96], DABS-tree [BK00]) and of the query objects (Adaptive R-tree [TP02]), as a compromise between random access and sequential scan.

Among the methods proposing cost-based indexing, the DABS-tree is only designed to manage multidimensional points, while the Adaptive R-tree is impractical for more than 2-3 dimensions and inappropriate for spatial objects with large extents over dimensions. The only method supporting multidimensional objects with spatial extents and claiming to work in higher dimensions remains the X-tree. For this reason, we conducted a study meant to determine to which extent the cost model of the X-tree is practical for indexing collections of multidimensional spatial objects. This study is presented next.

Table 2.1: Average performance parameters of our execution platform

I/O	**Value**
disk page access/seek time (T_{IO})	10 ms
disk transfer rate	90 MBytes/sec
disk transfer time per byte ($T_{Tr/byte}$)	$1.06 \cdot 10^{-2}$ μs/byte
CPU	**Value**
processing rate	2300 MBytes/sec
processing time per byte ($T_{CPU/byte}$)	$4.15 \cdot 10^{-4}$ μs/byte

A study of the X-tree cost model

The cost model of X-tree was presented in 2.2.6. Based on the formula (2.1), the authors of X-tree compute and suggest using an overlap threshold $MaxO = 20\%$. This value was obtained using the following system parameters: disk page size = 4KBytes, $T_{IO} = 20ms$, $T_{Tr} = 4ms$, and $T_{CPU} = 1ms$. Although these parameter values were considered as realistic at the time of publication (1996), the performance parameters of nowadays systems have significantly changed. While the disk page access time has only slightly improved, the transfer time and the processing time have considerably decreased. In Table 2.1, we present as reference the average values corresponding to the performance parameters of our execution platform. Note that T_{IO} is fixed, while T_{Tr} and T_{CPU} depend on the disk page size. With these new values and considering disk pages of 4KBytes we now obtain an overlap threshold $MaxO = 0.45\%$.

The difference between the two $MaxO$ values is considerable, which raises some questions about the usability in practice of the new threshold value. In this context, an important observation is that the overlap threshold value $MaxO$ only depends of the system performance characteristics, while the overlap between MBRs in the indexing tree is rather determined by the spatial distribution of the data objects. For this method to work, the overlap threshold value $MaxO$ has to "agree" with the overlap values occurring in practice between MBRs of nodes resulting from splits of directory nodes. Let $AvgO$ denote the average overlap between MBRs of nodes resulting from directory node splits. The following situations can occur:

1. $MaxO \approx AvgO$

   In this case, some directory nodes will be extended and others split, resulting in an indexing tree whose directory nodes are adapted to both the distribution of the spatial objects and the system performance characteristics.

2. $MaxO >> AvgO$

   In this case, most of the directory nodes will split, resulting in an indexing structure very close to a classical R-tree. This is a favorable situation because the resulting structure is theoretically efficient with respect to the system performance characteristics.

3. $MaxO << AvgO$

   In this case, very few splits will occur in directory nodes, resulting in an indexing structure with one or very few, but very large, supernodes. This only guarantees

Table 2.2: System parameters and $MaxO$ with increasing disk page size

Page size [kBytes]	$\mathbf{T_{IO}}$ [ms]	$\mathbf{T_{Tr}}$ [ms]	$\mathbf{T_{CPU}}$ [ms]	**MaxOValue** [%]
4	10	0.043	0.0017	0.45
16	10	0.174	0.0068	1.77
64	10	0.694	0.0272	6.73
256	10	2.778	0.1087	22.40
1024	10	11.111	0.4348	53.59
4096	10	44.444	1.7391	82.20
16384	10	177.778	6.9575	94.86

that the resulting indexing structure is more efficient than a classical R-tree. The search performance is mostly determined by the fraction of leaf nodes accessed during spatial selections. The leaf nodes have fixed sizes, and continue to be accessed in a random manner. When too many leaf nodes are accessed, the search performance gets worse than a sequential scan.

Based on this observation, two important questions need to be answered:

i. Can the $MaxO$ value be controlled and adapted to produce efficient indexing structures with respect to the actual system performance parameters?

ii. Can this cost model be used in practice to index data sets of multidimensional objects with spatial extents?

To answer these questions we studied the variation of the $MaxO$ value with varying disk page size (i.), and we measured the $AvgO$ value for data sets of spatial objects uniformly distributed in spaces with different dimensionality (ii.).

(i.) *Controlling the $MaxO$ value.* The only parameter that can be adjusted to control the $MaxO$ value is the disk page size. Therefore, we studied the variation of the $MaxO$ value with increasing page size. For this purpose, we considered the performance parameters of our execution platform (see Table 2.1) and we computed the $MaxO$ value for different page sizes from 4kBytes to 16MBytes. The page size increases by a factor of 4 each time. Figure 2.4 illustrates the variation of the $MaxO$ value with increasing page size. Table 2.2 provides complementary numbers showing the variation of the system performance parameters with varying page size.

We first notice that the $MaxO$ value is very low for pages with sizes between 4kBytes to 64kBytes (note the logarithmic scale on the vertical axis from Figure 2.4), it reaches the value of 20% for a page size of around 256kBytes, and grows up to 95% for a page size of 16MBytes. We have, at least in theory, the capacity to control the $MaxO$ value, by setting a suitable disk page size. However, this approach requires the $AvgO$ value to be known in advance. To obtain, for instance, a $MaxO$ value of about 20% as suggested by the authors of X-tree, we need to use a page size of 256kBytes. In practice, the $AvgO$ value can be only experimentally determined for static data sets available in advance.

We also notice that, according to the new performance parameters, we need to use very large pages (256kBytes), for relatively small $MaxO$ values (20%). A problem with using very large disk pages is that the insertion of new data objects becomes

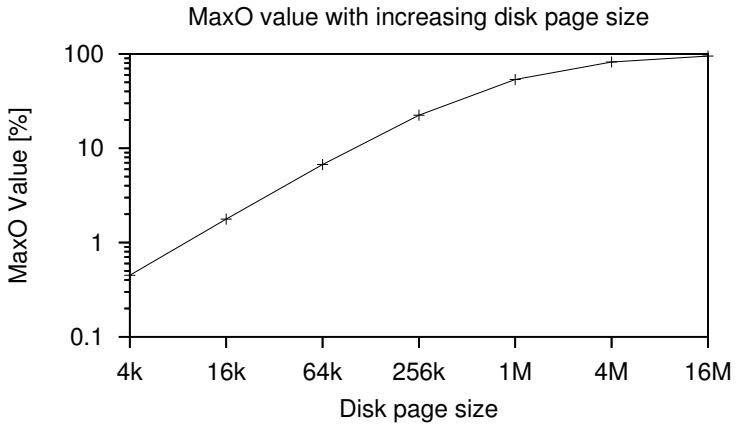


Figure 2.4: $MaxO$ threshold value with increasing disk page size

CPU-expensive. The cost of the insertion procedure increases in a significant manner due to the following reasons:

- All the MBRs from a directory node on the insertion path need to be checked in order to identify the one that requires the least enlargement. The cost of this task is linear with the number of MBRs from the node and with the number of dimensions. When the node capacity is large, an important number of MBRs need to be checked at each level of the indexing tree.
- When a node overflow occurs, the best partitioning of the set of MBRs has to be determined. The most efficient algorithm used to obtain a good partitioning is the one proposed by R*-tree [BKSS90]. This algorithm has a complexity of the form $C(2*d*(n+1)*log(n+1))$ where n represents the node capacity and d represents the number of dimensions. Therefore its cost increases significantly with the node capacity.

(ii.) *Measuring the AvgO value for spatial objects.* The second question is whether this cost model can be used in practice to index data sets of multidimensional objects with spatial extents. To answer this question we conducted the following experiment: We generated data sets of spatial objects uniformly distributed in multidimensional spaces of increasing dimensionality, from 2 to 40 dimensions. For each dimensionality, we inserted the objects in an R*-tree and we measured $AvgO$, the average overlap occurring between the MBRs of the nodes resulting from splits of directory nodes. We used three different disk page sizes: 4kBytes, 16kBytes, and 64kBytes.

Figure 2.5 illustrates the variation of the $AvgO$ value with increasing dimensionality. For 4kBytes pages, the $AvgO$ value initially grows from 74% in 2 dimensions to 89% in 12 dimensions, then gradually decreases to 46% in 40 dimensions. This behavior is related to the fact that the page capacity decreases with increasing dimensionality, from 204 objects per node in 2 dimensions to 12 objects per node in 40 dimensions. Although the average overlap tends to increase with the number of dimensions, when having fewer objects per node, the data objects are better grouped in leaf nodes. The spatial variation is less important in leaf nodes, which also decreases the overlap in

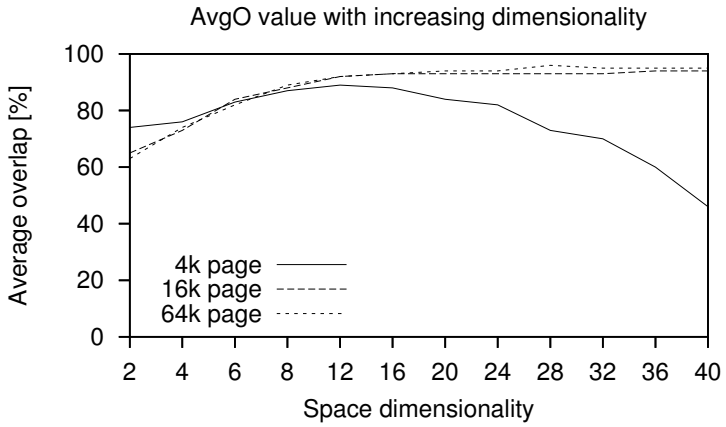


Figure 2.5: Overlap between MBRs of nodes resulting from directory node splits

directory nodes. In contrast, for pages of 16kBytes and 64kBytes, the number of objects per node remains important (i.e., 818 and 3276 objects in 2 dimensions, and respectively 50 and 202 objects in 40 dimensions), and the $AvgO$ value increases gradually from 65% and 63% in 2 dimensions to 94%a and 95% in 40 dimensions. The $AvgO$ value tends to increase with both increasing dimensionality and the page capacity.

More important, when comparing the $AvgO$ values from Figure 2.5 with the $MaxO$ values from Figure 2.4/Table 2.2, we notice that in all the cases $MaxO << AvgO$. This implies that no splits will occur in directory nodes. The resulting indexing structure will consist of only one very large supernode, the root node. Instead of a hierarchical directory, we obtain a single-level linear directory which is sequentially scanned during searches. Because all the leaf nodes are children of the root supernode, the query performance is determined by the linear scan of the directory and by the percentage of leaf nodes that are accessed. The leaf nodes are of fixed size and continue to be accessed in a random manner inflicting high I/O costs. Therefore this indexing structure cannot always ensure better average performance than sequential scan. This theoretical result was experimentally confirmed for both uniform and skewed data object distributions.

Conclusion on the X-tree cost model. Although X-tree uses a cost model to reduce the deterioration of the query performance caused by excessive random node accesses, the cost model is not practical when dealing with multidimensional objects with spatial extents because the average overlap resulting from directory node splits is much greater than the overlap threshold corresponding to the performance parameters. This induces the degeneration of the indexing structure to a single-level linear directory. The leaf nodes continue to be accessed in a random manner inflicting high I/O costs. The cost model of X-tree can ensure better performance than a classical R-tree, but it cannot always guarantee better average performance than sequential scan.

2.2.8 Conclusions

After studying the limitations of the existing multidimensional indexing methods, we can generally conclude that, in order to improve the object grouping and to globally

Spatial Access Methods			**Limitations** – considering spatial objects and disk–based storage
Space Partitioning based methods – point access methods adapted to spatial objects	Object clipping: Extended K–D–tree, Cell Tree		– 2/3 dimensions – small volume objects – poor storage utilization
	Overlapping regions: SKD–tree		– 2/3 dimensions – small volume objects
	Space transformation + point access methods: – object to point transformation – space–filling curve based object decomposition		– 2/3 dimensions – complex query decomposition and processing
Region Bounding methods – designed to support spatial objects	Dynamic	R–tree R+–tree R*–tree Hilbert R–tree	– few dimensions (< 6) – small volume objects – poor performance in higher dimensions (< sequential scan)
	Static – better performance than dynamic relatives	Packed R–tree Packed Hilbert R–tree	– few dimensions (< 6) – small volume objects – only for static datasets – poor performance in higher dimensions (< sequential scan)
	Cost–based – cost model embedding I/O and CPU parameters	X–tree – data–adaptive	– cost model only based on the data distribution – cost model not practical for spatial objects – poor performance in high dimensions (< sequential scan)
		Adaptive R–tree – query–adaptive	– 2/3 dimensions (impractical in higher dimensions) – not really adapted for spatial objects
Adaptive Clustering – our approach	Cost–based – designed for spatial objects – cost model embedding I/O and CPU system parameters – data–adaptive and query–adaptive – dynamically adapt to changes in data and query distributions – better search performance than sequential scan in virtually every case		

Figure 2.6: Spatial access methods: characteristics and limitations

ensure better average search performance than sequential scan, it is very important for the indexing solution to meet the following requirements:

- The object grouping should be assisted by a cost model taking into account the performance characteristics of the execution platform and considering both data and query distributions
- The indexing solution should be able to dynamically accommodate important changes that might occur over time in data or query patterns
- The cost model should apply to both leaf nodes and directory nodes

The alternative clustering approach that we propose is meant to follow these requirements.

Figure 2.6 summarizes the limitations of most existing multidimensional indexing methods supporting spatial objects, and outlines the main characteristics of the alternative clustering solution that we propose.

2.3 Principles and characteristics of our clustering solution

As an alternative to existing multidimensional indexing methods, we propose a cost-based query-adaptive clustering solution suitable for multidimensional objects with spatial extents and meant to improve the average performance of spatial range queries.

Our object grouping in clusters drops many properties of classical tree-based indexing structures (i.e., tree height balance, balanced node splits, and minimum bounding in all dimensions) in favor of a cost-based object clustering meant to ensure better average search performance than sequential scan in virtually every case.

The cost model supporting the object clustering takes into account the performance characteristics of the execution platform and relies on both data and query distributions to improve the object grouping and thus the average query performance.

The driving principles and the main characteristics of our clustering approach are:

- *Cost-based clustering*

  – The cost model is used to evaluate the average search performance of existing clusters and to estimate it for future cluster candidates in order to support, on the one hand, creation of new profitable clusters, and removal of older inefficient clusters, on the other hand.

  – The cost model integrates different system parameters affecting the spatial query execution. These parameters are determined by the performance characteristics of the execution platform (i.e., microprocessor, main memory, and external support) and depend on the storage scenario adopted for the spatial database (i.e., disk-based storage or main memory storage). The main system parameters embedded in the cost model are the disk access/seek time, the disk transfer rate, the memory access time and the object check rate.

  – With respect to the cost model, new clusters are only created when their presence is expected to improve the average query performance, while inefficient clusters whose profitability has decreased as result of changes in data or query distributions become subjects for restructuring operations. The cost model is meant to ensure for the set of clusters better average search performance than sequential scan in virtually every case.

- *Data-adaptive clustering*

  – The object clustering is based on a grouping criterion suitable for multidimensional objects with spatial extents. Our grouping criterion abandons the minimum bounding in all dimensions and groups spatial objects with similar intervals (locations and extents) in a reduced subset of dimensions, namely the most selective and discriminatory dimensions and domain regions relative to the query distribution.

  – The grouping dimensions and intervals are represented in the cluster signature. The cluster signature is used to decide if an object can become member of the cluster and if a spatial query needs to explore the cluster. When the cluster is explored during a spatial query, all the data objects that are members of the cluster are checked against the spatial selection criterion.

– To identify the best grouping dimensions and intervals, the grouping criterion is employed to deterministically partition each cluster into a number of candidate subclusters representing future cluster candidates. The number and the signatures of the candidate subclusters are accordingly determined by the grouping criterion. This enables maintenance of data and query statistics for the future cluster candidates.

– To take into account the spatial data distribution, for each existing cluster we maintain statistics regarding the number of data objects that are members of the cluster. Similarly, for all the candidate subclusters associated with the existing clusters we maintain statistics regarding the number of data objects qualifying for the corresponding subclusters.

– Most of the existing indexing solutions can be considered as data-adaptive. However, our clustering method relies on the cost model to promote the clusters with the best performance gains, and can dynamically adapt the object clustering to important changes that might occur over time in the distribution of the data objects as result of insertions and deletions.

- *Query-adaptive clustering*

  – Our object clustering is *query-adaptive* because the cost model takes into consideration the query distribution in order to support clustering decisions such as creation of new profitable clusters and removal of inefficient clusters. This is an important contribution because most existing cost-based indexing solutions assume that the query distribution follows the data distribution, which is not the case in many practical applications.

  – To take into account the spatial query distribution, for each existing cluster we maintain statistics regarding the number of spatial queries exploring the cluster. Similarly, for all the candidate subclusters associated with the existing clusters we maintain statistics regarding the number of spatial queries likely to explore the corresponding subclusters. The percentage of queries accessing a cluster over a period of time relative to the total number of queries addressed to the database system allows one to estimate the cluster access probability.

  – Together with the system performance parameters (i.e., cluster access cost, cluster read cost and object check cost), the cluster access probability enables the cost model to estimate the average search performance corresponding to a given cluster. Thanks to the cost model, our clustering method is able to dynamically adapt the object clustering to important changes that might occur over time in the distribution of the query objects.

The rationales of our cost-based data- and query-adaptive clustering strategy are illustrated in the following example.

Example 4 This example is based on Figure 2.7. Given the collection of 2-dimensional spatial objects $\{O_1, O_2, ..., O_8\}$ depicted in Figure 2.7-(A), we consider three alternative storage/indexing methods aiming to improve the average performance of spatial range queries as follows:

1. The first method consists of a sequential storage for the collection of data objects as depicted in the right side of Figure 2.7-(A).

2. The second method consists of an object organization based on an R-tree with 4 entries per node as depicted in Figure 2.7-(B).

   The R-tree has two leaf nodes, each of which storing four objects per node. The minimum bounding rectangles R_1 and R_2 of the two leaf nodes are stored in the root node. For the given set of spatial objects, inserted in the sequence indicated by the numbering order, the R-tree configuration depicted in the figure represents the best configuration that can result. We note that due to construction constraints, data objects that are very dissimilar, such as O_1 and O_3 or O_2 and O_8, fall in the same groups.

3. The third method consists of an object organization in clusters as depicted in Figure 2.7-(C).

   We have three clusters represented by the signatures C_1, C_2 and C_3. These clusters were obtained combining three grouping criteria:

   (a) A data-adaptive criterion which consists of grouping objects that are spatially similar with respect to their locations and extents in the data space

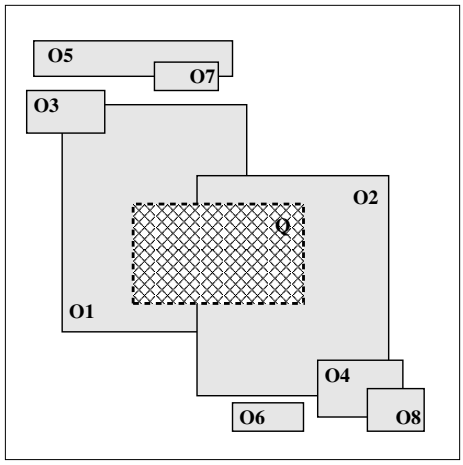
   (b) A query-adaptive criterion which consists of grouping objects that have similar access probabilities with respect to the distribution of the query objects

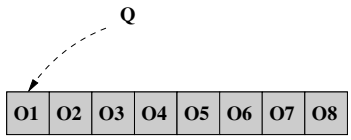
   (c) A cost-based criterion which consists of maintaining clusters that are profitable for the average query performance

   The first cluster C_1 regroups the data objects O_3, O_5 and O_7 situated in the top left region of the data space. The second cluster C_2 regroups the data objects O_4, O_6 and O_8 situated in the bottom right region of the data space. The third cluster C_3 regroups the data objects O_1 and O_2 occupying large areas in the center of the data space. To make our point, we assume that most of the spatial queries addressed to the database are falling in the hashed region denoted by Q. With respect to our query assumption, the top and the bottom regions of the data space are very seldom accessed by spatial queries, while the center region is very often accessed. As a result, the clusters C_1 and C_2 are seldom explored during spatial selections, so they contribute to improve the average query performance by avoiding to check a large fraction of data objects. The cluster C_3 is frequently explored, but it also contributes to improve the average query performance because only a small fraction of data objects need to be checked.

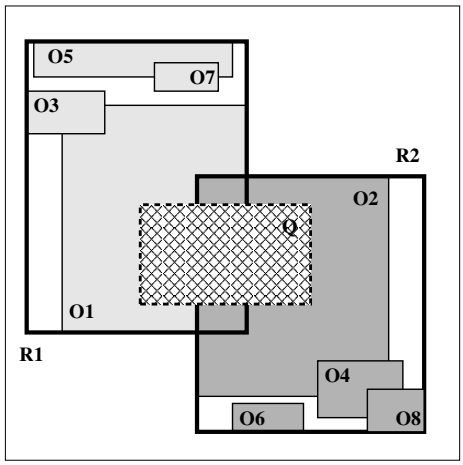
In the right sides of the figures, we have colored in gray the objects that are checked when answering a spatial query like the one represented by Q. The following observations can be made:

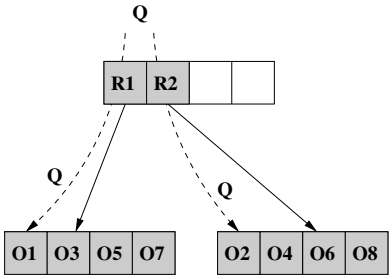
- The first storage method verifies all the data objects in a sequential manner.
- The R-tree based method checks the two MBRs from the root node and explores both leaf nodes because the query window overlaps both MBRs. As a result, all the data objects are finally verified. In this case, the global query cost is higher than



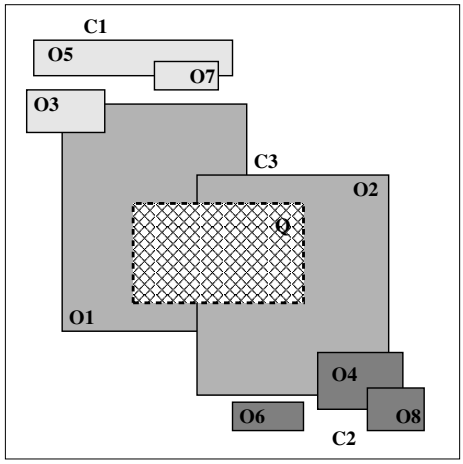


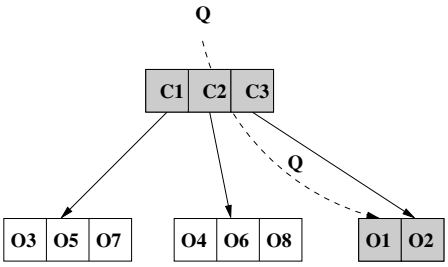
(A). Sequential object storage





(B). R−tree based object organization





(C). Data and query adaptive object organization

Figure 2.7: Example illustrating the rationales of our clustering approach

the sequential scan because: (1) the MBRs from the higher levels of the R-tree are also checked, and (2) the access to the leaf nodes is rather random, which inflicts higher access costs. Therefore the R-tree based organization is not efficient in this particular situation.

- The adaptive clustering method checks the three cluster signatures, and only the two objects from the cluster C_3 whose only signature match the query window. This clustering configuration is only efficient if the overhead cost caused by checking the cluster signatures and by randomly accessing the explored clusters does not exceed the performance gain resulting thanks to the smaller fraction of objects verified in the end.

As shown in our example, a clustering approach taking into account both data and query distributions, corroborated by a suitable cost model to monitor the average search performance, could lead to better object grouping with improved query performance results.

The challenging objective of our clustering solution is how to group the spatial objects into clusters guaranteed to improve the average performance of the spatial queries. A deterministic grouping criterion suitable for multidimensional objects with spatial extents is required to enable maintenance of data and query statistics, as well as a reliable cost model embedding the performance characteristics of the execution platform and taking into account both data and query distributions.

2.4 Conclusions

In this chapter we presented the problem that we address, we reviewed the related work with emphasis on the existing solutions applicable to our problem, and we outlined the driving principles and characteristics of our alternative solution.

The problem that we address belongs to the multidimensional data indexing domain. It consists of indexing large and dynamic collections of spatial objects with many dimensions and with possibly large extents over dimensions, to efficiently answer spatial range queries like intersection, containment, enclosure and similar-shape queries. A number of general application requirements have to be met such as scalability, search performance, update performance and adaptability to data and query distribution and to the performance characteristics of the execution platform. At once meeting all these requirements represents a great challenge for the multidimensional data indexing domain.

We reviewed the related work from the multidimensional data indexing domain. We presented the characteristics and the limitations of the existing indexing solutions, with emphasis on those suitable for multidimensional objects with spatial extents. We focused our attention on the evolution of the indexing methods based on the R-tree technique, specifically designed to support spatial objects. We showed that a number of requirements specific to our target applications, like the need to manage high-dimensional data, the need to handle data objects with possibly large extents over dimensions, and the need to cope with spatial range queries, are highly impractical for classical R-tree methods. The main reason is that, in such cases, the overlap between minimum bounding rectangles is important at node level, and an excessive number of nodes need to be explored

during spatial selections. The nodes are accessed in a random manner, which inflicts high I/O costs and deteriorates the search performance so much that a naive approach like sequential scan can be more efficient. To alleviate the performance degradation, some R-tree extensions were proposed as trade-off solutions between random access and sequential scan. They rely on cost models embedding the performance characteristics of the execution platform to adapt the node page sizes according to the data distribution like in X-tree and DABS-tree, and to the query distribution like in Adaptive R-tree. We analyzed the features and the cost models of these latter techniques and also emphasized their limitations.

After studying the limitations of the existing multidimensional indexing solutions, we concluded that an efficient access method should have its object grouping assisted by a cost model embedding the performance characteristics of the execution platform and considering both data and query distributions. As an alternative to existing indexing methods, we propose a cost-based query-adaptive clustering solution suitable for multidimensional objects with spatial extents and designed to improve the average search performance of spatial range queries. The driving principles and the characteristics of our clustering method are outlined in the last section of this chapter, together with an example meant to illustrate the rationales of our approach. Indeed, our clustering strategy is based on a cost model embedding the performance parameters of the execution platform and considering both data and query distributions as means to improve the object grouping and to ensure for spatial range queries better average search performance than sequential scan in virtually every case. The object clustering relies on a grouping criterion suitable for multidimensional objects with spatial extents and used to deterministically partition each existing cluster into a number of candidate subclusters representing future cluster candidates. Data and query statistics are maintained for the existing clusters and for the future cluster candidates in order to support clustering decisions such as creation of new profitable clusters and detection and removal of older inefficient clusters. The clustering strategy, the object grouping criterion, and the cost model supporting the clustering decisions are the main elements of our clustering solution. They are presented in the following chapter.

Chapter 3

Cost-based query-adaptive clustering

The main contribution of our work consists of a cost-based query-adaptive clustering solution for large and dynamic collections of multidimensional objects with spatial extents, meant to improve the average performance of spatial range queries like intersection, containment, enclosure, and similar-shape queries. Our object clustering is assisted by a cost model taking into account the performance characteristics of the execution platform and the spatial distributions of the data objects and of the query objects. The clustering solution aims to meet a number of general application requirements regarding the following aspects: scalability, search performance, update performance, and adaptability to data and and query distribution changes. In this chapter we present the main elements of our clustering approach: the clustering strategy, the object grouping criterion, and the cost model supporting the clustering decisions.

Chapter organization The chapter is organized as follows: Some useful terms and definitions are first introduced in Section 3.1. The clustering strategy and the clustering process are presented in Section 3.2. The object grouping criterion enabling our clustering method is described in Section 3.3. The cost model supporting the clustering decisions is detailed in Section 3.4.

3.1 Preliminary definitions

We now introduce some useful terms and definitions to facilitate the rest of the reading.

Cluster The term *cluster* is used to denominate a group of spatial objects sharing in common a number of properties or characteristics. Each cluster has a signature resuming the grouping characteristics and a storage space where the data objects belonging to the cluster are physically stored together. Depending on the storage scenario adopted for the spatial database, the storage space of a cluster can be in main memory or on secondary memory (disk). When a cluster is explored during the execution of a spatial query, all the data objects that are members of the cluster are accessed and checked

against the spatial selection criterion. As part of our clustering strategy, for performance reasons, the storage space of a cluster has to be contiguous. The contiguous storage is required to ensure sequential access to the data objects belonging to the cluster, during cluster explorations. Sequential data access can significantly improve the query execution performance, notably when the storage support is on secondary memory.

Cluster signature The grouping characteristics shared in common by the spatial objects belonging to a cluster are represented in the *cluster signature*. Thus each cluster is associated with a cluster signature. At this point, we only provide a semantical definition for the cluster signature, namely an object resuming the cluster's grouping characteristics. The cluster signature can be viewed as a class, while the spatial objects belonging to the cluster as class instances. The cluster signature is used to verify:

1. If a spatial object can become a member of the cluster – Only the spatial objects matching the cluster's signature can become members of the cluster;
2. If the cluster needs to be explored during the execution of a spatial query – Only the clusters whose signatures satisfy the spatial selection criterion (intersection, containment, enclosure, or similar shape) with respect to the query object are subjects for individual member examination.

An exact definition for the cluster signature is provided in Section 3.3.3.

Cluster subsignature We also use the term of *cluster subsignature* to denote a cluster signature that represents a specialization of another cluster signature. From a semantical point of view, all the spatial objects satisfying the cluster subsignature also satisfy the cluster signature, but only a subset of the spatial objects satisfying the cluster signature satisfy the cluster subsignature. In other words, the cluster signature represents a generalization of the cluster subsignature. In our semantics, it is possible for a cluster signature to have several different cluster subsignatures. In the same time, a cluster signature can be the subsignature of several different cluster signatures.

Subcluster Based on the notion of cluster subsignature, a cluster whose signature represents a specialization (a subsignature) of another cluster's signature can be referred to as a *subcluster* of the second cluster. The only relation established between the signatures of the two clusters enables the objects belonging to the subcluster to become members of the cluster without no additional membership verification. The cluster/subcluster relation imposes no constraint on the storage spaces of the two clusters, which remain completely independent.

Candidate cluster The term of *candidate cluster* is used to denominate a virtual cluster, namely an abstract group of spatial objects. A candidate cluster is represented by a cluster signature resuming the grouping characteristics, but it has no associated storage space. Hence, it can not be used to physically store data objects. In our case, candidate clusters represent future cluster candidates and are used to gather statistics regarding the number of data objects qualifying as their members, and regarding the

number of spatial queries whose selection criteria match their signatures. A candidate cluster can become a normal cluster if a storage space is assigned to it for its future member objects. The process of turning a candidate cluster into a normal cluster is called *materialization of a candidate cluster*.

Clustering By *clustering* we understand a disjoint partitioning of a collection of spatial data objects into a number of clusters. Each cluster is represented by a cluster signature. A data object could at once satisfy the signatures of several clusters. However, the clusters are disjoint in terms of storage spaces and a data object can not be physically placed in more than one cluster. The clustering should allow insertion of new data objects. For this purpose, at least one cluster should exist that accepts any spatial object.

3.2 Clustering strategy

Our clustering solution consists of dynamically grouping the data objects into clusters. The data objects from a cluster are stored in a sequential manner in order to minimize the cluster exploration cost. Sequential data access can significantly improve the query performance, notably when the storage support is on secondary memory. The object clustering is assisted by a cost model meant to improve the average performance of spatial range queries and to ensure better search performance than sequential scan. The object clustering relies on a grouping criterion suitable for multidimensional objects with spatial extents. The grouping criterion is used to identify groups of spatial objects with similar characteristics (similar interval locations and extents) in a reduced subset of dimensions, namely the most selective and discriminatory dimensions and domain regions relative to the query objects. The grouping dimensions and intervals are represented in the cluster signature. To identify the best grouping dimensions and intervals, each existing cluster is deterministically partitioned into a number of candidate subclusters representing future cluster candidates. Each candidate subcluster has a signature that represents a specialization of the cluster's signature in one of the possible dimensions. Data and query statistics are maintained for the existing clusters and for the future cluster candidates. These statistics are embedded in the cost model together with a number of system parameters affecting the query execution (disk seek/access time, disk transfer rate, memory access time and object check rate). The cost model is used to evaluate the search performance of existing clusters and to estimate it for future cluster candidates. A cluster is considered profitable when its presence contributes to improve the average query performance. The evaluation of the cluster search performance allows one to identify the most profitable future candidate clusters, and to detect the older clusters having lost their profitability as result of changes in the data or in the query distribution. The cost model is accordingly used to support creation of new profitable clusters and removal of inefficient clusters.

3.2.1 Outline of the clustering strategy

The clustering process is accomplished by recursively regrouping objects from existing clusters into new clusters by means of *cluster splits*, and by restructuring inefficient clusters through *cluster merges*. When performing a cluster split, subgroups of objects

from an existing cluster are extracted and relocated into new clusters. The different ways in which the data objects from a cluster can be grouped in new (sub)clusters are provided by the *clustering function* whose role is to implement the object grouping criterion. With respect to the cost model, cluster splits are only performed when the newly created clusters are expected to improve the average query performance. On the other hand, older clusters that are no longer profitable due to changes in the data or in the query distribution become subjects for merge operations: The inefficient clusters are withdrawn from the database and their member objects are relocated to the clusters representing their direct ancestors in the clustering hierarchy.

Our clustering strategy is assisted by three important operational functions:

1. *Clustering function*

   The clustering function implements the object grouping criterion providing the ways in which the data objects from an existing cluster are grouped into a number of candidate subclusters. The candidate subclusters represent future cluster candidates and correspond to different dimensions and interval regions. This function enables maintenance of data and query statistics for the future cluster candidates.

2. *Materialization benefit function*

   The materialization benefit function is primarily used to support creation of new profitable clusters. This function is based on the cost model and relies on data and query statistics to evaluate the impact of a new cluster on the average query performance. The materialization benefit function is employed to identify the candidate subclusters promising the best profits with respect to the average query performance.

3. *Merge benefit function*

   The merge benefit function is primarily used to support the detection and removal of inefficient clusters. This function is also based on the cost model and relies on data and query statistics to evaluate the impact of a cluster merge on the average query performance.

3.2.2 Application of the clustering strategy

Initially, the collection of data objects is stored in a single cluster called *root cluster*. The signature of the root cluster is set to the most general cluster signature, chosen to accept any spatial object. Since the signature of the root cluster makes no discrimination among spatial objects, all the spatial queries addressed to the system are exploring the root cluster. Thus the access probability of the root cluster is always 1.

At root cluster creation we invoke the *clustering function* to establish the signatures of the potential subclusters of the root cluster. The clustering function provides the ways in which the objects from the root cluster can be grouped in different subclusters. The candidate subclusters have no physical support (no storage space for data objects), but they represent future cluster candidates. To support the creation of new (profitable) clusters, performance indicators (data and query statistics) are attached to the root cluster and to all its candidate subclusters. For the root cluster we record the number of member objects and the number of visiting spatial queries. For a candidate subcluster we

record the number of data objects matching the signature of the candidate subcluster, and the number of spatial queries virtually visiting the candidate subcluster (spatial queries whose selection criteria match the signature of the candidate subcluster).

Cluster split The cluster split represents the first possible cluster restructuring operation. A cluster split is achieved by materializing a number of candidate subclusters of the considered cluster (initially, the root cluster is considered). The candidate subclusters selected for materialization are decided based on the *materialization benefit function*. This function applies onto each candidate subcluster and makes use of the cost model to evaluate the performance profit expected from the possible materialization of the considered candidate. Thus the materialization benefit function is used to identify the candidate subclusters with the best expected profits. These candidates become subjects for materialization.

The materialization of a candidate subcluster consists of two actions: First, a new cluster with the corresponding signature is created, and all the objects matching this signature are relocated from the original cluster to the new cluster. Second, the clustering function is applied on the signature of the new cluster to establish the subsignatures of the candidate subclusters of the new cluster. Performance indicators are attached to the new cluster and to all the candidate subclusters of the new cluster. They will be used to gather statistics in order to support future cluster restructuring decisions.

During cluster splits, the candidate subclusters with the best expected profits are materialized, so the original clusters get partitioned into a number of smaller clusters. As result of recursive cluster splits, we obtain a hierarchy of clusters, where each cluster is associated with a signature, a set of candidate subclusters, and the corresponding performance indicators.

Cluster merge The cluster merge represents the second possible cluster restructuring operation. When the separate management of an existing cluster becomes inefficient as result of changes in the data or in the query distribution, the given cluster is withdrawn from the database, and its objects are transferred back to the parent cluster. By parent cluster we understand the cluster representing the direct ancestor in the hierarchy of clusters. The merge between a cluster and its parent cluster is decided based on the *merge benefit function*. This function makes use of the cost model to evaluate the impact of the merge operation on the average query performance. Thus a merge operation is only triggered when it is expected to bring a performance gain.

The cluster merge allows the object clustering to dynamically adapt to important changes that might occur over time in the data or in the query distribution. To enable merge operations, each cluster maintains a reference to the direct parent, and a list of references to the child clusters. The root cluster has no parent and it can not be withdrawn from the database. Its role is to host the data objects matching no other cluster signatures.

Cluster restructuring invocation Decisions of cluster restructuring (cluster splits and cluster merges) are made periodically for all the existing clusters. A cluster restructuring operation is only considered when sufficient query statistics have been gathered

to properly support the restructuring decision.

The object grouping criterion and the clustering function implementing the grouping criterion will be presented in the following Section 3.3. The cost model supporting the clustering strategy, together with the two benefit functions will be presented in Section 3.4.

3.3 Object grouping criterion

The data objects from our target applications are multidimensional objects with spatial extents. Our clustering solution requires a grouping criterion suitable for this type of objects. In particular, the object grouping should take into account the spatial locations and the spatial extents of the data objects. This section presents the grouping criterion chosen to support our object clustering. We first present the multidimensional space model used to represent the data objects (Section 2.1.1). We then define a generic grouping model for multidimensional objects with spatial extents (Section 3.3.2). Based on this model, we define the notions of cluster signature and cluster subsignature (Section 3.3.3). We further discuss several possible grouping methods suitable for multidimensional spatial objects (Section 3.3.4). We finally present the clustering function that implements the grouping method chosen to support our clustering solution (Section 3.3.5).

3.3.1 Multidimensional space data representation model

As already shown, the data objects manipulated in our target applications are commonly defined as sets of <*attribute, interval of values*> associations. They can be represented as multidimensional objects with axes-parallel spatial extents in a multidimensional data space where each dimension stands for a different space axis. Such objects are also referred to as multidimensional spatial objects or multidimensional extended objects.

Example 5 The multidimensional space data representation model is illustrated in Figure 3.1. The figure depicts a three-dimensional spatial object corresponding to the following small ads subscription: "Looking for an apartment within 30 miles from Newark, with a rent price between 400$ and 700$, and having between 3 and 5 rooms". The three axes from the figure correspond to the three subscription attributes: the distance from Newark (where the given subscription ranges between 0 and 30), the rent price (where the subscription ranges between 400 and 700), and the number of rooms (where the subscription ranges between 3 and 5).

Let N_d be the number of dimensions of the multidimensional space used to represent the data objects. According to our data representation model, a multidimensional spatial object specifies an interval for each dimension and can be described as follows:

$$o = \{d_1[a_1, b_1], d_2[a_2, b_2], \ldots, d_{N_d}[a_{N_d}, b_{N_d}]\}$$

where the interval $[a_i, b_i]$ represents the extent of the spatial object o in the dimension $d_i, \forall i \in \{1, 2, \ldots, N_d\}$.

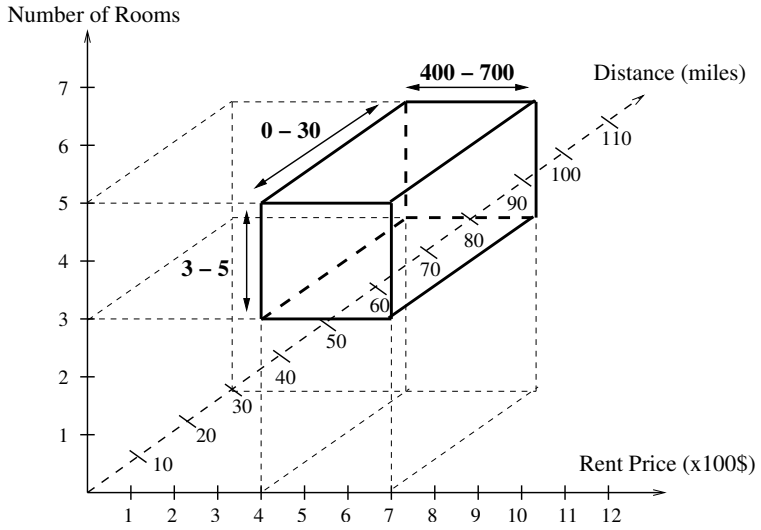


Figure 3.1: Multidimensional data space representation model

To simplify our definitions, we will assume that the data space is normalized, namely each dimension takes values in the domain $[0, 1]$:

$$0 \leq a_i \leq b_i \leq 1, \forall i \in \{1, 2, \ldots, N_d\}.$$

As we deal with multidimensional objects with spatial extents, our object clustering requires a grouping criterion suitable for this type of objects. In particular, the object grouping should take into account the spatial locations and extents of the data objects.

3.3.2 Generic grouping model for multidimensional spatial objects

To define a generic grouping model for spatial objects, we first introduce the notion of *similar intervals*, which applies to intervals from the same dimension. We then rely on the notion of similar intervals to support the definition of *similar multidimensional spatial objects*.

Similar intervals By *similar intervals* we understand intervals of comparable sizes, located in the same domain region. To precise the notion of interval similarity we define two *intervals of variation* where the starts and the ends of the intervals considered as similar can take values. So having two intervals of variation

$$[a^{min}, a^{max}] \text{ and } [b^{min}, b^{max}]$$

with $0 \leq a^{min} \leq b^{max} \leq 1$ and $0 \leq b^{min} \leq a^{max} \leq 1$, all the intervals starting in $[a^{min}, a^{max}]$ and ending in $[b^{min}, b^{max}]$ are considered similar with respect to these two intervals of variation.

Example 6 The intervals I_1, I_2 and I_3 from Figure 3.2 are similar with respect to the intervals of variation $[0, 0.25]$ and $[0.5, 0.75]$. In fact, all the intervals starting in the first quarter of the domain, and ending in the third quarter of the domain are considered similar with respect to the two domain quarters chosen as intervals of variation.

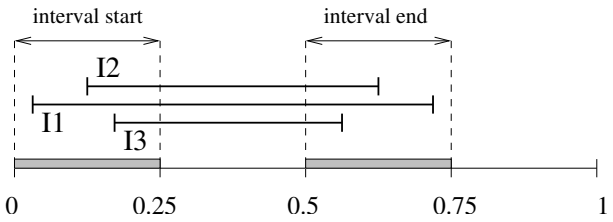


Figure 3.2: Similar intervals

We next present two special cases:

1. The two intervals of variation are identical:

$$[a^{min}, a^{max}] = [b^{min}, b^{max}] = [a, b]$$

   In this case, all the subintervals of the interval $[a, b]$ are considered similar with respect to this common interval of variation.

2. The two intervals of variation are both set to the full domain:

$$[a^{min}, a^{max}] = [b^{min}, b^{max}] = [0, 1]$$

   In this case, according to our definition, all the intervals defined in the domain $[0, 1]$ are considered similar.

The notion of interval similarity applies to intervals from the same dimension. One can rely on this notion to partition a collection of intervals into groups of similar intervals, where each group is characterized by two intervals of variation. However, our data objects are defined in a multidimensional space where an interval is provided for each dimension. A grouping model for multidimensional spatial objects has to take into account the spatial locations and the spatial extents of the data objects in all the dimensions. Based on the notion of similar intervals, we next define the notion of *similar multidimensional spatial objects*.

Similar multidimensional spatial objects Two or several multidimensional spatial objects are considered *similar* if their corresponding intervals are similar in all the dimensions. In each dimension the interval similarity is specified by two intervals of variation. So having the set of intervals of variation

$$\{[a_i^{min}, a_i^{max}] \text{ and } [b_i^{min}, b_i^{max}]\}$$

with $0 \leq a_i^{min} \leq b_i^{min} \leq 1$ and $0 \leq b_i^{min} \leq b_i^{max} \leq 1, \forall i \in \{1, 2, \ldots, N_d\}$, where i stands for the dimension d_i, all the spatial objects that in dimension d_i define intervals starting in $[a_i^{min}, a_i^{max}]$ and ending in $[b_i^{min}, b_i^{max}]$, $\forall i \in \{1, 2, \ldots, N_d\}$, are considered similar with respect to the given set of intervals of variation.

Example 7 The 2-dimensional spatial objects O_1, O_2 and O_3 from Figure 3.3 are similar with respect to the intervals of variation $[a_1^{min}, a_1^{max}]$ and $[b_1^{min}, b_1^{max}]$ corresponding to the dimension d_1, and $[a_2^{min}, a_2^{max}]$ and $[b_2^{min}, b_2^{max}]$ corresponding to the dimension d_2. In fact, all the rectangles having their corners in the four regions colored in gray are considered similar with respect to the given set of intervals of variation.

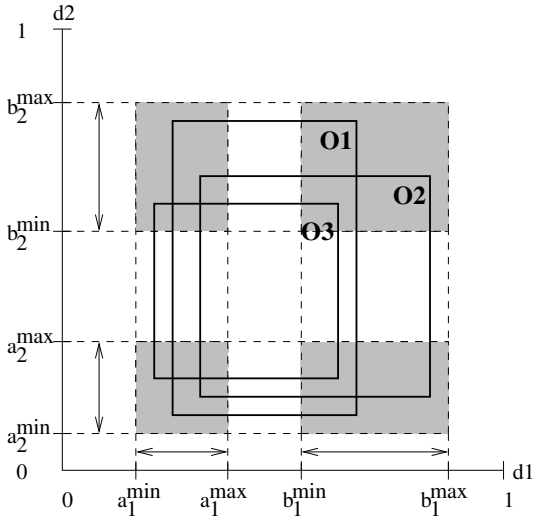


Figure 3.3: Similar multidimensional spatial objects

Generic grouping model We can now rely on the notion of object similarity to partition a collection of multidimensional spatial objects into different groups, where a group is characterized by the set of intervals of variation used to define the interval similarity in each dimension. In our terminology, a group of objects is referred to as a *cluster*. For a cluster, the set of intervals of variation characterizing the object similarity is represented in the *cluster signature*. Based on this generic grouping model, a definition for the cluster signature is provided in the following section.

3.3.3 Cluster signature and cluster subsignature

Cluster signature The *cluster signature* is defined as follows:

$$\sigma = \{\ d_1[a_1^{min}, a_1^{max}] : [b_1^{min}, b_1^{max}],\quad d_2[a_2^{min}, a_2^{max}] : [b_2^{min}, b_2^{max}],$$

$$\ldots,\quad d_{N_d}[a_{N_d}^{min}, a_{N_d}^{max}] : [b_{N_d}^{min}, b_{N_d}^{max}]\ \}$$

where σ represents the signature of a cluster regrouping spatial objects whose intervals in dimensions d_i start in $[a_i^{min}, a_i^{max}]$ and end in $[b_i^{min}, b_i^{max}]$, $\forall i \in \{1, 2, \ldots, N_d\}$.

Example 8 In our clustering model, we have a special cluster called *root cluster* whose role is to host the data objects matching no other clusters. The signature of the root cluster must be defined so to accept any spatial object as a member of this special cluster. This means that all the spatial objects have to be similar with respect to the set of intervals of variation characterizing the signature of the root cluster. The only way to fulfill this requirement is to set the intervals of variation corresponding to the root cluster signature to full domains in all dimensions:

$$\sigma_{root} = \{d_1[0,1] : [0,1],\quad d_2[0,1] : [0,1],\quad \ldots,\quad d_{N_d}[0,1] : [0,1]\}$$

Indeed, according to our definition, any two intervals are similar in any of the N_d dimensions, so all the spatial objects are similar with respect to the root cluster signature.

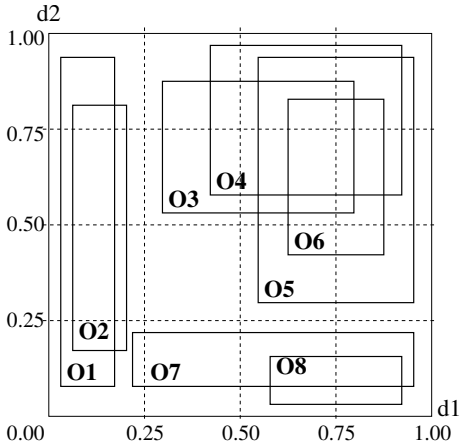


Figure 3.4: An example to illustrate some possible clusters of spatial objects

Example 9 Considering the data objects $O_1, O_2, ..., O_8$ from the 2-dimensional space depicted in Figure 3.4, here are some examples of possible clusters together with their corresponding signatures:

$\sigma_{root} = \{d_1[0,1] : [0,1],\ d_2[0,1] : [0,1]\}$ regroups all the objects;

$\sigma_1 = \{d_1[0,0.25) : [0,0.25),\ d_2[0,0.25) : [0.75,1]\}$ regroups the objects O_1 and O_2;

$\sigma_2 = \{d_1[0.25,0.50) : [0.75,1],\ d_2[0.50,0.75) : [0.75,1]\}$ regroups the objects O_3 and O_4;

$\sigma_3 = \{d_1[0.50,0.75) : [0.75,1],\ d_2[0,1] : [0,1]\}$ regroups the objects O_5, O_6 and O_8;

$\sigma_4 = \{d_1[0.50,0.75) : [0.75,1],\ d_2[0.25,0.5) : [0.75,1]\}$ regroups the objects O_5, O_6;

$\sigma_5 = \{d_1[0,1] : [0,1],\ d_2[0,0.25) : [0,0.25)\}$ regroups the objects O_7 and O_8.

Cluster subsignature Given two cluster signatures

$$\sigma = \{\ (d_i[a_{\sigma_i}^{min}, a_{\sigma_i}^{max}] : [b_{\sigma_i}^{min}, b_{\sigma_i}^{max}]) \mid i \in \{1,2,...,N_d\}\ \}$$

and

$$\zeta = \{\ (d_i[a_{\zeta_i}^{min}, a_{\zeta_i}^{max}] : [b_{\zeta_i}^{min}, b_{\zeta_i}^{max}]) \mid i \in \{1,2,...,N_d\}\ \}$$

ζ is a *cluster subsignature* of σ if and only if $\forall i \in 1,2,...,N_d$

$$a_{\sigma_i}^{min} \leq a_{\zeta_i}^{min} \leq a_{\zeta_i}^{max} \leq a_{\sigma_i}^{max}$$

and

$$b_{\sigma_i}^{min} \leq b_{\zeta_i}^{min} \leq b_{\zeta_i}^{max} \leq b_{\sigma_i}^{max}.$$

According to the definition, all the data objects matching ζ will also match σ, while the converse is in general not true.

Example 10 Considering the cluster signatures σ_3 and σ_4 from the previous example, σ_4 is a subsignature of σ_3.

3.3.4 Object grouping methods

The generic grouping model allows us to form clusters of spatial objects based on the notion of object similarity. However, the object grouping itself is actually determined by the choice of the intervals of variation supporting the notion of interval similarity over dimensions. The choice of the intervals of variation represents the subject of a *grouping method*. Two grouping methods suitable for multidimensional spatial objects are considered next: The first grouping method is based on a space division in cells of equal volume, using all the dimensions at once. The second grouping method is based on a space division in slices of equal volume, using one dimension at a time.

i. Object grouping based on space division using all dimensions

This grouping method first divides the multidimensional data space into spatial cells of equal volume, using all the dimensions at once. Then each possible pair of spatial cells is considered as a grouping support for a cluster of similar objects.

The domain of values of each dimension is divided into a fixed number of equal length regions. As a result, the multidimensional data space gets partitioned into a number of spatial cells of equal volume. Each spatial cell is represented by an interval per dimension. Considering a pair of spatial cells, we have two intervals per dimension. One can use these pairs of intervals as intervals of variation to support a group of similar objects. In such a case, the two spatial cells supporting the object grouping will determine the cluster signature. According to the generic grouping model, all the spatial objects with two diagonally opposite corners falling in the two grouping spatial cells are considered similar. If the two spatial cells supporting the object grouping coincide, all the spatial objects falling completely in the corresponding cell are considered similar.

Example 11 This grouping method is illustrated in Figure 3.5. We consider a 2-dimensional data space containing 12 spatial objects as in Figure 3.5-A. In Figure 3.5-B we superpose a 2-dimensional grid over the data space. The spatial grid divides the data space using 4 equal length regions per dimension. The rest of the drawings show groups of similar objects determined by different choices of spatial cell pairs. The pairs of spatial cells supporting the object grouping are represented in dark gray. We notice the following groups of objects: O_1 and O_2 in Figure 3.5-C; O_7 and O_8 in Figure 3.5-D; O_5 and O_6 in Figure 3.5-E; O_9 and O_{10} in Figure 3.5-F. Each pair of grouping cells corresponds to a cluster signature. For instance, the cluster regrouping the objects O_1 and O_2 is represented by the following signature:

$$\sigma_{(1,2)} = \{d_1[0, 0.25) : [0.50, 0.75),\ d_2[0.25, 0.50) : [0.75, 1]\};$$

Similarly, the cluster regrouping the objects O_9 and O_{10} has the following signature:

$$\sigma_{(9,10)} = \{d_1[0, 0.25) : [0, 0.25),\ d_2[0, 0.25) : [0, 0.25)\}.$$

This grouping method can be used to support the partitioning of a collection of spatial objects into clusters based on the notion of object similarity defined before. However, the number of potential candidate clusters is very large, actually proportional with the

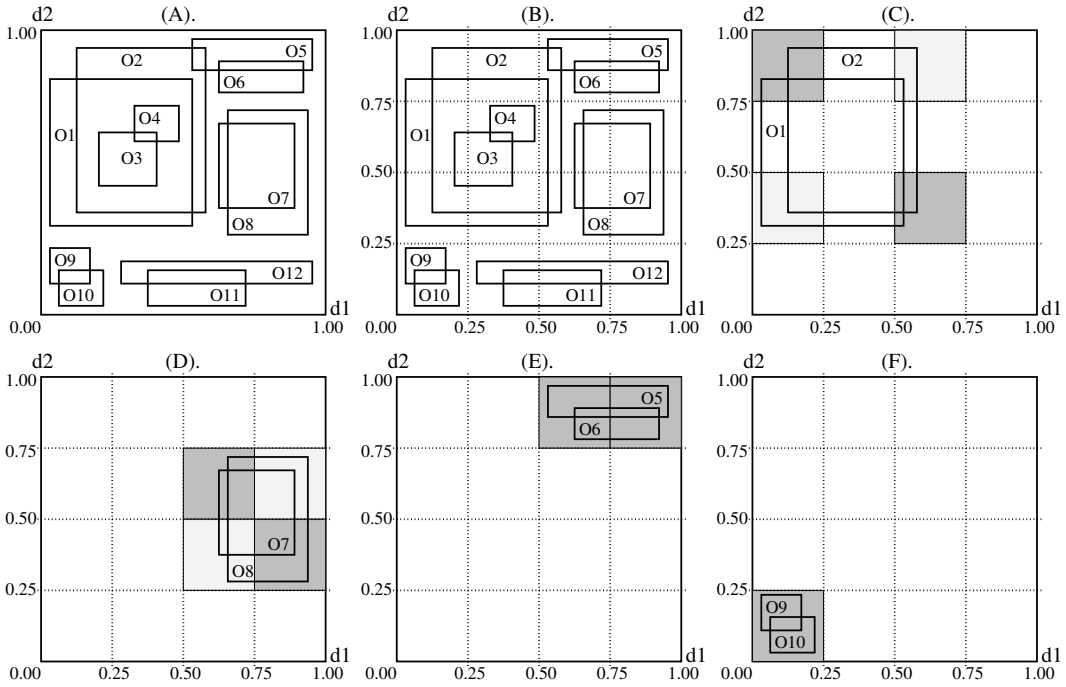


Figure 3.5: Object grouping based on space division using all dimensions

number of possible cell pair combinations. For a 2-dimensional space, the exact number of potential candidate clusters is given by the formula

$$[\frac{f \cdot (f+1)}{2}]^2$$

where f represents the number of divisions per dimension (also called domain division factor). Indeed, a number of $\frac{f \cdot (f+1)}{2}$ interval pair combinations can be formed in each dimension. Combining the intervals pairs between the 2 dimensions we obtain the formula above. In a multidimensional space, the number of potential candidate clusters is given by the formula

$$[\frac{f \cdot (f+1)}{2}]^{N_d}$$

where N_d represents the number of dimensions. Indeed, in a multidimensional space we have to consider all the combinations of interval pairs between the N_d dimensions.

The total number of potential candidate clusters generated by this grouping method is exponential with the number of dimensions. Considering, for instance, a 10-dimensional space, and using a domain division factor of 4 regions per dimension, we obtain 10 billion possible candidate clusters. To support cluster split decisions, our clustering solution needs to maintain performance indicators (data and query statistics) for all the candidate (sub)clusters. With this grouping method, maintaining performance indicators for all the candidate clusters would be highly impractical in spaces with more than a few dimensions. Moreover, in high dimensions the number of potential clusters would rapidly exceed the number of data objects. As a result, most of the candidate clusters would be either unoccupied or poorly populated. These are the main reasons for which we have adopted the second grouping method presented next.

ii. Object grouping based on space division using one dimension

This grouping method first divides the multidimensional data space into spatial slices of equal volume, using only one dimension at a time. Then each possible pair of spatial slices (from the same dimension) is considered as a grouping support for a cluster of similar objects.

We iteratively consider each dimension and divide its corresponding domain into a fixed number of equal length regions. As a result, for each dimension, the multidimensional space is partitioned into spatial slices of equal volume. A spatial slice is represented by an interval in the slicing dimension and by full domains in the other dimensions. Considering a pair of spatial slices as similarity support, all the spatial objects with two opposite $(N_d - 1)$-dimensional faces (or edges in 2 dimensions) falling in the two spatial slices are considered similar. When the two spatial slices supporting the object grouping coincide, all the spatial objects falling completely in the given slice are considered similar.

Example 12 This grouping method is illustrated in Figure 3.6 for a 2-dimensional space containing the same 12 objects as in the previous example. Figure 3.6-A shows the spatial slices obtained dividing the domain of the dimension d_1 by a factor of 4. The drawings from Figures 3.6-B to 3.6-D represent groups of similar objects determined by pairs of spatial slices corresponding to the dimension d_1: O_9 and O_{10} in Figure 3.6-B; O_1 and O_2 in Figure 3.6-C; and O_5, O_6, O_7 and O_8 in Figure 3.6-D. The pairs of spatial slices supporting the object grouping are represented in dark gray. Figure 3.6-E shows the spatial slices obtained dividing the domain of the dimension d_2. The drawings from Figures 3.6-F to 3.6-I represent groups of similar objects determined by pairs of spatial slices corresponding to the dimension d_2: O_5 and O_6 in Figure 3.6-F; O_1 and O_2 in Figure 3.6-G; O_3, O_7 and O_8 in Figure 3.6-H; O_9, O_{10}, O_{11} and O_{12} in Figure 3.6-I. Using this grouping method, the spatial objects can be clustered in several ways. For instance, O_1 and O_2 can be grouped into a cluster represented either by the signature

$$\sigma^1_{(1,2)} = \{d_1[0.00, 0.25) : [0.50, 0.75),\ d_2[0.00, 1.00] : [0.00, 1.00]\}$$

with respect to a pair of spatial slices corresponding to the dimension d_1, or by the signature

$$\sigma^2_{(1,2)} = \{d_1[0.00, 1.00] : [0.00, 1.00],\ d_2[0.25, 0.50) : [0.75, 1.00]\}$$

with respect to a pair of spatial slices corresponding to the dimension d_2.

The number of the potential candidate clusters generated by this second grouping method is given by the formula

$$\frac{f \cdot (f+1)}{2} \cdot N_d$$

where f represents the domain division factor, and N_d represents the number of dimensions. Indeed, $\frac{f \cdot (f+1)}{2}$ pairs of spatial slices can be formed for each of the N_d dimensions, which sums up to the formula above.

This time the total number of potential candidate subclusters keeps linear with the number of dimensions. Considering, for instance, a 10-dimensional space, and using a domain division factor of 4 regions per dimension, we obtain 100 possible candidate

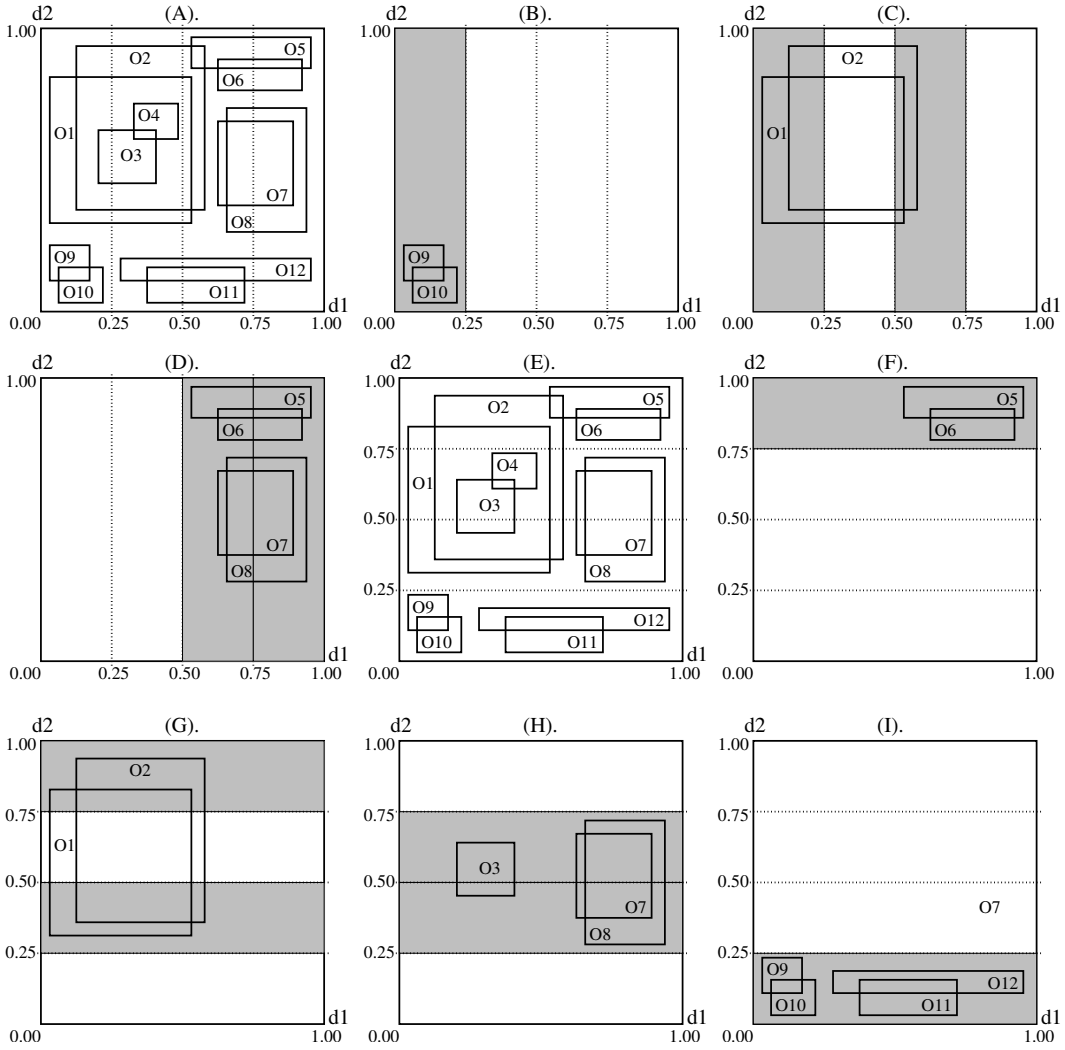


Figure 3.6: Object grouping based on space division using one dimension

clusters. Maintaining data and query statistics for all the potential candidate clusters becomes now feasible.

The limited number of candidate subclusters is the main consideration for which we have adopted this grouping method as support for our clustering approach. Another reason for this choice is that the object grouping is made on a "per dimension" basis. This is of practical interest because often in real applications some dimensions are more selective and more discriminatory than others with respect to the query objects. For a candidate subcluster, only one dimension actually determines the object grouping. So when examining different candidate subclusters, we implicitly make a "per dimension" object grouping analysis.

The grouping method based on space division using one dimension at a time is embedded in the clustering function to determine the candidate subclusters of the existing clusters. The clustering function is presented in the next section.

3.3.5 Clustering function

The role of the clustering function is to generate the set of signatures corresponding to the candidate subclusters of a given cluster. The signatures of the candidate subclusters are used to decide how many objects from the original cluster qualify for the corresponding candidate subcluster and how many queries visiting the original cluster would also visit the corresponding candidate subcluster. These two performance indicators are maintained for all the candidate subclusters and will serve as decision support for future splits of the original cluster.

In this context, a good clustering function should solve the following trade-off: On the one hand, the number of candidate subclusters should be sufficiently large to ensure good opportunities of clustering. On the other hand, if this number is too large, the cost of maintaining data and query statistics increases too much (performance indicators are maintained for each candidate subcluster). As a solution to this trade-off we adopted for our clustering function the grouping method based on a space division using one dimension at a time. We first provide a definition for the clustering function, then we present the way that our clustering function is implemented, and finally we present the properties of the clustering function.

Clustering function definition

Considering the signature σ_c of a cluster c, the clustering function γ produces the set of signatures $\{\sigma_s\}$ associated with the candidate subclusters $\{s\}$ of the cluster c. Formally,

$$\gamma(\sigma_c) \rightarrow \{\sigma_s\}$$

where $\forall \sigma_s \in \gamma(\sigma_c)$, any spatial object matching the signature σ_s (so qualifying as a member of the subcluster s of the cluster c) also matches the signature σ_c of the cluster c. This means that the signatures $\{\sigma_s\}$ of the candidate subclusters $\{s\}$ are subsignatures of the signature σ_c. The clustering function ensures a backward object compatibility in the hierarchy of clusters, which is necessary to enable merge operations between child and parent clusters. It is possible for a spatial object matching the signature σ_c of the cluster c to match the signatures σ_s of several subclusters s of the cluster c.

Clustering function implementation

Our clustering function works as follows: Given a cluster signature we iteratively consider each dimension. For each dimension, we divide both intervals of variation into a fixed number of equal length subintervals. We call *division factor* and note f the number of subintervals per dimension. We then replace the pair of intervals of variation of the cluster signature by each possible combination pair of subintervals. We have f^2 combination pairs of subintervals per dimension and thus f^2 cluster subsignatures for each dimension. In the case when the two intervals of variation of the selected dimension are identical, only $\frac{f \cdot (f+1)}{2}$ subinterval combination pairs are distinct because of the symmetry.

Example 13 We consider the following cluster signature

$$\sigma = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.50, 0.75) : [0.75, 1.00]\}$$

and apply the clustering function using a division factor $f = 4$:

- We first take the dimension d_1 and generate the sets of equal length subintervals corresponding to the two intervals of variation $[0.00, 0.25)$, identical in this case:

  $$S_1^{d_1} = S_2^{d_1} = \{[0.0000, 0.0625), [0.0625, 0.1250), [0.1250, 0.1875), [0.1875, 0.2500)\}$$

  We then generate all the possible pairs of subintervals between the two sets $S_1^{d_1}$ and $S_2^{d_1}$ and obtain the following cluster subsignatures:

  $$\begin{aligned}
  \sigma_1^{d_1} &= \{d_1[0.0000, 0.0625) : [0.0000, 0.0625),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_2^{d_1} &= \{d_1[0.0000, 0.0625) : [0.0625, 0.1250),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_3^{d_1} &= \{d_1[0.0000, 0.0625) : [0.1250, 0.1875),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_4^{d_1} &= \{d_1[0.0000, 0.0625) : [0.1875, 0.2500),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_5^{d_1} &= \{d_1[0.0625, 0.1250) : [0.0625, 0.1250),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_6^{d_1} &= \{d_1[0.0625, 0.1250) : [0.1250, 0.1875),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_7^{d_1} &= \{d_1[0.0625, 0.1250) : [0.1875, 0.2500),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_8^{d_1} &= \{d_1[0.1250, 0.1875) : [0.1250, 0.1875),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_9^{d_1} &= \{d_1[0.1250, 0.1875) : [0.1875, 0.2500),\ d_2[0.50, 0.75) : [0.75, 1.00]\};\\
  \sigma_{10}^{d_1} &= \{d_1[0.1875, 0.2500) : [0.1875, 0.2500),\ d_2[0.50, 0.75) : [0.75, 1.00]\};
  \end{aligned}$$

  There are 16 possible subinterval combination pairs, but only the 10 enumerated before are distinct because of the symmetry.

- We next consider the second dimension d_2. For the first interval of variation $[0.50, 0.75)$, we generate the corresponding set of equal length subintervals:

  $$S_1^{d_2} = \{[0.5000, 0.5625), [0.5625, 0.6250), [0.6250, 0.6875), [0.6875, 0.7500)\};$$

  For the second interval of variation $[0.75, 1.00]$, we generate its corresponding set of equal length subintervals:

  $$S_2^{d_2} = \{[0.7500, 0.8125), [0.8125, 0.8750), [0.8750, 0.9375), [0.9375, 1.0000]\};$$

  Generating all the pairs of subintervals between the two sets $S_1^{d_2}$ and $S_2^{d_2}$, we obtain 16 more candidate subclusters represented by the following signatures:

  $$\begin{aligned}
  \sigma_1^{d_2} &= \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.5000, 0.5625) : [0.7500, 0.8125)\};\\
  \sigma_2^{d_2} &= \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.5000, 0.5625) : [0.8125, 0.8750)\};\\
  \sigma_3^{d_2} &= \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.5000, 0.5625) : [0.8750, 0.9375)\};\\
  \sigma_4^{d_2} &= \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.5000, 0.5625) : [0.9375, 1.0000]\};\\
  \sigma_5^{d_2} &= \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.5625, 0.6250) : [0.7500, 0.8125)\};\\
  \sigma_6^{d_2} &= \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.5625, 0.6250) : [0.8125, 0.8750)\};
  \end{aligned}$$

$$\sigma_7^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.5625, 0.6250) : [0.8750, 0.9375)\};$$
$$\sigma_8^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.5625, 0.6250) : [0.9375, 1.0000]\};$$
$$\sigma_9^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.6250, 0.6875) : [0.7500, 0.8125)\};$$
$$\sigma_{10}^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.6250, 0.6875) : [0.8125, 0.8750)\};$$
$$\sigma_{11}^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.6250, 0.6875) : [0.8750, 0.9375)\};$$
$$\sigma_{12}^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.6250, 0.6875) : [0.9375, 1.0000]\};$$
$$\sigma_{13}^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.6875, 0.7500) : [0.7500, 0.8125)\};$$
$$\sigma_{14}^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.6875, 0.7500) : [0.8125, 0.8750)\};$$
$$\sigma_{15}^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.6875, 0.7500) : [0.8750, 0.9375)\};$$
$$\sigma_{16}^{d_2} = \{d_1[0.00, 0.25) : [0.00, 0.25),\ d_2[0.6875, 0.7500) : [0.9375, 1.0000]\}.$$

Properties of the clustering function

Number of subsignatures of a cluster signature. The clustering function applies on each of the N_d dimensions of the cluster signature. As a result, we obtain a number N_s of cluster subsignatures where

$$N_d \cdot \frac{f \cdot (f+1)}{2} \leq N_s \leq N_d \cdot f^2$$

In this inequality f represents the domain division factor. The last term of the inequality corresponds to the case where the pairs of intervals of variation from the cluster signature involve distinct intervals in all the dimensions. In this case, the number of distinct subinterval combination pairs is f^2 per dimension. The first term of the inequality corresponds to the case the pairs of intervals of variation from the cluster signature involve identical intervals in all the dimensions. In this case, the number of distinct subinterval combination pairs is $\frac{f \cdot (f+1)}{2}$ per dimension, smaller because of the symmetry. However, in all the cases, the number of cluster subsignatures, thus the number of candidate subclusters, keeps linear with the number of dimensions N_d.

Number of subsignatures matched by a spatial object satisfying a cluster signature. By the choice of the clustering function, a spatial object matching a cluster signature will also match a number of N_d cluster subsignatures of the original cluster signature, where N_d represents the number of dimensions of the data space. Indeed, the spatial object will match one and only one subsignature per dimension because, in each dimension, one and only one of the possible subinterval pair combinations will embed the extent of the spatial object in the given dimension. This property is important because it gives the number of candidate subclusters whose data statistics need to be updated when a data object is inserted, removed, or relocated to another cluster.

Object grouping based on a reduced subset of dimensions and domain regions. According to our clustering function, a cluster subsignature differs from its parent cluster signature in only one dimension. In the corresponding dimension, the intervals of variation of the cluster subsignature represent subintervals of the intervals of variation of the parent cluster signature. As a result, when descending in the clustering hierarchy, only one dimension has the power to discriminate from one level to the next one. The other dimensions are not discriminatory and cannot serve to decide data or query qualification. The clustering strategy privileges the materialization of the candidate subclusters promising the best performance profits with respect to the cost model. Therefore the object grouping corresponding to a cluster is determined by a reduced subset of dimensions and domain regions, namely the most selective and discriminatory ones relative to the query cost.

Thanks to the clustering function, each existing cluster is associated with a known number of candidate subclusters representing future cluster candidates. Statistics regarding data and query distributions are maintained for all the candidate subclusters and employed in the cost model to support creation of new profitable clusters. The cost model supporting clustering decisions such as creation of new clusters and removal of inefficient clusters is presented in the following section.

3.4 Cost model and benefit functions

As part of our clustering strategy, we rely on a cost model to support cluster restructuring operations like cluster splits and cluster merges. The cost model is used to evaluate the average search performance of existing clusters and to estimate it for future cluster candidates (candidate subclusters of existing clusters). The cost model embeds several system performance parameters affecting the query execution time, and also considers the spatial distribution of the data objects and of the query objects. The cost model and the benefit functions supporting our clustering strategy are presented next.

We first present the system parameters embedded in the cost model (Section 3.4.1). Then we define the performance indicators associated with all the clusters to gather statistics about the spatial distribution of the data objects and of the query objects (Section 3.4.2). We further provide a generic cost model that can be applied to different storage scenarios (Section 3.4.3). We finally derivate the two benefit functions supporting the clustering strategy (Section 3.4.4).

3.4.1 System performance parameters

The cost model takes into consideration a number of system parameters directly affecting the query execution time. These parameters depend on the performance characteristics of the microprocessor (CPU), of the main memory (RAM), and of the secondary memory (I/O). The system performance parameters considered in our cost model are:

- *memory access time* = the average time required to access a data object in main memory. This represents the cost of a random memory access to a known memory position.

- *memory object check rate / memory object check time* = the average number of data objects checked against the spatial selection criterion (e.g., intersection, containment) per time unit, considering that the data objects are sequentially placed in main memory / the average time required to check one data object against the spatial selection criterion in main memory. The access cost to the first data object is not counted.

- *disk seek/access time* = the average time spent to locate and access a data object in secondary memory. This represents the cost of a random disk access to a known disk position.

- *disk object transfer rate / disk object transfer time* = the average number of data objects transferred from secondary storage to main memory per time unit, considering that the data objects are sequentially placed on disk / the average time required to transfer one data object from secondary storage to main memory. The seek/access cost to the first data object is not counted.

3.4.2 Cluster performance indicators

To evaluate the cluster search performance, we associate each cluster with two statistics gathering useful information about the distribution of the data objects and of the query objects. These two statistics, further referred to as *cluster performance indicators*, are:

1. *Number of data objects (n_c)*

   - For a normal cluster we consider the number of data objects that are members of the cluster;
   - For a candidate subcluster, we count the number of data objects from the original cluster, matching the signature of the candidate subcluster.

   Combined with the system's performance parameters according to the storage scenario (i.e., memory object check time, disk object transfer time), this statistics allows one to estimate the *cluster exploration cost*. Indeed, cluster exploration implies object transfer from disk to memory when the data objects are stored on disk, and individual checking in memory of all the data objects that are members of the cluster.

2. *Number of visiting spatial queries (q_c)*

   - For a normal cluster we consider the number of spatial queries visiting the cluster over a period of time.

     By queries visiting the cluster we understand spatial queries that require the cluster exploration, namely spatial queries for which the cluster signature satisfies the spatial selection criteria with respect to the query objects;
   - For a candidate subcluster, we count the number of queries virtually visiting the cluster over a period of time.

     We consider that a query is virtually visiting a candidate cluster when the signature of the candidate cluster satisfies the spatial selection criterion with respect to the query object.

This statistics represents a good indicator for the *cluster access probability*. Indeed, the access probability of a cluster can be estimated as the ratio between the number of queries exploring the cluster and the total number of queries addressed to the system over a period of time:

$$p_c = \frac{q_c}{q_{system}}$$

where

- q_{system} represents the total number of spatial queries addressed to the system over a period of time;
- q_c represents the number of queries visiting the cluster c over the same period of time.

The cluster exploration cost together with the cluster access probability combined with the I/O and CPU performance parameters allows one to evaluate the search performance of a given cluster. The cost model is detailed in the next section.

3.4.3 Cost model and database storage scenarios

Our cost model is meant to evaluate the average query execution time associated with a cluster. The time cost induced by a spatial query at the exploration of a cluster is composed of:

- The cost of verifying the cluster signature

  The verification of the cluster signature is necessary to decide the cluster exploration. All the cluster signatures are stored in main memory, so this cost implies signature access and check in main memory.

- The cost of locating the storage space of the cluster in order to get access to the data objects

  This cost rises every time the cluster exploration is triggered. Depending on the storage scenario, this cost might embed a memory access or a disk seek/access operation.

- The cost of accessing and checking each data object against the spatial selection criterion

  This cost also rises every time the cluster exploration is triggered. This cost is proportional with the number of data objects that are members of the cluster and implies object transfer from disk to memory if the data objects are stored on disk, and verification of the spatial selection criterion against all the data objects that are members of the cluster.

- The cost of updating the query statistics associated with the cluster and with the candidate subclusters of the cluster

  Updating the query statistics is part of our clustering strategy. Its cost also rises every time the cluster exploration is triggered. The query statistics are maintained in memory for all the clusters and candidate subclusters.

The average spatial query execution time associated with a cluster c can be generally expressed as:

$$T_c = A + p_c \cdot (B + n_c \cdot C) \tag{3.1}$$

where

- n_c is the number of data objects from the cluster c

  n_c represents the first performance indicator maintained for the cluster c;

- p_c is the access probability associated with the cluster c

  p_c is estimated based on the number of visiting queries q_c, the second performance indicator maintained for the cluster c;

- A, B and C are three cost model parameters depending on the performance characteristics of the execution platform with respect to the chosen storage scenario

  For the cost model parameters A, B and C, we envisage two database storage scenarios. The first storage scenario considers that the data objects are stored in main memory. The second storage scenario considers that the data objects are stored on external support (disk). The two storage scenarios are examined next, and suitable definitions are provided for the three cost model parameters A, B and C.

  i. Main memory storage scenario

  With the recent proliferation of very large memory systems, motivated and supported by the decreasing price of RAM, a main memory database storage solution can be now envisaged. A main memory storage would significantly improve the query execution time because it enables the system to completely avoid expensive I/O data seek, access and transfer operations. Even if the database is entirely managed in main memory, when large quantities of data are involved, indexing methods are still necessary to accelerate the retrieval of the relevant data and to further improve the query execution. In particular, a main memory database indexing could be of special interest for applications where the query response time is important like in real-time information dissemination systems.

  In our case, for this first storage scenario, we consider that the size of the main memory is large enough to support the entire collection of spatial data objects, together with the additional data structures necessary to implement our clustering solution (i.e., the tree of cluster signatures and the performance indicators associated to clusters and to candidate subclusters). The data objects are arranged in clusters which are stored in the main memory. The data objects that are members of the same cluster are placed sequentially in order to maximize the data locality and to benefit from memory cache lines and read ahead capabilities of the modern processors, while performing cluster explorations.

  With respect to this storage scenario, the three cost model parameters A, B and C have the following signification:

  - A represents the *cluster signature verification time* = the time spent to check a cluster signature against the spatial selection criterion in order to decide the cluster exploration;

- B represents the *cluster access and query statistics time* = the time required to prepare the cluster exploration (call of the corresponding function, initialization of the object scan), plus the time spent to update the query statistics for the current cluster and for the candidate subclusters of the current cluster;
- C represents the *memory object check time* = the time required to check one object against the spatial selection criterion in order to decide the object qualification for the query answer;

ii. Disk storage scenario

For this second storage scenario we adopt a more classical approach where the data objects are stored in clusters on external support (on disk). The data objects that are members of the same cluster are placed sequentially on disk in order to minimize the disk head repositioning and to benefit from the the better performance of the sequential data transfer between disk and memory, while performing cluster explorations.

The additional data structures necessary to implement our clustering solution, namely the tree of cluster signatures, together with the data and the query statistics associated to clusters and to candidate subclusters, are still managed in main memory. Such a consideration can be afforded because on disk the number of clusters is less important than in main memory due to specific performance characteristics: expensive cluster seek/access and object transfer costs.

With respect to this storage scenario, the three cost model parameters A, B and C have the following signification:

- A represents the *cluster signature verification time* = the same as in the first scenario because all the cluster signatures are stored in main memory;
- B represents the *cluster access and query statistics time* = the time required to position the disk head at the beginning of the cluster in order to prepare the object read (disk seek/access time), plus, as in the main memory storage scenario, the time required to prepare the cluster exploration (call of the corresponding function, initialization of the object scan), and the time spent to update the query statistics for the current cluster and for the candidate subclusters of the current cluster;
- C represents the *object transfer and verification time* = the time required to transfer one object from disk to memory (disk object transfer time), plus the time required to check the data object against the spatial selection criterion in order to decide the object's qualification for the query answer (memory object check time – this second component is the same as in the memory storage scenario).

In the disk storage scenario, the cost components corresponding to I/O operations (disk seek/access time in B, and disk object transfer time in C) are predominant because they are much more expensive in terms of execution time.

3.4.4 Benefit functions supporting the clustering strategy

Equation (3.1) expresses the average spatial query execution time associated with a cluster, as a function of the system parameters A, B and C, and of the cluster performance indicators n_c and p_c. We derivate next cost-based expressions for the *materialization benefit function*, and for the *merge benefit function*.

Materialization benefit function

Each cluster is associated with a set of candidate subclusters potentially qualifying for materialization. The *materialization benefit function* β applies on each candidate subcluster and evaluates the performance gain expected from its possible materialization. For this purpose, β takes into consideration the performance indicators of the candidate subcluster, the performance indicators of the original cluster, and the set system parameters A, B and C affecting the query response time. Formally, if $\sigma_s \in \gamma(\sigma_c)$ (s is a candidate subcluster of the cluster c) then

$$\beta(s,c) \rightarrow \begin{cases} \geq \mathsf{min\_}\beta & \text{if the materialization of } s \text{ is profitable;} \\ < \mathsf{min\_}\beta & \text{otherwise.} \end{cases}$$

$\mathsf{min\_}\beta$ represents a threshold value corresponding to the minimum profit expected from the materialization of a candidate subcluster. This threshold value helps to avoid the creation of new clusters expected to bring very low performance gains because such clusters could rapidly turn into inefficient clusters.

To obtain the expression of the materialization benefit function β, we compare the query execution times before and after the hypothetical materialization of the candidate subcluster:

$$T_{bef} = T_c$$
$$T_{aft} = T_{c'} + T_s$$

- T_{bef} represents the execution time associated with the original cluster c;
- T_{aft} represents the joint execution time associated with the clusters c' and s resulted after the materialization of the candidate s of c.

The materialization operation is considered profitable if the query execution time resulting after performing this operation is lower than the query execution time before. The materialization benefit function is defined as

$$\beta(s,c) = T_{bef} - T_{aft} = T_c - (T_{c'} + T_s) \tag{3.2}$$

and represents the profit in terms of execution time, expected from the materialization of the candidate s of c.

Using Equation (3.1) to expand the three terms of Equation (3.2)

$$T_c = A + p_c \cdot (B + n_c \cdot C)$$
$$T_{c'} = A + p_{c'} \cdot (B + n_{c'} \cdot C)$$
$$T_s = A + p_s \cdot (B + n_s \cdot C)$$

and considering

i. $n_{c'} = n_c - n_s$

The sum of the numbers of objects in the resulted clusters c' and s is necessarily equal to the total number of objects in the original cluster c.

ii. $p_{c'} = p_c$

The access probability of c' remains p_c because c' takes the place and inherits the signature of c. The grouping characteristics of c, thus of c', do not change, neither the associated access probability.

then Equation (3.2) becomes:

$$\beta(s,c) = ((p_c - p_s) \cdot n_s \cdot C) - (p_s \cdot B) - A \tag{3.3}$$

Equation (3.3) gives the expression of the materialization benefit function. According to this equation, the interest in the materialization of a candidate subcluster is high when the candidate subcluster has an access probability lower than the access probability of the original cluster, and when enough objects from the original cluster qualify for the considered candidate subcluster to compensate the additional cluster access cost.

The usage of the materialization benefit function as decision support for split operations will be illustrated in the following chapter when presenting the cluster split procedure (Section 4.2.3).

Merge benefit function

The role of the *merge benefit function* μ is to evaluate the suitability of a merge operation between a cluster and his parent cluster. For this purpose, μ takes into consideration the performance indicators of the considered cluster, of the parent cluster, and the set of system parameters A, B and C affecting the query response time. Formally, if $\sigma_c \in \gamma(\sigma_a)$ (a is the parent cluster of the cluster c) then

$$\mu(c,a) \rightarrow \begin{cases} \geq \mathsf{min\_}\mu & \text{if the merge of } c \text{ to } a \text{ is profitable;} \\ < \mathsf{min\_}\mu & \text{otherwise.} \end{cases}$$

$\mathsf{min\_}\mu$ represents a threshold value corresponding to the minimum profit expected from the merge of a cluster to its parent cluster. This threshold value helps to avoid cluster merges that bring very low performance gains because such clusters could rapidly turn into profitable clusters.

To obtain the expression of the merge benefit function μ, we consider and compare the query execution times before and after the hypothetical merge operation:

$$T_{bef} = T_c + T_a$$

$$T_{aft} = T_{a'}$$

- T_{bef} represents the joint execution time associated with the original cluster c and to the parent cluster a

- T_{aft} represents the execution time associated with the cluster a' resulted after the merge of the clusters c and a.

The merge operation is considered profitable if the query execution time resulting after performing this operation is lower than the query execution time before. The merge benefit function is defined as

$$\mu(c, a) = T_{bef} - T_{aft} = (T_c + T_a) - T_{a'} \tag{3.4}$$

and represents the profit in terms of execution time, expected from the merge of the clusters c and a.

Using Equation (3.1) to expand the three terms of Equation (3.4)

$$T_c = A + p_c \cdot (B + n_c \cdot C)$$

$$T_a = A + p_a \cdot (B + n_a \cdot C)$$

$$T_{a'} = A + p_{a'} \cdot (B + n_{a'} \cdot C)$$

and considering

i. $n_{a'} = n_a + n_c$

The total number of objects in a' represents the sum of the numbers of objects in a and c.

ii. $p_{a'} = p_a$

The access probability of a' remains p_a because a' takes the place and inherits the signature of a. The grouping characteristics of a, thus of a', do not change, neither the associated access probability.

then Equation (3.4) becomes:

$$\mu(c, a) = A + (p_c \cdot B) - ((p_a - p_c) \cdot n_c \cdot C) \tag{3.5}$$

Equation (3.5) gives the expression of the merge benefit function. According to this equation, the interest in a merge operation is high when the access probability of the child cluster gets close to the access probability of the parent cluster (due to changes in the spatial distribution of the spatial queries), or when the number of objects in the child cluster decreases too much (due to data object removals).

The usage of the merge benefit function as decision support for merge operations will be illustrated in the following chapter when presenting the cluster merge invocation procedure (Section 4.2.1).

3.5 Conclusions

The clustering strategy was the first topic of this chapter. Our clustering strategy relies on a grouping criterion suitable for multidimensional objects with spatial extents and on a cost model embedding the performance characteristics of the execution platform. The grouping criterion is used to partition each cluster into a number of candidate subclusters representing future cluster candidates. Data and query statistics are maintained for the existing clusters and for the candidate subclusters. These statistics are used in

the cost model to evaluate the search performance of existing clusters and to estimate it for the candidate subclusters. The evaluation of the cluster search performance is required to support clustering decisions such as creation of new profitable clusters and removal of older inefficient cluster. Two types of cluster restructuring operations are used to accomplish the object clustering: cluster splits and cluster merges. A cluster split is achieved by materializing a number of candidate subclusters of a given cluster. New clusters are only created (materialized) if they are expected to improve the average query performance. The cluster split decisions are assisted by the materialization benefit function based on the cost model. A cluster merge is performed when the profitability of the considered cluster has decreased as result of changes in the data or in the query distribution. The inefficient cluster is withdrawn from the spatial database and its objects are transferred to the direct ancestor from the clustering hierarchy. The cluster merge decisions are assisted by the merge benefit function also based on the cost model.

The object grouping criterion was the second topic of this chapter. We first defined a generic grouping model suitable for multidimensional objects with spatial extents. This grouping model is based on the notion of similar objects denoting spatial objects with intervals of comparable sizes and located in the same domain regions over dimensions. The cluster signature was defined to precise the notion of object similarity. The cluster signature consists of two intervals per dimension, called intervals of variation. The intervals of variation represent domain regions where the starts and the ends of the intervals of the qualifying spatial objects are allowed to fall in each dimension. According to the clustering strategy, each existing cluster needs to be partitioned into a number of candidate subclusters. The different ways in which a cluster is partitioned into several candidate subclusters are determined by the clustering function. So the role of the clustering function is to generate the signatures of the candidate subclusters. To implement the clustering function, a grouping method is required, able to generate the possible subsignatures of a cluster signature. We examined two possible grouping methods. The first method is based on a space division into cells of equal volume, using all the dimensions at once. This grouping method proved to be impractical because the number of possible subsignatures is exponential with the number of dimensions, which implies high maintenance costs. The second method is based on a space division into slices of equal volume, using one dimension at a time. The number of possible subsignatures generated by this grouping method is linear with the number of dimensions, making possible the statistics maintenance. For this reason, the second grouping method was retained and used to implement the clustering function. We explained the implementation of the clustering function and emphasized its properties: (1) Given a cluster signature, the clustering function ensures a number of cluster subsignatures, which is linear with the number of dimensions; (2) The number of subsignatures matched by a spatial object satisfying the cluster signature is equal to the number of dimensions; and (3) The clustering function enables an object grouping based on a reduced subset of dimensions and domain regions, namely the most selective and discriminatory with respect to the cost model.

The cost model was the last topic of this chapter. We presented the system parameters embedded in the cost model, as well as the performance indicators associated with clusters and with candidate subclusters to maintain data and query statistics (number of qualifying data objects and number of visiting queries). The average spatial query time associated with a cluster was expressed as a function of the cluster performance indicators and of three generic hardware-depended parameters A, B, and C. The three

parameters A, B, and C are defined according to the storage scenario adopted for the spatial database (i.e., main memory storage or disk-based storage). Based on the generic expression of the average query time associated with a cluster, we derived computing formulas for the materialization benefit function and for the merge benefit function. To obtain them we considered the balance between the average query time before and after performing the corresponding operation. The cost model is intended to ensure that materializations of candidate subclusters and cluster merges are only performed if they are expected to improve the average performance of spatial queries.

In this chapter we presented the main elements of our clustering method: the clustering strategy, the object grouping criterion and the cost model supporting the clustering strategy. In the next chapter, we provide algorithms and execution procedures for database clustering operations and for standard database manipulation operations, designed to ensure the implementation of the clustering strategy.

Chapter 4

Database clustering and manipulation algorithms

Our clustering approach is designed to: (i.) partition the collection of multidimensional spatial objects into clusters such as to improve the average performance of spatial queries; (ii.) ensure fast data object update operations (insertions and deletions); (iii.) dynamically adapt the object clustering to important changes that might occur in data or query distributions in order to avoid significant performance degradation. In this chapter we provide algorithms for database clustering operations such as cluster restructuring invocation, cluster split and cluster merge, and for standard database manipulation operations like spatial query execution, data object insertion and data object deletion.

Chapter organization In Section 4.1 we introduce a number of notations used throughout the presented algorithms. In Section 4.2 we discuss and present the execution procedures corresponding to the object grouping operations used to accomplish the database clustering: cluster restructuring invocation, cluster split, and cluster merge. In Section 4.3 we provide the algorithms corresponding to standard database manipulation operations: spatial query execution, data object insertion and data object deletion.

4.1 Notations

The following notations will be used throughout the algorithms presented next:

- $\mathcal{C}$ represents the set of (materialized) clusters from the spatial database;
- $\forall \mathsf{c} \in \mathcal{C}$
  - $\sigma(\mathsf{c})$ represents the signature of the cluster c;
  - *objects*(c) represents the set of data objects that are members of the cluster c;
  - *parent*(c) represents the parent cluster of the cluster c (direct ancestor in the clustering hierarchy);
  - *children*(c) represents the set of child clusters of the cluster c (clusters resulting from splits of the cluster c);

- $candidates(\mathsf{c})$ represents the set of candidate subclusters of the cluster c;
- $n(\mathsf{c})$ represents the number of data objects belonging to the cluster c;
- $q(\mathsf{c})$ represents the number of spatial queries visiting the cluster c since the cluster's creation or since the last initialization of this parameter;
- $p(\mathsf{c})$ represents the access probability associated with the cluster c;

- $\forall \mathsf{c} \in \mathcal{C}$ and $\forall s \in candidates(\mathsf{c})$
  - $\sigma(\mathsf{s})$ represents the signature of the candidate subcluster s of the cluster c;
  - $n(\mathsf{s})$ represents the number of data objects from the cluster c, qualifying for the candidate subcluster s;
  - $q(\mathsf{s})$ represents the number of spatial queries virtually accessing the candidate subcluster s since the creation of the cluster c or since the last initialization of this parameter;
  - $p(\mathsf{s})$ represents the access probability associated with the candidate subcluster s;

- Operational functions
  - $\gamma()$ represents the clustering function;
  - $\beta()$ represents the materialization benefit function;
  - $\mu()$ represents the merge benefit function;

- Other notations
  - $\mathsf{root} \in \mathcal{C}$ represents the root cluster;
  - $q(\mathsf{system}) = q(\mathsf{root})$ represents the total number of spatial queries addressed to the database system (all the spatial queries are visiting the root cluster);
  - $\mathsf{period\_q}$ represents the period in number of spatial queries (addressed to the database system) at which cluster restructuring operations are considered;
  - $\mathsf{min\_q}$ represents the minimum number of spatial queries that should visit a cluster before considering a cluster restructuring decision;
  - $\mathsf{min\_}\beta$ represents the minimum profit expected from the materialization of a candidate subcluster;
  - $\mathsf{min\_}\mu$ represents the minimum profit expected from a merge between a cluster and its parent cluster;

Table 4.1 summarizes the main notations used throughout the presented algorithms.

4.2 Algorithms for database clustering operations

The data objects from the spatial database are initially stored in the root cluster. By means of iterative cluster splits, groups of objects are gradually extracted from the existing clusters (initially from the root cluster) and relocated into new clusters. The cost model is used to assist the cluster splits, ensuring that new clusters are only created (materialized) when they are expected to be profitable. On the other hand, older clusters

Table 4.1: Notations

$\mathcal{C}$	set of database clusters
$\sigma(\mathsf{c})$	signature of cluster c
$objects(\mathsf{c})$	set of data objects from cluster c
$parent(\mathsf{c})$	parent cluster of cluster c
$children(\mathsf{c})$	set of child clusters of cluster c
$candidates(\mathsf{c})$	set of candidate subclusters of cluster c
$n(\mathsf{c})$	number of objects in cluster c
$q(\mathsf{c})$	number of queries visiting cluster c
$p(\mathsf{c})$	access probability of the cluster c
$\gamma()$	clustering function
$\beta()$	materialization benefit function
$\mu()$	merge benefit function

having lost their profitability are withdrawn from the spatial database through merge operations (the corresponding objects are transferred back to the parent clusters). The order in which the cluster restructuring operations are invoked for the existing clusters is important for the clustering process. We first present and discuss the invocation of the cluster restructuring operations (4.2.1). Then we present and detail the cluster merge procedure (4.2.2) and the cluster split procedure (4.2.3).

4.2.1 Cluster restructuring invocation

Cluster splits and cluster merges need to be periodically considered for all the clusters in order to accomplish and refine the object clustering and to adapt it to data and query distribution changes. In our case, the cluster restructuring operations are invoked on a periodical basis determined by the occurrence of a fixed number of spatial queries. This enables the system to gather sufficient statistics on the spatial distribution of the query objects in order to properly support the cluster restructuring decisions.

Figure 4.1 illustrates the procedure used to invoke the cluster restructuring operations. Every period_q spatial queries (condition from Step 1), two global cluster re-

```
RestructureClusters()
1.   if ( q(system) % period_q = 0 ) then
2.       RecursiveMergeClusters(root);
3.       RecursiveSplitClusters(root);
End.
```

Figure 4.1: Cluster restructuring invocation

structuring actions are triggered: First, merge operations are considered for the existing database clusters (Step 2). Then split operations are considered for the database clusters (Step 3). In the condition from Step 1, period_q denotes the period in number of

spatial queries at which the cluster restructuring operations are considered, $q(\mathsf{system})$ represents the total number of spatial queries addressed to the database system, while % represents the "modulo" operator. The procedures corresponding to the two global restructuring actions, *RecursiveMergeClusters* and *RecursiveSplitClusters*, apply recursively to the clusters from the clustering hierarchy, starting with the root. The two recursive procedures are presented next.

Recursive cluster merges According to the algorithm from Figure 4.2, the task of the *RecursiveMergeClusters*-procedure is to consider and, when profitable, perform all the merges between inefficient children of the current cluster and this cluster (Steps 1-5). Then the *RecursiveMergeClusters*-procedure applies in a recursive manner to each child

RecursiveMergeClusters($\mathsf{c} \in \mathcal{C}$)

1. **if** $q(\mathsf{c}) \geq \mathsf{min\_q}$ **then**
2. **for each** s **in** *children*(c) **do**
3. **if** $q(\mathsf{s}) \geq \mathsf{min\_q}$ **then**
4. **if** $\mu(\mathsf{s}, \mathsf{c}) \geq \mathsf{min\_}\mu$ **then**
5. ClusterMerge(s, c);
6. **for each** s **in** *children*(c) **do**
7. RecursiveMergeClusters(s);

End.

Figure 4.2: Recursive cluster merges

cluster of the current cluster (Steps 6-7). Since the *RecursiveMergeClusters*-procedure is initially invoked for the root cluster, the suitability of the merge operation is examined for all parent/child pairs of clusters from the clustering hierarchy.

A merge decision can only be taken when enough query statistics are available for both parent and child clusters. The query statistics are necessary to estimate the access probabilities associated with the two clusters involved in the merge operation. The cluster access probabilities are used in the cost model to evaluate the search performance of the two clusters in order to determine the profitability of the merge operation. The condition regarding the minimum number of query statistics is verified in Step 1 for the parent cluster and in Step 3 for the child cluster: $q(\mathsf{c})$ represents the number of spatial queries that have visited the cluster c, $q(\mathsf{s})$ represents the number of spatial queries that have visited the child cluster s, and $\mathsf{min\_q}$ represents the minimum number of spatial queries required to properly estimate the access probability of a cluster. If the minimum number of spatial queries is not reached for both, parent and child, clusters, the merge operation is not considered for the corresponding pair of clusters.

When enough query statistics are available to support the merge decision, the merge

benefit function $\mu(\mathsf{s}, \mathsf{c})$ is invoked to compute the profit expected from merging the child cluster s to the current cluster c (Step 4). If the expected profit is superior to $\mathsf{min\_}\mu$, the merge operation can be performed and the cluster merge procedure, $ClusterMerge$, is executed in Step 5. The threshold value $\mathsf{min\_}\mu$ represents the minimum expected profit required to perform a merge operation. When the expected merge profit does not exceed the $\mathsf{min\_}\mu$ value, the child cluster is considered still profitable and therefore preserved. The expression of the merge benefit function can be found in Section 3.4.4. The $ClusterMerge$-procedure invoked in Step 5 to perform the actual merge of a cluster to its parent cluster is detailed in Section 4.2.2.

When a child cluster s is merged to the current cluster c, s is withdrawn from the database and also from the set of children of c. Since the cluster s is removed from the database, the current cluster c becomes the new parent of the children of s. So the children of s are added to the set of children of c. As illustrated in Step 2, the consideration of the merge operation applies to all the children of the current cluster. This also includes the new children acquired as results of subsequent cluster merges. As a result, all the clusters from the clustering hierarchy are tested for the merge operation.

Recursive cluster splits The $RecursiveSplitClusters$ procedure is illustrated in Figure 4.3. The task of this procedure is to consider and, when profitable, perform the split of the current cluster, c, given as argument (Steps 2-3). Then the $RecursiveSplitClusters$-

RecursiveSplitClusters($\mathsf{c} \in \mathcal{C}$)

1. **let** $\mathcal{S} \leftarrow children(\mathsf{c})$;
2. **if** $q(\mathsf{c}) \geq \mathsf{min\_q}$ **then**
3. ClusterSplit(c);
4. **for each** s **in** $\mathcal{S}$ **do**
5. RecursiveSplitClusters(s);

End.

Figure 4.3: Recursive cluster splits

procedure applies in a recursive manner to all the clusters from $\mathcal{S}$, where $\mathcal{S}$ represents the initial set of children of the cluster c (Steps 4-5). The initial set of children of the cluster c needs to be recorded (Step 1) because afterwards it grows when the cluster c gets split. However, the new acquired children of the cluster c should not be considered for splits because they do not have any query statistics associated yet. Since the $RecursiveSplitClusters$-procedure is initially invoked for the root cluster, all the clusters from the clustering hierarchy are considered and tested for the split operation.

A split decision can only be taken when enough query statistics are associated with the corresponding cluster. The query statistics are necessary to estimate the access probabilities associated with the cluster considered for the split operation, and with the candidate subclusters of this cluster. These access probabilities are further used to evaluate the

profits expected from the possible materializations of the candidate subclusters of this cluster. The condition regarding the minimum number of query statistics is verified in Step 2: $q(\mathsf{c})$ represents the number of spatial queries that have visited the cluster c, and $\mathsf{min\_q}$ represents the minimum number of spatial queries required to support the split decision. If the minimum number of spatial queries is not reached, the split operation is not considered for the current cluster.

When enough query statistics are available to support the split decision, the split procedure called *ClusterSplit* is invoked for the current cluster in Step 3. This procedure will attempt to split the current cluster by materializing some of its profitable candidate subclusters. The *ClusterSplit*-procedure is presented in Section 4.2.3.

Some considerations As illustrated in Figure 4.1, we first consider all possible cluster merges, then we consider all possible cluster splits. This invocation order is meant to enable alternative splits for the clusters having acquired new data objects from unprofitable children as result of merges. Splitting an inefficient cluster is not interesting because the remaining cluster would continue to be unprofitable (its access probability would not change). Rather than splitting an inefficient cluster, we first merge it to its parent cluster, and later attempt to split the parent cluster. As the merge operations are performed before the split operations, several child clusters could be merged to the same parent before considering the split of this last one. This enables our restructuring method to regroup several clusters before performing a split, which can lead to better clustering configurations.

The merge and the split operations are considered for all the existing database clusters every $\mathsf{period\_q}$ spatial queries. However, in practice only a few restructuring operations are actually triggered, namely those involving clusters with enough query statistics, and whose expected profitability exceeds the minimum profit threshold.

By the usage of the $\mathsf{min\_q}$ parameter, the clusters frequently accessed are more often examined and considered for restructuring operations. This behavior is intended because frequently accessed clusters contribute more to the average execution performance of the spatial queries.

4.2.2 Cluster merge procedure

The cluster merge procedure is invoked when the merge between a cluster and its parent cluster is considered as profitable for the average spatial query performance. The data objects of the child cluster are first transferred back to the parent cluster, then the child cluster is withdrawn from the spatial database.

Figure 4.4 illustrates the actions performed during the cluster merge: The data objects from the child cluster are relocated to the parent cluster in Step 1. This object relocation requires the actualization of the data statistics associated with the parent cluster: The number of objects in the parent cluster is updated in Step 2, as well as the numbers of qualifying objects associated with the candidate subclusters of the parent cluster (Steps 3-5). To preserve the clustering hierarchy, the parent cluster becomes the parent of the children of the child cluster (Steps 6-7), and the list of children of the parent cluster is accordingly updated in Steps 8-9. Finally, the child cluster is removed from the spatial database (Step 10).

ClusterMerge ($\mathsf{c} \in \mathcal{C}$, $\mathsf{a} \in \mathcal{C} \mid \mathsf{a} \leftarrow \mathit{parent}(\mathsf{c})$)

```
//    Move data objects from child cluster c to parent cluster a:
1.    let objects(a) ← objects(a) ∪ objects(c);

//    Update data statistics for parent cluster a:
2.    let n(a) ← n(a) + n(c);

//    Update data statistics for candidate subclusters of parent cluster a:
3.    for each s in candidates(a) do
4.        let M(s, c) ← {o ∈ objects(c) | o matches σ(s)};
5.        let n(s) ← n(s) + card(M(s, c));

//    Set parent reference for child clusters of c:
6.    for each s in children(c) do
7.        let parent(s) ← a;

//    Update list of child references for parent cluster a:
8.    let children(a) ← children(a) ∪ children(c);
9.    let children(a) ← children(a) \ {c};

//    Remove c from database:
10.   let C ← C \ {c};
```

End.

Figure 4.4: Cluster merge procedure

Complexity of the cluster merge procedure The complexity of the merge procedure is dominated by the cost of (i.) relocating the data objects from the child cluster to the parent cluster (Step 1), and by the cost of (ii.) updating the data object statistics for the candidate subclusters of the parent cluster (Step 3-5). The other actions involve very simple operations whose costs are negligible compared to the first two actions. Next we further detail the data object relocation and the data object statistics updating.

Data object relocation. For performance reasons, our clustering strategy requires that the storage spaces of the clusters be contiguous. The data objects belonging to the same cluster have to be placed together. When relocating the data objects from the child cluster to the parent cluster, the storage space allocated for the parent cluster might not be large enough to accommodate all the objects acquired from the child cluster. To handle this problem in a simple manner, we systematically allocate a new storage space for the parent cluster and transfer all the data objects from the child cluster and from the parent cluster to the new storage location. The old storage space of the parent cluster is then liberated, together with the storage space of the child cluster.

Accordingly, the data object relocation requires one read and one write for each object from the two clusters involved (parent and child). Since the data objects are sequentially placed, their transfer can be performed in an efficient manner by means of buffered reads and writes. The buffered object transfer is very important for the execution performance, notably in the case of a disk-based cluster storage, where it can help to avoid numerous expensive I/O seek/access operations, saving a lot of execution time. Globally, the cost

of the data object relocation is determined by the sizes of the two involved clusters in terms of number of buffered read/write operations. Depending on the clusters and buffer sizes, this cost should not significantly exceed twice the cost of a buffered read of the two clusters. Read of the two clusters together is very likely to occur during spatial selections because a cluster that is subject for a merge has an access probability close to the access probability of the parent cluster.

Data object statistics updating. The task of updating the data object statistics associated with the candidate subclusters of the parent cluster is accomplished when the objects from the child cluster are transferred to the new storage location. This task consists of incrementing the numbers of objects of the candidate subclusters that could accommodate an object from the child cluster. The cost of this operation is proportional to the number of objects in the child cluster multiplied by the number of relevant candidate subclusters. The number of relevant candidate subclusters is given by the clustering function and is equal to N_d, where N_d represents the number of dimensions (see Section 3.3.5).

4.2.3 Cluster split procedure

The cluster split procedure attempts to split a database cluster by materializing some of its candidate subclusters. Only the candidate subclusters whose materializations are expected to be profitable are turned into real clusters. The cluster split procedure is illustrated in Figure 4.5. The candidate subclusters promising the best materialization profits are first selected in Step 1: $\mathcal{B}$ represents the set of the most profitable candidate subclusters. To obtain the $\mathcal{B}$-set, the materialization benefit function β computes the materialization profits for all the candidate subclusters, and the most profitable candidates are retained. In order to consider a candidate subcluster, the profit expected from its materialization must exceed the minimum acceptable value $\mathsf{min\_}\beta$. The expression of the materialization benefit function can be found in Section 3.4.4.

If the $\mathcal{B}$-set contains candidates subclusters (Step 2), then one of its members becomes subject for materialization (Step 3). The materialization process consists of the following actions: A new database cluster is created and added to the spatial database in Step 4. The objects qualifying for the selected candidate are identified in Step 5 and moved from the original cluster to the new cluster (Steps 6-7). The configuration of the new cluster is set in Steps 8 (signature), 9 (parent cluster), and 10 (number of member data objects). Steps 11-12 initialize the data object statistics associated with the candidate subclusters of the new cluster. The number of objects remaining in the original cluster is accordingly updated in Step 13. The numbers of qualifying objects are also updated for the candidate subclusters of the original cluster in Steps 14-16. Steps 14-16 are necessary because the data objects qualifying for the selected candidate subcluster also count for other candidate subclusters. Object qualification for multiple candidate subclusters is enabled by the clustering function because the candidate subclusters are just virtual clusters. However, once relocated from the original cluster to the new materialized cluster, the corresponding data objects can no more count for the candidate subclusters of the original cluster.

When the materialization of a candidate subclusters is finished, the split procedure continues with the selection of the next best candidate subcluster. Thus the materialization process repeats from Step 1 until no profitable candidate subcluster is found. The

ClusterSplit ($\mathsf{c} \in \mathcal{C}$)

// Find best candidate subclusters for materialization:
1. **let** $\mathcal{B} \leftarrow \{\mathsf{b} \in \mathit{candidates}(\mathsf{c}) \mid \beta(\mathsf{b},\mathsf{c}) > \mathsf{min\_}\beta \wedge \beta(\mathsf{b},\mathsf{c}) \geq \beta(\mathsf{d},\mathsf{c}),\ \forall \mathsf{d} \neq \mathsf{b} \in \mathit{candidates}(\mathsf{c})\}$;

2. **if** ($\mathcal{B} \neq \emptyset$) **then**

// One of best candidate subclusters is materialized:
3. **let** $\mathsf{b} \in \mathcal{B}$;

// Create new database cluster d;
4. **let** $\mathsf{C} \leftarrow \mathsf{C} \cup \{\mathsf{d}\}$;

// Move qualifying data objects from cluster c to new cluster d;
5. **let** $\mathcal{M}(\mathsf{b},\mathsf{c}) \leftarrow \{\mathsf{o} \in \mathit{objects}(\mathsf{c}) \mid \mathsf{o} \text{ matches } \sigma(\mathsf{b})\}$;
6. **let** $\mathit{objects}(\mathsf{d}) \leftarrow \mathcal{M}(\mathsf{b},\mathsf{c})$;
7. **let** $\mathit{objects}(\mathsf{c}) \leftarrow \mathit{objects}(\mathsf{c}) \setminus \mathcal{M}(\mathsf{b},\mathsf{c})$;

// Set configuration for new cluster d:
8. **let** $\sigma(\mathsf{d}) \leftarrow \sigma(\mathsf{b})$;
9. **let** $\mathit{parent}(\mathsf{d}) \leftarrow \mathsf{c}$;
10. **let** $n(\mathsf{d}) \leftarrow n(\mathsf{b})$;

// Set data object statistics for candidate subclusters of cluster d:
11. **for each** s **in** $\mathit{candidates}(d)$ **do**
12. **let** $\mathcal{M}(\mathsf{s},\mathsf{d}) \leftarrow \{\mathsf{o} \in \mathit{objects}(\mathsf{d}) \mid \mathsf{o} \text{ matches } \sigma(\mathsf{s})\}$;

// Update data object statistics for cluster c:
13. **let** $n(\mathsf{c}) \leftarrow n(\mathsf{c}) - n(\mathsf{d})$;

// Update data object statistics for candidate subclusters of cluster c:
14. **for each** s **in** $\mathit{candidates}(\mathsf{c})$ **do**
15. **let** $\mathcal{M}(\mathsf{s},\mathsf{d}) \leftarrow \{\mathsf{o} \in \mathit{objects}(\mathsf{d}) \mid \mathsf{o} \text{ matches } \sigma(\mathsf{s})\}$;
16. **let** $n(\mathsf{s}) \leftarrow n(\mathsf{s}) - \mathit{card}(\mathcal{M}(\mathsf{s},\mathsf{d}))$;
17. **let** $n(\mathsf{s}) \leftarrow \mathit{card}(\mathcal{M}(\mathsf{s},\mathsf{d}))$;

// Consider next candidate subcluster for materialization:
18. **go to** 1.

// If the cluster c was split, then reset associated query statistics:
19. **if** c *was split* **then**
20. **let** $q(\mathsf{c}) \leftarrow 0$;
21. **for each** s **in** $\mathit{candidates}(\mathsf{c})$ **do**
22. **let** $q(\mathsf{s}) \leftarrow 0$;

End.

Figure 4.5: Cluster split procedure

selection for materialization of the candidate subclusters is performed in a greedy manner and the most profitable candidates are materialized first. In order to take into consider-

ation the data changes induced in the original cluster and in the candidate subclusters by subsequent materializations, the $\mathcal{B}$-set of the best candidates needs to be recomputed each time (Step 1). At the end, the original cluster will only host the objects qualifying for none of the new materialized subclusters. Of course, if no candidate subcluster is expected to be profitable ($\mathcal{B} = \emptyset$), the original cluster remains unchanged.

At the end of the split procedure, if at least one candidate subcluster of the original cluster was materialized, we reset the query statistics associated with the original cluster and with the candidate subclusters of the original cluster (Steps 19-22). The purpose of this action is to enable the system to gather new query statistics at cluster level, further used to dynamically adapt the object grouping to changes in the spatial distribution of the query objects that might occur over time.

Complexity of the cluster split procedure A cluster split may consist of several successive object relocations from the original cluster to different new materialized candidate subclusters. Each time a candidate subcluster is materialized, the qualifying data objects need to be relocated from the original cluster to the new cluster. Thanks to the data statistics associated with the candidate subclusters, we know exactly the number of objects qualifying for the new materialized cluster. To perform the object relocation, we allocate two new storage spaces: one to receive the data objects of the new materialized candidate subcluster, and one to receive the data objects remaining in the original cluster. During the object relocation, each data object from the original cluster is read, checked against the signature of the candidate subcluster, and written either to the storage space corresponding to the new materialized candidate subcluster, or to the new storage space of the original cluster. As required, the objects are sequentially placed in the new storage locations. At the end of the object relocation, the old storage space of the original cluster is liberated and its place is taken by the new corresponding storage space.

Data object relocation. The cost of the data object transfer is proportional to the number of objects from the original cluster, in terms of number of read/check/write operations. For efficiency reasons, the read/write operations are buffered, which is important for the execution performance, notably when adopting a disk-based cluster storage.

Data object statistics updating. Together with the object transfer, we also need to actualize the data statistics indicators: the numbers of data objects for the two clusters (the original cluster and the new materialized cluster) and for the candidate subclusters of the two clusters. The updating of the data statistics indicators is performed during the object relocation: As soon as we find a data object qualifying for the new materialized candidate subcluster, we decrement the numbers of objects associated with the relevant candidate subclusters of the original cluster, and increment the corresponding statistics associated with the relevant candidate subclusters of the new materialized cluster. The cost of this task is proportional to the number of objects qualifying for the materialized candidate subcluster multiplied by the number of relevant candidate subclusters. This last is given by the clustering function and is equal to the number of dimensions N_d (see Section 3.3.5).

4.3 Algorithms for database manipulation operations

In this section we introduce the execution algorithms corresponding to standard database manipulation operations such as spatial query execution (4.3.1) and database update operations like data object insertion (4.3.2) and data object deletion (4.3.3).

4.3.1 Spatial query execution

A spatial range query specifies a spatial object representing the query object, ρ, and a spatial selection criterion, ∇, requested between the query object and the data objects forming the query answer. We first define the spatial selection criteria corresponding to different types of spatial range queries that we support, then we introduce the spatial query execution procedure and discuss its complexity.

Spatial selection criteria The spatial selection criterion, denoted by ∇, depends on the type of the spatial range query that we want to execute. Four types of queries are of interest in our case: intersection, containment, enclosure and similar-shape queries. To define the corresponding spatial selection criteria we will use the following notations:

$\rho = \{ ([a_{\rho_i}, b_{\rho_i}]) \mid i \in \{1, 2, ..., N_d\} \}$ – ρ represents a query object with its intervals $[a_{\rho_i}, b_{\rho_i}]$ in the N_d dimensions ($i \in \{1, 2, ..., N_d\}$)

$o = \{ ([a_{o_i}, b_{o_i}]) \mid i \in \{1, 2, ..., N_d\} \}$ – o represents a data object with its intervals $[a_{o_i}, b_{o_i}]$ in the N_d dimensions ($i \in \{1, 2, ..., N_d\}$)

$\sigma = \{ ([a^{min}_{\sigma_i}, a^{max}_{\sigma_i}] : [b^{min}_{\sigma_i}, b^{max}_{\sigma_i}]) \mid i \in \{1, 2, ..., N_d\} \}$ – σ represents a cluster signature with its intervals of variation $[a^{min}_{\sigma_i}, a^{max}_{\sigma_i}] : [b^{min}_{\sigma_i}, b^{max}_{\sigma_i}]$ in the N_d dimensions ($i \in \{1, 2, ..., N_d\}$)

The notations $(\rho \nabla o)$ and $(\rho \nabla \sigma)$ are used in the following algorithms to decide if the data object o, respectively the cluster signature σ, is satisfying the spatial selection criterion ∇ with respect to the query object ρ. We define now the semantics of $(\rho \nabla o)$ and $(\rho \nabla \sigma)$ for each type of spatial range query:

- *Intersection Query:* Find all the data objects intersecting the query object

  $$(\rho \nabla o) \equiv \{(a_{\rho_i} \leq b_{o_i}) \wedge (a_{o_i} \leq b_{\rho_i}) \mid \forall i \in \{1, 2, ..., N_d\} \}$$

  $$(\rho \nabla \sigma) \equiv \{(a_{\rho_i} \leq b^{max}_{\sigma_i}) \wedge (a^{min}_{\sigma_i} \leq b_{\rho_i}) \mid \forall i \in \{1, 2, ..., N_d\} \}$$

- *Containment Query:* Find all the data objects enclosed by the query object

  $$(\rho \nabla o) \equiv \{(a_{\rho_i} \leq a_{o_i}) \wedge (b_{o_i} \leq b_{\rho_i}) \mid \forall i \in \{1, 2, ..., N_d\} \}$$

  $$(\rho \nabla \sigma) \equiv \{(a_{\rho_i} \leq a^{max}_{\sigma_i}) \wedge (b^{min}_{\sigma_i} \leq b_{\rho_i}) \mid \forall i \in \{1, 2, ..., N_d\} \}$$

- *Enclosure Query:* Find all the data objects enclosing the query object

$$(\rho \nabla o) \equiv \{(a_{o_i} \leq a_{\rho_i}) \wedge (b_{\rho_i} \leq b_{o_i}) \mid \forall i \in \{1, 2, ..., N_d\}\ \}$$

$$(\rho \nabla \sigma) \equiv \{(a^{min}_{\sigma_i} \leq a_{\rho_i}) \wedge (b_{\rho_i} \leq b^{max}_{\sigma_i}) \mid \forall i \in \{1, 2, ..., N_d\}\ \}$$

- *Similar-shape Query:* Find all the data objects with similar locations and extensions with those of the query object according to some maximum acceptable variation values

$$(\rho \nabla o) \equiv \{(a_{\rho_i} - \epsilon_i \leq a_{o_i} \leq a_{\rho_i} + \epsilon_i) \wedge (b_{\rho_i} - \epsilon_i \leq b_{o_i} \leq b_{\rho_i} + \epsilon_i) \mid \forall i \in \{1, 2, ..., N_d\}\ \}$$

$$(\rho \nabla \sigma) \equiv \{(a_{\rho_i} - \epsilon_i \leq a^{max}_{\sigma_i}) \wedge (a^{min}_{\sigma_i} \leq a_{\rho_i} + \epsilon_i) \wedge (b_{\rho_i} - \epsilon_i \leq b^{max}_{\sigma_i}) \wedge (b^{min}_{\sigma_i} \leq b_{\rho_i} + \epsilon_i) \mid \forall i \in \{1, 2, ..., N_d\}\ \}$$

  where ϵ_i represents the maximum acceptable variation value with respect to the interval bounds of the query object in dimension i ($i \in \{1, 2, ..., N_d\}$).

Spatial query execution procedure Answering a spatial query implies the exploration of all the database clusters whose signatures satisfy the spatial selection criterion with respect to the query object. The spatial query execution algorithm is illustrated in

SpatialQuery (query object ρ, spatial selection criterion ∇) : data object set

```
//    Initialize the query answer set:
1.    let R ← ∅;

//    Determine the clusters to be explored:
2.    let X ←ClustersToExplore(root, ρ, ∇);

//    Exploration of qualifying clusters:
3.    for each cluster c ∈ X do

//        Check all data objects against the selection criterion:
4.        for each object o in objects(c) do
5.            if (ρ ∇ o) then
6.                let R ← R ∪ {o};

//        Update query statistics for cluster c:
7.        let q(c) ← q(c) + 1;

//        Update query statistics for candidate subclusters of cluster c:
8.        let S ← {s ∈ candidates(c) | ρ ∇ σ(s)};
9.        for each s in S do
10.           let q(s) ← q(s) + 1;

//    Return the query result:
11.   return R;
```

End.

Figure 4.6: Spatial query execution algorithm

Figure 4.6. The set of clusters requiring exploration is computed in Step 2 by invoking the recursive procedure *ClustersToExplore* on the root cluster (*ClustersToExplore* procedure is presented in Figure 4.7). The data objects belonging to the clusters qualifying for exploration are individually checked against the spatial selection criterion (Steps 4-6). As indicator for the access probability, the number of exploring queries is incremented for each explored cluster, as well as for the corresponding candidate subclusters virtually explored (Steps 7-10).

The *ClustersToExplore* procedure from Figure 4.7 constructs the set of clusters requiring exploration by verifying in a recursive manner the cluster signatures against the spatial selection criterion. We note that the children of the clusters whose signatures do not satisfy the spatial selection criterion ∇ are not considered for exploration.

ClustersToExplore($\mathsf{c} \in \mathcal{C}$, query object ρ, spatial selection criterion ∇) : cluster set

// Check the cluster signature against the spatial selection criterion:
1. **if** ($\rho \nabla \sigma(\mathsf{c})$) **then**

// Recursively construct and return the set of clusters to be explored:
2. **let** $\mathcal{X} \leftarrow \{\mathsf{c}\}$

3. **for each** s **in** *children*(c) **do**
4. **let** $\mathcal{X} \leftarrow \mathcal{X} \cup$ ClustersToExplore(s, ρ, ∇);

5. **return** $\mathcal{X}$;

6. **else**

7. **return** $\emptyset$;

End.

Figure 4.7: Recursively determine the clusters to be explored

Complexity of the spatial query execution The complexity of the spatial query execution is given by: (i.) checking the cluster signatures to determine the clusters requiring exploration, (ii.) individually checking the data objects from the qualifying clusters, and (iii.) updating the query statistics for the explored clusters/candidate subclusters. The cost of the first task is in the worst case proportional to the total number of clusters in terms of signature checks. However, in practice, the hierarchy of clusters helps to avoid checking all the signatures. Indeed, the clusters descending from a cluster whose signature does not require cluster exploration do not require exploration either. The higher the query selectivity, the smaller the number of explored clusters, and thus the cost of the first task. The cost of the second task is proportional to the number of explored clusters in terms of data access/seek operations, and to the number of member objects in terms of data reads/verifications. When the clusters are stored on secondary memory, expensive I/O operations are occurring like disk seek/access and object transfer from disk to memory. The third task consists of incrementing the numbers of visiting queries associated with the explored clusters and with their corresponding candidates

subclusters whose signatures satisfy the spatial selection criterion. The cost of this task is proportional to the number of explored clusters multiplied by the number of candidate subclusters per cluster. The number of candidate subclusters per cluster is is given by the clustering function (see Section 3.3.5).

4.3.2 Data object insertion

When inserting a new data object in the spatial database, we need to find a cluster where to place the given object. Beside the root cluster whose signature accepts any spatial object, other database clusters might also be able to accommodate the new data object. Figure 4.8 illustrates the insertion procedure. Among the clusters capable to receive the new data object, identifiable based on their signatures, we choose to place the object in the cluster with the lowest access probability (Steps 1-2). Although simple, our insertion strategy aims at minimizing the probability of accessing the new data object, during spatial selections that do not include it in the query answers. The insertion of the new data object has to increment the number of data objects associated with the selected cluster, and with the relevant candidate subclusters of the selected cluster (Steps 4-6).

ObjectInsertion (data object ρ)

// Determine and select best cluster accepting object ρ:
1. **let** $\mathcal{B} \leftarrow \{\mathsf{b} \in \mathcal{C} \mid \rho \text{ matches } \sigma(\mathsf{b}) \ \wedge \ p(\mathsf{b}) \leq p(\mathsf{c}), \forall \mathsf{c} \neq \mathsf{b} \in \mathcal{C}\}$;
2. **let** $\mathsf{b} \in \mathcal{B}$;

// Insert data object ρ into selected cluster b:
3. **let** $objects(\mathsf{b}) \leftarrow objects(\mathsf{b}) \cup \{\rho\}$

// Increment data statistics for cluster b:
4. **let** $n(\mathsf{b}) \leftarrow n(\mathsf{b}) + 1$;

// Update data statistics for candidate subclusters of cluster b:
5. **let** $\mathcal{S} \leftarrow \{\mathsf{s} \in candidates(\mathsf{b}) \mid \rho \text{ matches } \sigma(\mathsf{s})\}$;
6. **for each** s **in** $\mathcal{S}$ **do**
7. **let** $n(\mathsf{s}) \leftarrow n(\mathsf{s}) + 1$;

End.

Figure 4.8: Object insertion procedure

Complexity of the data object insertion The complexity of the data object insertion is given by: (i.) checking the cluster signatures to identify the clusters able to accommodate the new data object, (ii.) determining the less accessed cluster among the qualifying clusters, (iii.) inserting the object in the selected cluster, and (iv.) updating the data object statistics for the selected cluster and for the candidate subclusters of the selected cluster whose signatures match the new data object. The cost of the first task is in the worst case proportional to the total number of clusters in terms of signature checks. However, in practice, the hierarchy of clusters helps to avoid checking all the cluster signatures. Indeed, the clusters descending from a cluster whose signature does

not accept the new data object do not accept this object either. The second task only regards the clusters qualifying for object insertion. Their access probabilities are evaluated and the cluster with the smallest access probability is retained. The cost of this operation is linear with the number of clusters able to host the new data object. The third task usually requires one write operation to place the new data object at the end of the selected cluster. When the clusters are stored on disk, this involves one disk seek/access operation and one object write. This consideration assumes that free spaces for new data objects are available in the selected cluster. To ensure fast data object insertions, a number of free spaces are by default reserved at the end of each cluster created or relocated. However, rare situations can occur where the storage space of a cluster might have no space left for new data objects. In such a case, another storage space, large enough, is assigned to the given cluster and all the data objects are relocated to the new storage space. The cost of the object relocation is proportional to the size of the cluster in terms of buffered read/write operations. When such situation occurs, a number of free places are reserved in the new storage space, proportional to the cluster size, so ensuring fast future data object insertions. The cost of the forth task is proportional to the number of candidate subclusters for which we need to increment the number of matching data objects. The number of relevant candidate subclusters is given by the clustering function and is equal to the number of dimensions N_d (see Section 3.3.5). Since only one data object is involved, the cost of the last action is negligible, compared, for instance, to the cost of the initial checking of the cluster signatures.

4.3.3 Data object deletion

To remove a data object from the spatial database, we first need to find the cluster hosting the wanted object. For this purpose, all the clusters able to host the given object need to be searched. Figure 4.9 illustrates the object deletion algorithm. The clusters whose signatures enclose the wanted object are identified in Step 1. The data objects from these clusters are individually compared to the wanted object, until the cluster containing the wanted object is discovered (Steps 2-4). The data object is then removed from the corresponding cluster (Step 5). The deletion of the given object requires to decrement the data statistics of the corresponding cluster (Step 6), and of the relevant candidate subclusters of the corresponding cluster (Steps 7-9). Once the wanted object is found and removed, the deletion operation ends (Step 10).

When performing a data object deletion, the last object from the cluster takes the place of the object removed. This way, free spaces are always located at the end of the cluster, which simplifies the data object insertion.

Complexity of the data object deletion The cost of the data object deletion is in the worst case given by the complexity of the spatial query execution, plus the cost of removing the data object from the corresponding cluster. Indeed, the cluster containing the wanted object is found by means of a spatial query whose selection criterion is the exact signature matching. In practice, the object deletion operation is faster than a common spatial query. On the one hand, the exact matching criterion is much more selective than a common range query (i.e., intersection), and therefore fewer clusters are explored. On the other hand, the cluster explorations end as soon as the cluster

```
ObjectDeletion ( data object ρ )

    //      Determine the clusters to be explored:
    1.      let 𝒳 ← {c ∈ 𝒞 | ρ matches σ(c)};

    //      Cluster explorations:
    2.      for each cluster c ∈ 𝒳 do

    //          Consider all data objects from cluster c:
    3.          for each object o in objects(c) do

    //              Look for the wanted object ρ:
    4.              if (ρ equals o) then

    //                  Remove data object ρ from the cluster c:
    5.                  let objects(c) ← objects(c) \ {ρ}

    //                  Decrement data statistics of cluster c:
    6                   let n(c) ← n(c) − 1;

    //                  Update data statistics of candidate clusters of cluster c:
    7.                  let 𝒮 ← {s ∈ candidates(c) | ρ matches σ(s)};
    8.                  for each s in 𝒮 do
    9.                      let n(s) ← n(s) − 1;

    //                  End deletion procedure:
    10.                 go to End;

End.
```

Figure 4.9: Object deletion procedure

containing the wanted object is found. Because the place of the object removed is taken by the last object from the cluster, the deletion operation requires two data accesses: one object read, and one object write. This adds the cost of a supplementary data seek/access operation to the cost of the deletion procedure, but facilitates the management of free spaces at cluster level.

4.4 Conclusions

In this chapter we provided algorithms and execution procedures for database clustering operations (i.e., cluster restructuring invocation, cluster split and cluster merge) and for standard database manipulation operations (i.e., spatial query execution, data object insertion and data object deletion).

The database clustering operations were the first topic of this chapter. We first discussed the invocation of the cluster restructuring operations. Periodical restructuring of the existing clusters is necessary in order to accomplish and refine the object clustering and to adapt it to important changes that might occur over time in data or query distributions. The restructuring operations are invoked on a periodical basis determined by

the occurrence of a fixed number of spatial queries. This allows the system to gather sufficient query statistics such as to properly support clustering decisions. As a general rule, we first consider cluster merges and then consider cluster splits. This restructuring order is intended to enable alternative splits for clusters having acquired additional data objects as result of cluster merges. The cluster merges and the cluster splits are invoked in a recursive manner for all the clusters from the clustering hierarchy starting with the root. However, restructuring operations are only performed when: (1) a sufficient number of query statistics are available at the level of the clusters involved, such as to properly support the clustering decisions; and when (2) the corresponding restructuring operations are estimated as profitable for the average query performance with respect to the cost model. The period in number of spatial queries at which the cluster restructuring operations are invoked, and the minimum number of spatial queries required to visit a cluster before considering a restructuring operation, are two parameters that help the system to schedule and accomplish the restructuring operations in a gradual manner. The clusters frequently explored are more often considered for restructuring, because they contribute more to the average query cost.

We further analyzed the two cluster restructuring operations. We first presented and discussed the cluster merge operation. A cluster merge involves a cluster and its parent, and is performed when the merge benefit function considers this operation as profitable for the average query performance with respect to the cost model. When performing a cluster merge, all the data objects from the child cluster are transferred to the parent cluster. The data statistics of the parent cluster and of the candidate subclusters of the parent cluster need to be updated in order to take into account the new acquired objects. As required by the clustering strategy, the data objects need to be contiguously stored in the parent cluster in order to minimize the cluster exploration cost. For this purpose, the physical implementation of the cluster merge consists of moving the data objects from the two clusters to a new storage location large enough to fit all the objects. The object relocation is done by means of buffered read and write operations in order to reduce the I/O costs.

We also presented and discussed the cluster split operation. The split of a cluster is achieved by materializing some of the candidate subclusters of the given cluster. To identify the candidate subclusters whose materializations are expected to be profitable for the average query performance, the materialization benefit function is evaluated for all the subcluster candidates. The most profitable candidates are materialized first, in a greedy manner. When a candidate cluster is materialized, the qualifying objects are transfered from the initial cluster to a new created cluster. The data statistics need to be accordingly updated for the initial cluster and for the candidate subclusters of the initial cluster, as well as for the new cluster and for the candidate subclusters of the new cluster. The data objects have to be contiguously stored in the two resulting clusters. For this purpose, the physical implementation of a cluster split consists of moving the data objects from the initial cluster to two new storage locations: the first corresponding to the new cluster, and the second meant to acquire the objects remaining in the initial cluster. The object relocation is done by means of buffered read and write operations in order to reduce the I/O costs. The initial cluster might be split several times, if multiple profitable candidate subclusters exist. Before attempting a new split, the materialization benefit function has to be reevaluated for the remaining candidate subclusters, in order to take into account the data changes induced at the cluster level by previous splits.

The standard database manipulation operations were the second topic of this chapter. We first considered the algorithm for the spatial query execution. When executing a spatial range query, the clusters whose signatures satisfy the spatial selection criterion relative to the query object (i.e., intersection, containment, enclosure, or similar shape test) are explored and the data objects belonging to them are individually checked. The query statistics of the clusters explored during the execution of a spatial query need to be incremented, as well as the query statistics of the candidate subclusters (of the explored clusters) that match the selection criterion. The cost of the spatial query execution depends on the query selectivity. We remind that the cost model supporting the object clustering is meant to improve the average performance of spatial range queries and to globally ensure better search performance than sequential scan.

The database update operations were also considered: data object insertions and data object deletions. The data object insertion checks the query statistics associated with the clusters whose signatures accept the new object and places the object in the cluster with the lowest access probability. Although simple, our insertion strategy is intended to minimize the object's probability of being accessed during spatial queries. Physically, the new data object is stored at the first free place of the selected cluster. Free places are reserved for new data objects at the end of clusters in order to avoid cluster relocations during object insertions. The data statistics of the cluster selected to host the new object are incremented, as well as those of the candidate subclusters (of the selected cluster) accepting the object. The object insertions are fast operations because the clusters are not explored during object insertions. The cluster signatures and the cluster statistics are maintained in memory. The signature verification and the computation of the access probability are very simple operations. Only one I/O access is required to physically write the new object at the storage position in the selected cluster.

The data object deletion has to explore all the clusters whose signatures accept the query object until the wanted object is retrieved. However, the selection criterion (exact signature matching) is much more selective than a common spatial query. Therefore, in practice fewer clusters are explored and the object deletions perform faster than common spatial queries. Physically, the place of the data object removed is taken by the last object from the corresponding cluster. This requires an additional I/O access, but simplifies the management of free spaces at cluster level and facilitates object insertions. The data statistics have to be decremented for the cluster from which the object was removed, as well as for the candidate subclusters (of the given cluster) whose signatures were matching the object removed.

In this chapter we presented and analyzed from a theoretical point of view the database clustering operations and the standard database manipulation operations. In the next chapter, we provide more details on the physical implementation and proceed to an extensive experimental evaluation of our clustering method.

Chapter 5

Implementation and performance evaluation

To show the practical relevance of our clustering approach, we implemented it and performed an extensive experimental evaluation. In the first part of this chapter, we address some implementation related aspects such as database storage management and memory management. In the second part, we present an advanced performance study of our clustering solution. We experimentally evaluate the search performance, the adaptability and the update performance of our clustering method, comparing it to alternative indexing techniques like R*-tree, X-tree, and Sequential Scan.

Chapter organization Section 5.1 addresses implementation related aspects. Section 5.2 presents a series of experiments. Conclusions are provided in Section 5.3.

5.1 Implementation considerations

The cost model supporting our clustering strategy requires that data objects from a cluster be contiguously stored in order to minimize the cluster exploration cost during spatial selections. The database storage space management has to ensure compliance to this requirement without significantly affecting the system's availability. Its implementation is explained in Section 5.1.1. In addition to the management of the database storage space, the database system has to manage the hierarchy of cluster signatures and to ensure data and query statistics maintenance for the existing clusters and for the candidate subclusters associated with the existing clusters. The data structures enabling the implementation of our clustering approach are presented in Section 5.1.2.

5.1.1 Database storage management

With the evolution of the computing equipments, the cost per byte of memory and of hard-disk devices has significantly lowered. This has lead to an important increase in the number of computing systems commonly equipped with very large amounts of main memory (several gigabytes) and with hard-disk devices of very high storage capacity (several terabytes). In this context, for many applications the storage capacity is not representing

a limiting factor anymore. This consideration primarily applies to applications relying on disk-based storage, but it can also apply to fast applications running on large memory systems and using the memory as storage support to ensure high execution performance. When the storage space does not represent a critical resource, extra storage space can be used to improve the application's execution performance.

Our clustering solution is intended to support either a main memory database storage, or a disk-based database storage. In either cases, the database storage space is globally managed as a large and continuous storage unit, where the storage spaces of the database clusters are allocated in a controlled manner. Due to our clustering strategy, the database system has to deal with a dynamic collection of clusters of various sizes. This might raise two performance problems related to the storage space management: First, the contiguous cluster storage requirement could trigger expensive cluster relocations during data object insertions. Second, the dynamic creation and removal of the database clusters could lead to an excessive fragmentation of the global database storage space. We next present our approach to deal with these potential storage management problems.

How to avoid cluster relocations during new data object insertions: To avoid frequent cluster relocations during data object insertions, we reserve a number of free places at the end of each database cluster, created or relocated. This way, a cluster does not require relocation during object insertions, unless all the free places are exhausted. When the storage space of the cluster gets full, the cluster is relocated and a number of new free spaces are reserved. For the number of reserved places, we consider 25% of the cluster size, thus globally taking into account the data distribution. Indeed, larger clusters will have more free places than smaller clusters. The free places from a cluster are always located at the end of the cluster. When the cluster needs to be explored, only the first part of the cluster is fetched, corresponding to the data objects actually stored at the cluster level. By reserving free places, we only waste storage space, but not processing time, since the empty parts of clusters are not subject to reading operations.

How to avoid situations of excessive fragmentation of the global storage space: The free space from the global storage unit can get fragmented over time due to cluster relocations. Cluster relocations are occurring during cluster splits, cluster merges, and rarely during data object insertions. Excessive free space fragmentation can lead to situations where no contiguous storage can be allocated for new clusters, even though the sum of the total free space exceeds the required cluster sizes. In such cases, a global relocation of the existing clusters has to be performed in order to obtain a contiguous free space placement. Such an operation is called defragmentation. The defragmentation of the storage space is an expensive operation which might involve many cluster relocations. Therefore, it should be avoided. In practice, the probability of having a fragmentation problem depends on the ratio between the total capacity of the global storage space and the size of the set of data objects. The larger the free space size in the global storage space, the lower the probability that a fragmentation problem will occur. In practice, to avoid frequent storage space defragmentations, the capacity of the storage space should be at least three times larger than the expected database size.

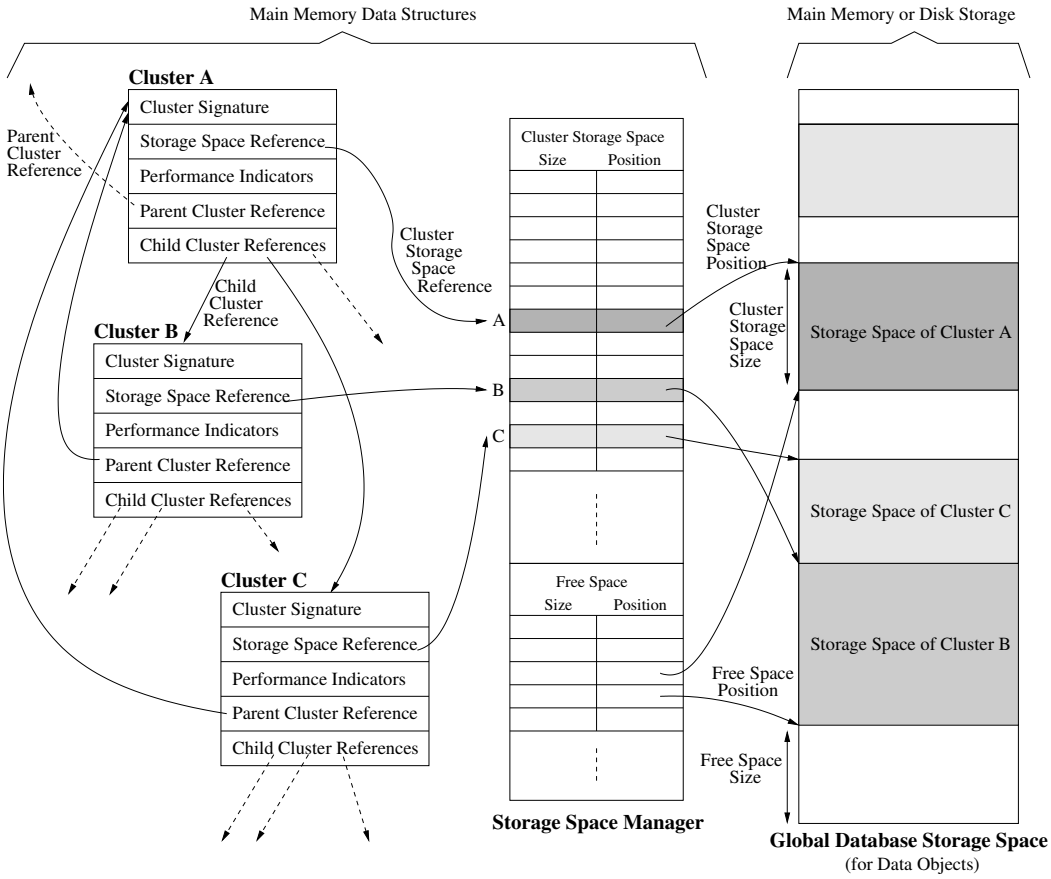


Figure 5.1: Data structures enabling the implementation of our clustering approach

5.1.2 Data structures supporting our clustering solution

The data structures enabling the implementation of our clustering approach are outlined in Figure 5.1. On the one hand, we distinguish the data structures handling in main memory the information relative to the database clusters, to their hierarchical organization, and to the storage spaces allocated for them in the global storage space. On the other hand, we have the *Global Database Storage Space* where the data objects belonging to clusters are physically stored. According to the chosen storage scenario, the global database storage can be set either in main memory, or on disk. In Figure 5.1, the data structures handling the information associated with clusters are represented by *Cluster A*, *Cluster B*, and *Cluster C*. We have also depicted an additional data structure, called *Storage Space Manager*, whose role is to handle the information associated with cluster storage spaces (i.e., position in the global storage space and size) and regarding the free space from the global storage space (i.e., free place position and size). Further details on the data structures used to support our clustering method are provided next.

Cluster data structure

As depicted in Figure 5.1 (*Cluster A*, *Cluster B*, and *Cluster C*), the data structure associated with a database cluster embeds the following elements:

dim_{id}	a_{min}	a_{max}	b_{min}	b_{max}
1	0.25	0.50	0.50	0.75
2	0	1	0	1
3	0	1	0	1
4	0.00	0.25	0.75	1.00
	⋮			
$N_d - 1$	0.50	0.75	0.75	1.00
N_d	0	1	0	1

(A) Complete Signature Representation

N_r	dim_{id}	a_{min}	a_{max}	b_{min}	b_{max}
1	1	0.25	0.50	0.50	0.75
2	4	0.00	0.25	0.75	1.00
	⋮				
N_r	$N_d - 1$	0.50	0.75	0.75	1.00

(B) Representation of Relevant Dimensions

Figure 5.2: Possible representations for a cluster signature

- cluster signature
- storage space reference
- performance indicators
- parent cluster reference
- list of references to the child clusters

We next detail each of these elements.

Cluster signature. The cluster signature follows the definition from Section 3.3.3. In each dimension we have four real values: two values for each of the two intervals of variation, defining the positions and the extents of the qualifying spatial objects in the corresponding dimension. According to our grouping criterion, all the cluster signatures are subsignatures of the root cluster signature. The intervals of variation of the root cluster signature are set to full domains in all dimensions. The other clusters, obtained by means of cluster splits, have their signatures derived from the root cluster signature, or, in a recurrent manner, from subsignatures of the root cluster signature. As ensured by the clustering function (see Section 3.3.5), a direct subsignature of a cluster signature differs from the cluster signature in only one dimension. This means that a direct subsignature of the root cluster signature has all the intervals of variation set to full domains, except for one dimension. A subsignature of level k of the root cluster signature differs from the root cluster signature in at most k dimensions. The rest of dimensions have their intervals of variation set to full domains. When the number of dimensions N_d is high, only a small fraction of dimensions are used for the data space division. In such a case, a cluster signature has many dimensions with intervals of variation set to full domains. The corresponding dimensions are not relevant for membership or spatial selection tests because full domains make no discrimination between spatial objects. Therefore they do not need to be represented in the cluster signature. An alternative representation of the cluster signature consists of storing only the information corresponding to the dimensions for which the intervals of variation are not full domains. Such a representation can help to save memory space. However, the number and the identifiers of the relevant

dimensions need to be represented in the data structure too. Figure 5.2 illustrates the two possible representations for the cluster signature. Figure 5.2-(A) shows the complete signature representation which implies the storage of the min and max values for the two intervals of variation corresponding to each of the N_d dimensions. Figure 5.2-(B) shows the condensed signature representation which implies the storage of the min and max values for the two intervals of variation corresponding to the relevant dimensions only.

Storage space reference. The storage space reference consists of a numerical identifier indicating the entry from the *Storage Space Manager*, that handles the information associated with the storage space allocated for the corresponding cluster.

Performance indicators. The performance indicators consist of data and query statistics associated with the cluster and with the candidate subclusters of the cluster. For a cluster c we need to handle the following information:

- $n(\mathsf{c})$ – the number of data objects belonging to the cluster
- $q_{last}(\mathsf{c})$ – the number of spatial queries addressed to the system before the cluster's creation, or before the last reinitialization of the query statistics
- $q(\mathsf{c})$ – the number of spatial queries visiting the cluster since the cluster's creation, or since the last reinitialization of the query statistics
- $p_{last}(\mathsf{c})$ – the access probability before the last reinitialization of the query statistics

The query statistics $q(\mathsf{c})$ and $q_{last}(\mathsf{c})$ are used to compute the access probability associated with the cluster c, according to the formula

$$p(\mathsf{c}) = \frac{q(\mathsf{c})}{q(\mathsf{system}) - \ q_{last}(\mathsf{c})}$$

where $q(\mathsf{system})$ represents the total number of spatial queries addressed to the system. The value $p_{last}(\mathsf{c})$, representing the cluster's access probability before the last reinitialization of the query statistics, has to be recorded for usage that might be required before new query statistics are gathered (e.g., to assist insertion of new data objects).

Besides the data and the query statistics associated with the cluster, we also have to maintain data and query statistics for a number of candidate subclusters of the given cluster. As ensured by the clustering function (see Section 3.3.5), in the general case, the number of candidate subclusters of a cluster is equal to $N_d \cdot f^2$, where N_d represents the number of dimensions and f represents the space division factor. For each candidate subcluster s of the cluster c, we have to maintain the following statistics:

- $n(\mathsf{s})$ – the number of data objects from the cluster c matching the signature of the corresponding candidate subcluster s
- $q(\mathsf{s})$ – the number of spatial queries matching the signature of the candidate subcluster s since the creation of the cluster c or since the last reinitialization of the query statistics

The query statistics $q(\mathsf{s})$ is used to compute the access probability of the candidate subcluster s, according to the formula

$$p(\mathsf{s}) = \frac{q(\mathsf{s})}{q(\mathsf{system}) - \ q_{last}(\mathsf{c})}$$

which also makes use of the query statistics $q_{last}(\mathsf{c})$ of the cluster c.

Because in the general case the number of candidate subclusters of a cluster is equal to $N_d \cdot f^2$, the statistics n and q corresponding to the candidate subclusters are stored in arrays of size $N_d \cdot f^2$. The position in the two statistics arrays corresponding to a candidate subcluster is determined by the dimension associated with the given subcluster, and by the choice of the two subintervals of variation associated with the given subcluster (see the clustering function).

When performing object insertions, deletions, relocations or selections, the signatures of the candidate subclusters have to be checked against the selection criterion in order to update the data and the query statistics associated with the candidate subclusters of the involved clusters. However, the signatures of the candidate subclusters do not need to be stored because they can be easily derived from the signature of the corresponding cluster by using the clustering function: The signature of a candidate subcluster differs from the signature of the corresponding cluster in only one dimension. Therefore the matching criterion needs to be checked for the relevant dimension only.

Summing up, in order to manage the performance indicators of a cluster we require $3 + 2 \cdot N_d \cdot f^2$ integer values for the data and the query statistics associated with the cluster and with its candidate subclusters, and one real value to record the last access probability computed for the cluster.

Parent cluster reference. The parent cluster reference consists of a pointer to the cluster data structure representing the parent of the cluster in the clustering hierarchy.

List of child cluster references. This data structure consists of a list of pointers to the cluster data structures associated with the children of the cluster in the clustering hierarchy. The actual number of children depends on how many candidate subclusters are materialized during the clustering process. The number of children of a cluster can not exceed the total number of candidate subclusters, which, according to the clustering function, is equal to $N_d \cdot f^2$ (see Section 3.3.5).

Storage space manager structure

To facilitate the management of the global database storage space, we use a data structure referred to as *Storage Space Manager* (see Figure 5.1). The role of the *Storage Space Manager* is to handle the information regarding the storage spaces of clusters in the global storage space, and regarding the free space distribution in the global storage space.

The information concerning the storage space allocated for a database cluster consists of size and position in the global database storage space. The position is used to provide direct access to the corresponding storage location when the cluster needs to be accessed

during spatial queries or object update operations. The storage space size indicates the total number of places available for data objects in the given cluster. Remember that the number of data objects that are actually stored in the cluster is stored in the cluster data structure. An entry is allocated for each database cluster in the *Storage Space Manager*. This entry is referenced in the data structure corresponding to the database cluster.

The *Storage Space Manager* also handles the information concerning the free space from the global database storage space. This information is needed to enable storage space allocation for new database clusters, to recollect the space corresponding to withdrawn clusters, and to handle situations of excessive storage fragmentation. For this purpose, in the *Storage Space Manager* we keep an ordered list of free space positions, together with their corresponding sizes. To allocate storage spaces for new database clusters, we adopt a simple allocation politics: To each new cluster we allocate the first free position where enough space is available to fit the required cluster size. If no suitable position is found, the storage defragmentation operation is triggered. However, such a situation is not likely to occur frequently if the amount of free space is large enough compared to the actual size of the spatial database.

5.2 Performance evaluation

To show the practical relevance of our cost-based query-adaptive clustering solution, we performed extensive experiments, executing spatial range queries over large collections of spatial objects with many dimensions, using uniform and skewed data and query distributions. We compared our technique to R*-tree, to X-tree, and to Sequential Scan, evaluating the query execution time, the number of cluster/node accesses, and the amount of data objects actually verified. We investigated the behavior of our clustering method and its ability to adapt to query and data distribution changes. We also analyzed the performance of database update operations like object insertions and object deletions.

Organization. Section 5.2.1 presents our experimental setup providing details on each of the following elements: experimental platform, database storage scenarios, data representation, competing indexing methods, definition of cost model and execution parameters (for our clustering method, as well as for the alternative indexing methods), experimental procedure, workload parameters and performance indicator parameters. Section 5.2.2 presents a number of experiments meant to evaluate the average search performance of the four competing methods under various workloads consisting of static datasets of spatial objects with uniform and skewed spatial distributions and of spatial range queries with different selectivities and spatial distributions. Section 5.2.3 presents additional experiments meant to examine the behavior and the ability of our clustering method to adapt to dynamical changes in query and data distributions. Several aspects are considered such as evolution of search performance, evolution of number of clusters, time required to accomplish the database clustering, costs of cluster restructuring operations, and costs of database update operations.

5.2.1 Experimental setup

Experimental platform We ran all experiments on a Pentium IV workstation with an i686 CPU at 3 GHz and 1 GBytes RAM, operating under Linux. The execution platform had a SCSI disk with a storage capacity of 36 GBytes and presenting the following performance characteristics: average disk access/seek time of 10 ms, and average sustained transfer rate of 90 MBytes/sec.

Database storage scenarios In our tests we considered two database storage scenarios: a main memory database storage and a disk-based database storage. When testing the disk-based storage scenario, we limited the capacity of the main memory to 64 MBytes. We also deactivated the file caching system, and we used experimental datasets at least twice larger than the available memory. This way we ensured data transfer between disk and memory.

Data representation The representation of a spatial object consists of an object identifier and of N_d pairs of real values representing the interval limits of the corresponding object in the N_d dimensions of the data space. The interval limits, as well as the object identifier, are each represented on 4 bytes. Accordingly, the size in bytes of a data object (in memory or on disk) is given by the following formula:

$$size\_in\_bytes(o) = (1 + 2 * N_d) * 4$$

Competing methods Our cost-based query-adaptive clustering solution is further referred to as *Adaptive Clustering*. We compared the performance of our Adaptive Clustering to the following competing methods:

1. *Sequential Scan*

   Sequential Scan is the simplest method to answer queries. It is based on a sequential storage of the data objects and it consists of scanning the entire database and checking all the objects against the selection criterion. Although quantitatively expensive, Sequential Scan benefits of good data locality, and thus of fast (sustained) data transfer between disk and memory. In high-dimensional spaces, Sequential Scan has become a comparison reference because it often outperforms advanced tree-based indexing methods like R*-tree, Hilbert R-tree, or X-tree [BBK98b, BK00].

2. *R*-tree*

   R*-tree is the most famous variant of R-trees. It has been widely accepted in literature and is often used as reference for performance comparisons. The R*-tree implementation used in our tests follows [BKSS90]. We used a node page size of 16 KBytes. Considering a storage utilization of 70%, a node accommodates, for instance, 86 objects with 16 dimensions, or 35 objects with 40 dimensions. Using smaller node page sizes would lead to creation of too many tree nodes resulting in very high overheads due to numerous node accesses.

Table 5.1: Average I/O and CPU operations costs

I/O operation	**Cost**
disk access/seek time	10 ms
disk transfer rate	90 MBytes/sec
disk transfer time (per byte)	$1.06 \cdot 10^{-2}$ μs/byte
CPU operation	**Cost**
average cluster signature check time	0.5 μs
cluster access & statistics update time	2 μs
average object verification rate	2300 MBytes/sec
verification time (per byte)	$4.15 \cdot 10^{-4}$ μs/byte

3. *X-tree*

   X-tree is a cost-based R-tree extension meant to alleviate the performance deterioration of classical R-trees. The X-tree implementation follows [BKK96]. The page size corresponding to a normal node was set to 16 KBytes, as for R*-tree. The supernodes have sizes that are multiples of the elementary page size. For node splits we used the split algorithm of R*-tree, as recommended by the authors of X-tree. We do not record information regarding the split history at node level because the proposed method to determine an overlap-minimal split dimension does not apply when dealing with data objects with spatial extents. Supernodes are accessed through buffered I/O operations (reads and writes) such as to minimize the I/O costs, as we do for clusters in our clustering approach.

Cost model and execution parameters for Adaptive Clustering

- *Cost model parameters*

  Parameters A, B, and C are part of the cost model supporting the clustering strategy. They are determined by the performance characteristics of the execution platform and depend on the storage scenario adopted for the database: main memory storage or disk-based storage. The system-specific parameters, namely the costs of I/O and CPU operations, are evaluated in advance and used to determine the values of the cost model parameters A, B and C with respect to the storage scenario, as described in Section 3.4.3. The average cost values for I/O and CPU operations measured for our system and used to compute the cost model parameters A, B and C are provided in Table 5.1.

- *Domain division factor*

  For the domain division factor used to implement the clustering function (see Section 3.3.5), we tested several different values: $f \in \{2, 3, 4, 5\}$. We found that a domain division factor $f = 3$ ensures the best trade-off between the number of candidate subclusters and the cost of statistics maintenance. Thus, in all the experiments reported here, we used a domain division factor $f = 3$. Accordingly, the number of candidate subclusters associated with a database cluster ranges between $6 * N_d$ and $9 * N_d$, where N_d represents the space dimensionality. For example, in a 16-dimensional space, each database cluster will have between 96 and 144 candi-

date subclusters. The candidate subclusters are virtual, so only their performance indicators need to be maintained.

- *Cluster restructuring invocation period*

  Cluster restructuring is periodically invoked to accomplish and refine the object clustering and to adapt it to important changes that might occur over time in data or query distributions (see Section 4.2.1). In our experiments, cluster restructuring operations are globally invoked every 100 spatial queries ($\mathsf{period\_q} = 100$). The minimum number of spatial queries required to visit the clusters involved in restructuring operations is of $\mathsf{min\_q} = 25$. We tried several combinations $\mathsf{period\_q}/\mathsf{min\_q}$. The 100/25 combination proved to be a good choice: on the one hand it allows the system to gather sufficient statistics such as to properly support the clustering decisions, and on the other hand it enables the system to schedule the cluster restructuring operations in a gradual manner, as such as to not affect its availability significantly.

X-tree cost model and consequences As already shown in the related work (see X-tree considerations in Sections 2.2.6 and 2.2.7), an overlap threshold value $MaxO$ is calculated with respect to the performance characteristics of the execution platform, based on which the directory nodes are split or extended to supernodes. With respect to our system's parameters (see Table 5.1), depending on the storage scenario adopted for the database, we obtained the following behaviors:

1. *In memory* the computed $MaxO$ threshold is very high ($\approx 94\%$). As a result, directory nodes are always split and X-tree behaves like a classical R*-tree. This is normal because random access in memory is not much more expensive than sequential access, and the R*-tree remains globally effective.

2. *On disk* the computed $MaxO$ threshold is very low ($\approx 2\%$). As a result, the root node is never split and the X-tree degenerates in a tree with a single supernode, the root. All the leaf nodes are direct children of the root supernode. The advantage of this configuration is that a new data object is always inserted in the leaf node which best accommodates it (requiring least MBR enlargement). A drawback is that the insertion time becomes very expensive since all the MBRs from the root supernode have to be verified in order to identify the leaf nodes that best fit the new data objects. In the best case, the insertion cost is proportional to the number of entries from the root supernode. The number of entries increases when the supernode is extended while more data objects are added to the indexing structure. When the root supernode overflows (which corresponds to the worst case of the insertion procedure), a node split should be attempted in order to measure the resulting overlap and to decide the node's split or further extension. The split algorithm is highly expensive, almost quadratic with the number of node entries and linear with the number of dimensions. However, this computation effort is worthless in practice where splits of the root supernode never happen. Indeed, insertion of more data objects is very unlikely to produce low-overlap split configurations. Based on this observation, we decided not to consider root supernode splits at all. This allows to accelerate the object insertion, thus the construction time of the X-tree on disk. For example, in our case, the construction on disk of an R*-tree with 2000000 data

objects requires about one and a half hours. The construction of an X-tree requires more than 17 hours when supernode splits are attempted and about 5 hours when supernode splits are never considered.

Because in memory X-tree behaves like R*-tree, we only report performance of R*-tree. In contrast, on disk we separately examine the performance of R*-tree and X-tree.

Experimental procedure We indicate now the general experimental procedure followed to accomplish our tests:

- For Sequential Scan:

  The data objects are loaded and sequentially stored in a single cluster. Spatial queries are launched, and the average query execution time is raised.

- For R*-tree and X-tree:

  The data objects are first inserted in the indexing structure, then the search performance of spatial queries is evaluated in terms of average query execution time, number and percentage of accessed nodes, and number and percentage of verified objects.

- For Adaptive Clustering:

  The data objects are first inserted in the root cluster, then a number of spatial queries are launched to trigger the object clustering. Global cluster restructuring operations are invoked every 100 spatial queries ($\mathsf{period\_q} = 100$). If no significant changes occur in the query distribution, the clustering process reaches a stable state in less than 10 restructuring steps. Then we evaluate and report the average query response time and other performance indicators such as number and percentage of explored clusters, and number and percentage of data objects verified. The query time reported includes the time spent to update the query statistics associated with the accessed clusters/candidate subclusters since query statistics maintenance is an essential element of our clustering strategy.

Workload parameters and query performance indicators The following workload parameters were varied in our experiments:

- Number of data objects: up to 2000000
- Number of dimensions: from 16 to 40
- Query selectivity: between 0.00005% and 50%

In each experiment, a large number of spatial range queries are addressed to the indexing structure and average values are raised for the following query performance indicators:

- Query execution time – combining all the costs
- Number and percentage of clusters/nodes explored – relevant for the cost of random access operations
- Number and percentage of data objects verified – relevant for data transfer and object check costs

5.2.2 Search performance evaluation

In this section, we evaluate and compare the average search performance of the four competing methods under various workloads consisting of static datasets of spatial objects with uniform and skewed spatial distributions, and spatial range queries with different selectivities and spatial distributions, both in memory and on disk.

1. Uniform data objects and uniform queries with different selectivities

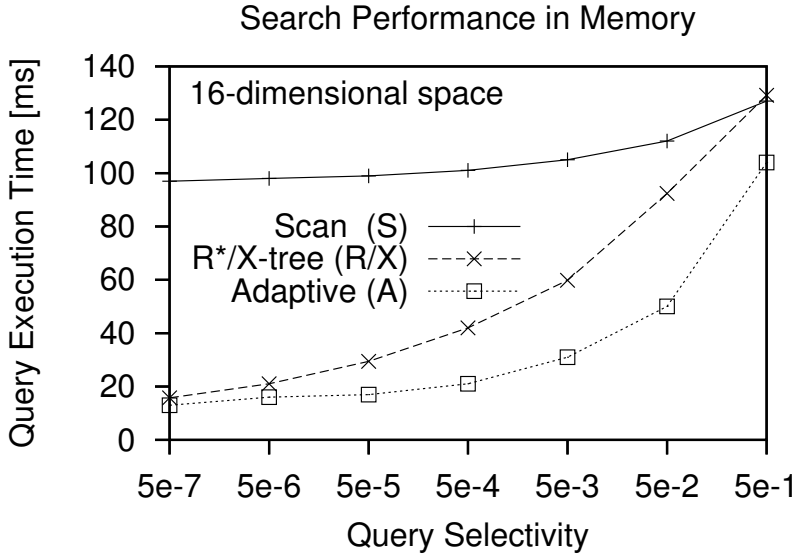
The objective of the first experiment is to investigate the influence of the query selectivity on the search performance. For this purpose, we launch uniform spatial queries with different selectivities on a dataset of spatial objects uniformly distributed in the multidimensional space. We analyze the general behavior of our Adaptive Clustering and compare its performance to R*-tree, X-tree and Sequential Scan, both in memory and on disk.

Experiment 1 We consider a dataset of 2000000 objects uniformly distributed in a 16-dimensional data space (251 MBytes of data). The intervals of the data objects are generated with random sizes and arbitrary positions in all dimensions. We evaluate the response time for intersection queries with different (decreasing) average selectivities: 0.00005%, 0.0005%, 0.005%, 0.05%, 0.5%, 5% and 50%. To ensure a given query selectivity, minimal and maximal sizes are imposed to the intervals of the query objects, which otherwise are uniformly distributed in each dimension.

Performance results are presented in Figure 5.3 for the memory storage scenario and in Figure 5.4 for the disk storage scenario. The charts from the two figures depict the evolution of the average query execution time with varying query selectivity, for the four competing methods: Adaptive Clustering (**A**), R*-tree (**R**), X-tree (**X**), and Sequential Scan (**S**). As already explained, in memory X-tree behaves like an R*-tree. Therefore, for the memory storage scenario, we only report the performance indicators of R*-tree under the notation **R/X**. Additional information helping to explain the resulting query times is provided in the tables accompanying the charts, regarding the data organization and the data access. The tables compare Adaptive Clustering, R*-tree, and X-tree in terms of: (1) total number of clusters, and respectively total number of nodes (directory nodes plus leaf nodes), (2) average ratio of explored clusters/nodes, and (3) average ratio of verified objects. Relative to the total number of cluster/nodes, the ratio of explored clusters/nodes is relevant for the random access cost, while the ratio of verified objects is relevant for the data transfer and verification costs. For Sequential Scan we already know that all the data objects are verified each time.

Experimental facts and remarks:

- Regarding Sequential Scan:
  - Every spatial query verifies all the data objects, no matter the storage scenario.
  - *In memory*, the query cost is dominated by the object verification. It slightly increases for queries with lower selectivity because the average number of dimension/interval checks per object increases. Indeed, when the query selectivity is lower, the data objects are more likely to be checked in several dimensions.



Memory Database Storage - Data Organization and Data Access						
Query	**Clusters**	**Nodes (dirs.+leaves)**	**Explored %**		**Verif. Objs. %**	
Select.	**Adaptive**	**R*-tree/X-tree**	**A**	**R/X**	**A**	**R/X**
5e-7	4839	273+23656	12	16	12	17
5e-6	5198	273+23656	14	21	14	22
5e-5	5062	273+23656	14	30	14	30
5e-4	5329	273+23656	17	41	18	41
5e-3	5145	273+23656	23	56	26	56
5e-2	4450	273+23656	34	80	39	80
5e-1	1493	273+23656	61	99	76	99

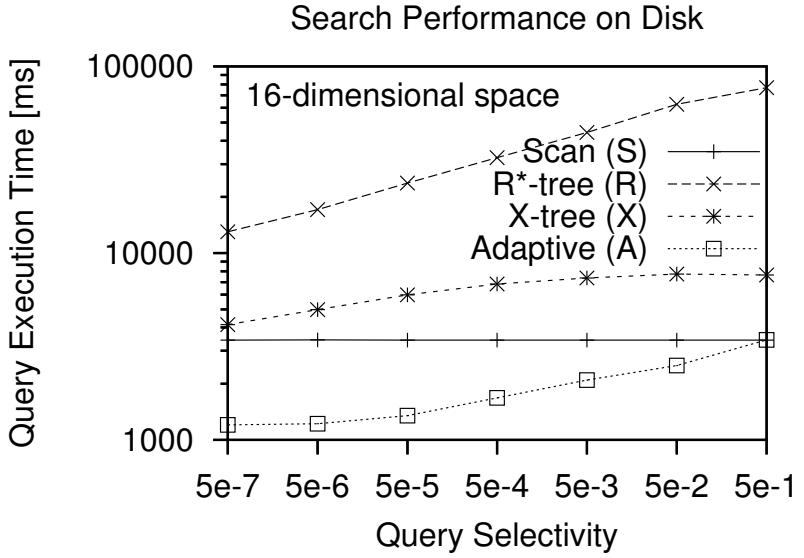
Figure 5.3: Search performance in Memory for different query selectivities (uniform data and query objects)

– *On disk*, the query cost is dominated by the I/O cost, which is proportional to the constant amount of data transfered from disk to memory.

- Regarding Adaptive Clustering:

  – Adaptive Clustering organizes the data objects in clusters. The number of resulting clusters is different depending on the storage scenario, as well as on the query selectivity.

  – *On disk*, when the queries are more selective more clusters are formed since only few of them are expected to be explored. When the queries are less selective fewer clusters are produced. Indeed, the exploration of too many clusters would otherwise trigger significant I/O access cost overhead. As shown in the table from Figure 5.4, the number of clusters significantly drops with decreasing query selectivity, from 1068 clusters at 5e-5% query selectivity to 16 clusters at 50% selectivity. Although a 50% query selectivity is highly unrealistic, when we forced such a selectivity only 16 clusters were formed, however succeeding to avoid a search performance degra-



Disk Database Storage - Data Organization and Data Access									
Query	**Clusters**	**Nodes (dirs.+leaves)**		**Explored %**			**Verif. Objs. %**		
Select.	**Adaptive**	**R*-tree**	**X-tree**	**A**	**R**	**X**	**A**	**R**	**X**
5e-7	1068	273+23656	1+22535	12	16	8	17	17	9
5e-6	818	273+23656	1+22535	14	21	12	20	22	12
5e-5	633	273+23656	1+22535	16	30	18	23	30	18
5e-4	395	273+23656	1+22535	21	41	29	32	41	29
5e-3	203	273+23656	1+22535	34	56	44	44	56	45
5e-2	125	273+23656	1+22535	44	80	73	59	80	74
5e-1	16	273+23656	1+22535	52	99	99	96	99	99

Figure 5.4: Search performance on Disk for different query selectivities (uniform data and query objects)

dation below Sequential Scan. The variation in the number of clusters with the query selectivity demonstrates the ability of Adaptive Clustering to take into account the query characteristics and to adapt the object clustering to the system's performance parameters. Adaptive Clustering explores only a fraction of existing clusters, succeeding to outperform Sequential Scan in most cases. Better performance gains are obtained for queries with higher average selectivity.

– *In memory*, much more clusters are formed because cluster accesses and cluster explorations are much less penalizing than on disk. As illustrated in the table from Figure 5.3, the variation in the number of clusters is more substantial for queries with lower selectivity. More than three times fewer clusters are produced in the extreme case of 50% query selectivity. Adaptive Clustering explores only a fraction of existing clusters, succeeding to outperform Sequential Scan in all the cases, even for queries with very low selectivity. The best performance gains are obtained for queries with higher average selectivity (up to 7 times faster than Sequential Scan).

– The number of clusters on disk is much smaller than the number of clusters in memory due to high costs of I/O operations. On disk, Adaptive Clustering produces fewer, larger clusters in order to reduce the number of expensive random I/O accesses. This demonstrates that our cost model is flexible and able to adapt to platform-specific performance parameters.

– We globally conclude that, thanks to the query-adaptive and hardware-oriented cost model, Adaptive Clustering succeeds to ensure better search performance than Sequential Scan both in memory and on disk.

- Regarding R*-tree:

  – R*-tree organizes the data objects in an indexing tree with 23656 leaf nodes and 273 directory nodes. The indexing tree has 4 levels from the root to the leaves. The average number of data objects per leaf node is of about 84.55. The indexing tree is the same no matter the query selectivity or the storage scenario.

  – *In memory*, R*-tree is less expensive than Sequential Scan in most cases, except for the extreme case of 50% query selectivity. As reported in the table from Figure 5.3, R*-tree explores/verifies only a fraction of nodes/objects. However, compared to Adaptive Clustering, R*-tree is globally more expensive. Indeed, the ratio of clusters explored with Adaptive Clustering is systematically smaller than the ratio of nodes accessed with R*-tree. In addition, the number of clusters of Adaptive Clustering is on average five times smaller than the number of R*-tree nodes. Although the clusters produced by Adaptive Clustering are larger than the R*-tree nodes, the object grouping is globally more efficient, because fewer objects are verified in the end, notably for queries with lower selectivity (compare the ratios of verified objects). Even for queries with selectivity as low as 50%, for which R*-tree practically checks the entire database (99%), only 76% of objects are verified by Adaptive Clustering.

  – *On disk*, R*-tree is much more expensive than Sequential Scan (note the logarithmic scale from the chart of Figure 5.4). The bad performance of R*-tree is due to the large number of nodes accessed, notably in a random manner. The difference in ratio of verified objects between R*-tree and Adaptive Clustering is lower on disk than in memory, but the cost overhead generated by expensive I/O accesses is significantly higher.

  – We conclude that in memory R*-tree is more efficient than Sequential Scan, but less efficient than our Adaptive Clustering. On disk, R*-tree is impractical showing much worse performance than Sequential Scan.

- Regarding X-tree:

  – X-tree is different from R*-tree only *on disk*, where it organizes the data objects in an indexing tree with 22535 leaf nodes and only one supernode, the root node. The size of the root supernode is equivalent to 182 elementary pages. All the leaf nodes are direct children of the root supernode. The average number of data objects per leaf node is of about 88.75.

  – X-tree is more selective than R*-tree. The percentage of explored nodes/verified objects is smaller. This is due to the fact that all the leaf nodes are children of the root supernode, and the data objects are always placed in the leaf nodes that

fit them best (requiring least MBR enlargement). The object grouping is more efficient, thus X-tree performs better than R*-tree. However, a high number of leaf nodes are still accessed, notably in a random manner, which results in a search performance worse than Sequential Scan.

– When the query selectivity is higher than 0.5%, X-tree verifies fewer data objects than Adaptive Clustering. Even though more objects are verified in the end, globally Adaptive Clustering shows better search performance than X-tree because it requires much fewer random I/O accesses. Indeed, the number of clusters of Adaptive Clustering is much smaller than the number of leaf nodes of X-tree, and so is the average number of I/O accesses. Considering for instance the case of the highest query selectivity (0.00005%), X-tree verifies 9% of the data objects by means of 1803 I/O accesses, while Adaptive Clustering verifies 17% of the data objects through 128 I/O accesses. X-tree fails to outperform Sequential Scan due to the high I/O cost overhead, while Adaptive Clustering succeeds.

– We conclude that X-tree performs better than R*-tree on disk, but is still less efficient than Sequential Scan. Even though X-tree is more selective than Adaptive Clustering for highly-selective queries, its performance suffers from numerous I/O accesses and degrades below Sequential Scan.

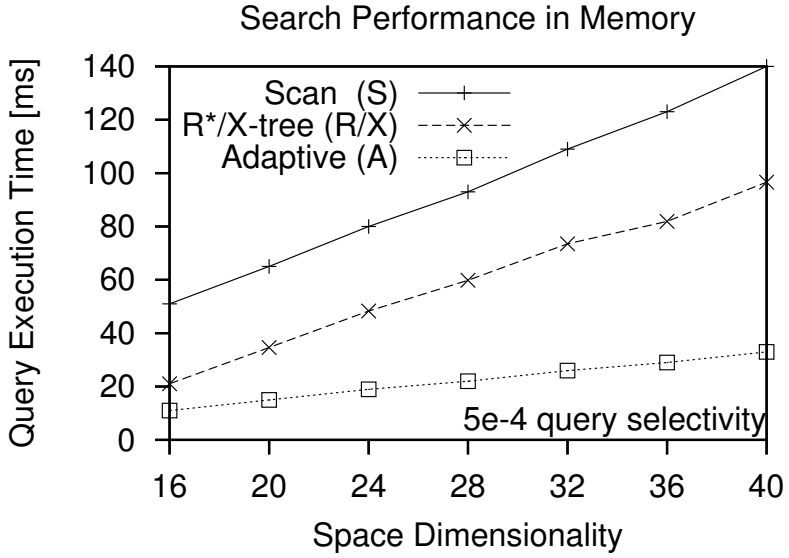
2. Data objects with skewness at dimension level, uniform queries with constant selectivity, data spaces with different dimensionalities

Data skewness at dimension level is closer to reality where different dimensions exhibit different characteristics. For example, from object to object, some dimensions can be more selective than others. With this experiment we intend to examine the search performance for datasets of objects with skewness at dimension level, and to investigate the evolution of the search performance with increasing dimensionality.

Experiment 2 We set up the following experimental scenario: 1000000 data objects with different size constraints over dimensions, and uniformly distributed query objects with no interval constraints. We enforced data skewness at dimension level as follows: For each data object a quarter of dimensions, arbitrarily chosen, were set to be twice more selective than the rest of dimensions. We still could control the global query selectivity because the query objects were uniformly distributed. For our tests, we ensured an average query selectivity of 0.05% (500 of 1000000 objects). We tested increasing space dimensionalities: 16, 20, 24, 28, 32, 36, 40. Performance results are illustrated in Figure 5.5 for the memory storage scenario, and in Figure 5.6 for the disk storage scenario.

Experimental facts and remarks:

- The average query execution time globally increases with the space dimensionality in both storage cases. This is normal because with increasing dimensionality the size of the dataset increases too, from 126MBytes at 16 dimensions to 309MBytes at 40 dimensions.



Memory Database Storage - Data Organization and Data Access						
Num.	**Clusters**	**Nodes (dirs.+leaves)**	**Explored %**		**Verif. Objs. %**	
Dims.	**Adaptive**	**R*-tree/X-tree**	**A**	**R/X**	**A**	**R/X**
16	2753	138+11833	19	40	18	40
20	3498	217+15129	20	52	19	52
24	4220	331+18328	20	58	19	58
28	4894	448+21808	20	61	19	62
32	5368	586+25530	20	67	20	68
36	6001	731+28108	20	65	20	65
40	6868	907+31382	20	69	20	69

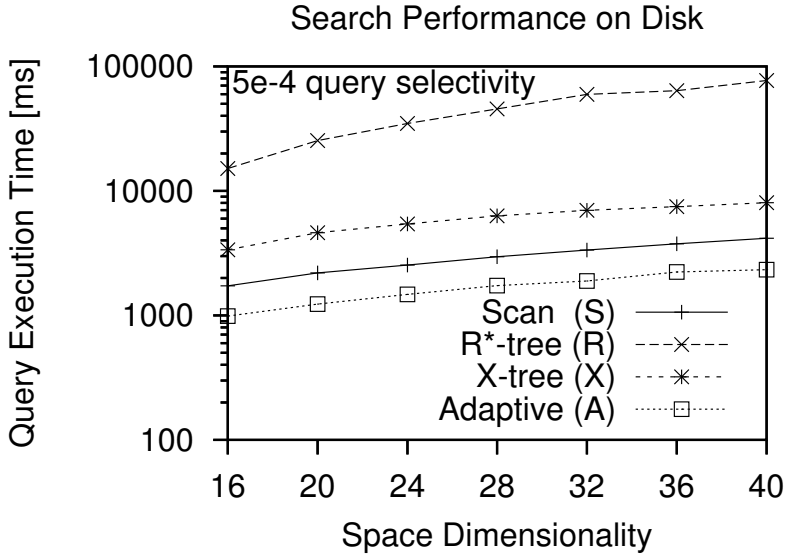
Figure 5.5: Search performance in Memory for different space dimensionalities (skewed data)

- Compared to Sequential Scan, Adaptive Clustering exhibits better search performance, scaling well with the number of dimensions, both in memory and on disk.

- *In memory:*

  – R*-tree shows better performance than Sequential Scan, but worse performance than Adaptive Clustering. Adaptive Clustering verifies between 18% and 20% of objects, while R*-tree verifies between 40% and 69% of objects. The ratio of verified objects is quite stable with increasing dimensionality for Adaptive Clustering, but degrades for R*-tree. This demonstrates that Adaptive Clustering takes better advantage of data skewness, succeeding to cluster the data objects according to their most selective dimensions and intervals. R*-tree benefits less from the skewness at dimension level and suffers more with increasing space dimensionality.

  – In this experiment, Adaptive Clustering is 5 times faster than Sequential Scan, and between 2 and 3 times faster than R*-tree.



Disk Database Storage - Data Organization and Data Access									
Num.	**Clusters**	**Nodes (dirs.+leaves)**		**Explored %**			**Verif. Objs. %**		
Dims.	**Adaptive**	**R*-tree**	**X-tree**	**A**	**R**	**X**	**A**	**R**	**X**
16	304	138+11833	1+11496	34	40	24	43	40	25
20	318	217+15129	1+14700	41	52	27	48	52	27
24	368	331+18328	1+17853	40	58	24	45	58	25
28	380	448+21808	1+21242	38	61	22	44	62	23
32	408	586+25530	1+24743	38	67	20	44	68	20
36	429	731+28108	1+27526	36	65	19	44	65	19
40	440	907+31382	1+30723	36	69	17	44	69	17

Figure 5.6: Search performance on Disk for different space dimensionalities (skewed data)

- *On disk:*

  – Both R*-tree and X-tree fail to outperform Sequential Scan due to the high numbers of accessed nodes. Again, the ratio of verified objects of X-tree is smaller than the one of Adaptive Clustering, but the overhead cost due to numerous random I/O accesses is much superior.

  – The selectivity of X-tree slightly improves with increasing dimensionality. This is due to the fact that as the space dimensionality increases fewer objects fit in each leaf node. Three times more leaf nodes are formed in 40 dimensions than in 16 dimensions. A data object is always placed in the leaf node that best accommodates it. Therefore, the objects are globally better grouped and the overlap between leaf nodes is minimized. This is not the case for R*-tree where object insertions are directed by directory nodes and data objects often end up in other leaf nodes than the most suitable ones. For X-tree, smaller page capacity (thus fewer objects per node) leads to better object grouping, but this obviously increases the number of I/O accesses. Indeed, while the number of nodes has tripled from 16 to 40

Table 5.2: Overlap degree between regions of interest

Overlap Probability [%]	Average Number of Overlapping Regions	Average Overlap Volume Ratio [%]
0.16	1.79	0.000162
0.8	9.47	0.000861
4	41.82	0.003012
20	202.50	0.013622
100	998.90	1.258195

dimensions, the query selectivity has only improved by 30%.

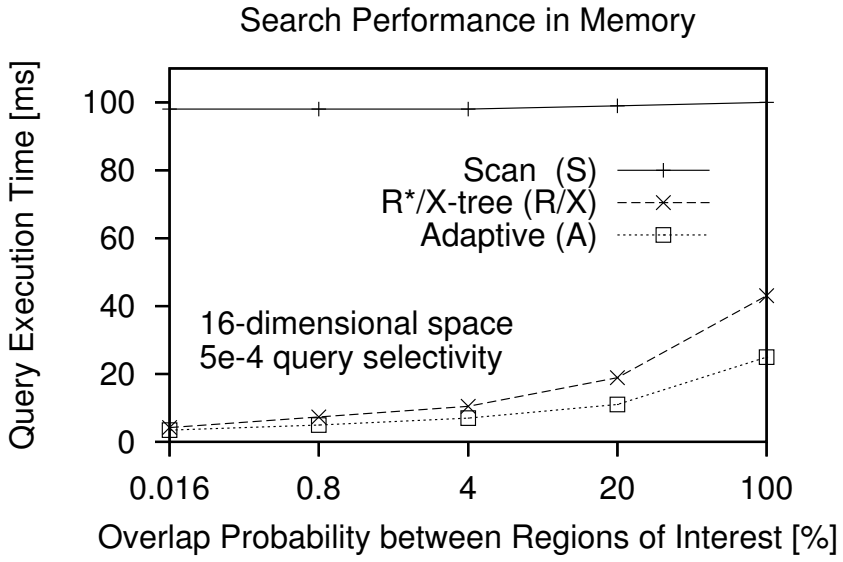
– In this experiment, Adaptive Clustering is 2 times faster than Sequential Scan, 3.4 times faster than X-tree, and between 15 and 33 times faster than R*-tree.

3. Regions of interest for data and query objects

In the previous experiment, we analyzed the search performance for data objects with skewness at dimension level. However, the data and the query objects were uniformly distributed in the multidimensional space. In practice, the data space is in general sparsely populated. Data and query objects tend to follow some regions of interest. Within the next experiment, we investigate the search performance for data and query objects with skewed spatial distributions, following a predefined set of regions of interest. By region of interest we understand a hyper-rectangular region of the multidimensional space, where objects and queries are constrained to fall.

Experiment 3 We randomly choose 1000 regions of interest in a 16-dimensional space. By imposing minimal and maximal interval length constraints over dimensions, we control the degree of overlap between the regions of interest. We consider 2000000 data objects uniformly populating the regions of interests: 2000 data objects are generated into each region of interest. The spatial queries are set to uniformly visit the regions of interest. Inside regions, both data and query objects are uniformly distributed. An average query selectivity of 0.05% is ensured at region level.

In this experiment, we vary the degree of overlap between regions of interest and examine the behavior of the four competing access methods. Table 5.2 shows the overlap degrees used in our tests. The first column indicates the probability that two arbitrary regions overlap each other. The second column complementary indicates the average number of regions that overlap with an arbitrary region. The third column shows the average overlapping volume between two overlapping regions, namely the ratio between the volume of the intersection zone and the total volume occupied by the two overlapping regions in the data space. The overlap probability increases each time by a factor of 5 from 0.16% to 100%. The average overlapping volume accordingly increases from 0.0002% and 1.2582%. Performance results are illustrated in Figure 5.7 for the memory storage scenario and in Figure 5.8 for the disk storage scenario.



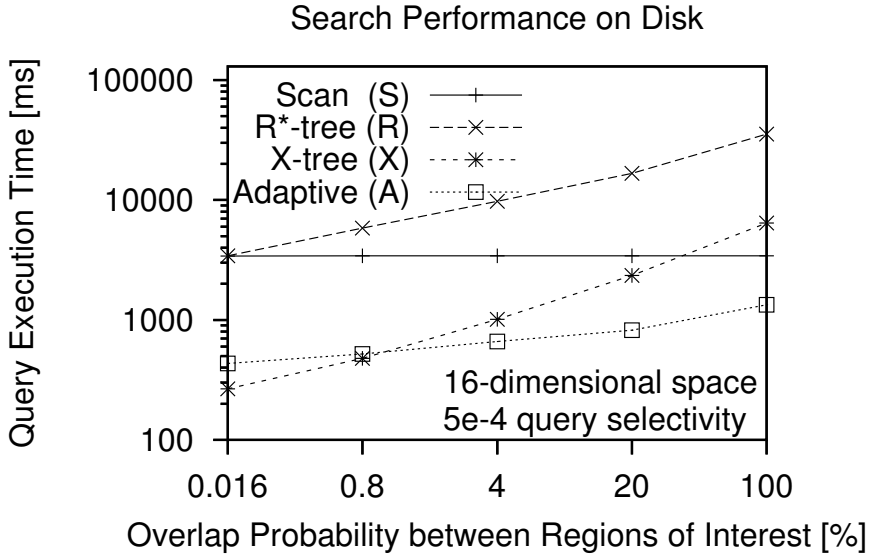
Memory Database Storage - Data Organization and Data Access						
Overlap	**Clusters**	**Nodes (dirs.+leaves)**	**Explored %**		**Verif. Objs. %**	
Prob.	**Adaptive**	**R*-tree/X-tree**	**A**	**R/X**	**A**	**R/X**
0.016	3795	264+22985	6	5	4	5
0.8	3200	266+22904	7	8	5	8
4	3671	264+22853	9	12	7	12
20	4443	268+22845	11	21	9	21
100%	7485	267+23073	19	45	20	45

Figure 5.7: Search performance in Memory with regions of interest (spatial data and query skewness)

Experimental facts and remarks:

- As expected, increasing overlap between regions of interest has negative impact on the search performance: the average query time increases for both Adaptive Clustering and R*-tree in memory and on disk, as well as for X-tree on disk.
- Adaptive Clustering significantly outperforms Sequential Scan: in memory by a factor ranging between 4, at high overlap, and 28, at low overlap, and on disk by a factor ranging between 2.5 and 8. This demonstrates that Adaptive Clustering can handle skewed data/query distributions, succeeding to cluster the data objects according to the most selective intervals/dimensions with respect to the regions of interest.
- *In memory:*

  – R*-tree also outperforms Sequential Scan. When the overlap probability is low, the regions of interest are well separated and R*-tree succeeds to preserve the region separation when grouping the data objects into nodes. As a result, the queries are well directed to the corresponding nodes and the search performance is high. In



Disk Database Storage - Data Organization and Data Access									
Overlap	**Clusters**	**Nodes (Dirs.+Leaves)**		**Explored %**			**Verif. Objs. %**		
Prob.	**Adaptive**	**R*-tree**	**X-tree**	**A**	**R**	**X**	**A**	**R**	**X**
0.016	598	264+22985	1+24085	6	5	0.4	9	5	0.4
0.8	542	266+22904	1+24164	7	8	0.7	11	8	0.7
4	474	264+22853	1+23970	9	12	2	13	12	2
20	403	268+22845	1+23727	12	21	4	17	21	4
100%	265	267+23073	1+22254	19	45	27	28	45	27

Figure 5.8: Search performance on Disk with regions of interest (spatial data and query skewness)

contrast, when the overlap probability is elevated, the region separation is not well captured, and a large number of nodes are explored, leading to the degradation of the search performance.

– Adaptive Clustering creates fewer clusters than R*-tree nodes and explores/verifies smaller ratios of clusters/objects. As a result, Adaptive Clustering shows better performance than R*-tree in all the cases. Adaptive Clustering is visibly less affected than R*-tree by the increasing overlap. The performance gap between the two indexing methods increases for larger overlap degrees.

- *On disk:*

  – Adaptive Clustering adapts the object grouping to the system's performance characteristics and shows a clear advantage compared to Sequential Scan. In contrast, R*-tree fails to outperform Sequential Scan, suffering from numerous I/O accesses.

  – Compared to R*-tree, X-tree manages to better arrange the data objects in nodes, preserving the region separation and ensuring good search performance, notably when the overlap between regions of interest is low. When the overlap

probability is less than 25%, the very low ratios of accessed nodes (see them in the table from Figure 5.8) ensure for X-tree better performance than Sequential Scan. When the overlap degree is very low ($< 0.8\%$), X-tree also outperforms Adaptive Clustering. However, in practice the regions of interests are not so well separated (overlap probability $< 0.8\%$). For more realistic overlap degrees ($> 0.8\%$), Adaptive Clustering demonstrates better search performance than X-tree.

5.2.3 Behavior and adaptability study

The objective of the next experiments is to study the behavior of our clustering method and its ability to adapt to dynamical changes in query and data distributions. Several aspects are considered such as evolution of search performance, evolution of number of clusters, time required to accomplish the database clustering, costs of cluster restructuring operations, and costs of database update operations.

4. Dynamically changing query selectivity

In this experiment, we employ uniform data and queries, as in the first experiment from the previous section. We dynamically change the query selectivity and observe the evolution of the search performance of Adaptive Clustering, comparing it to the three other competing methods.

Experiment 4 We consider 2000000 objects uniformly distributed in 16-dimensional space, and launch 100 samples of intersection queries with 100 queries per sample. The queries are uniformly distributed in the data space, but we impose minimal and maximal length constraints on the query intervals in order to control the average query selectivity. The average query selectivity is changed every 25 query samples, taking the following values: 5e-5, 5e-4, 5e-6, 5e-3. After each query sample, we trigger a global cluster restructuring operation. Performance results, averaged per sample, are depicted in Figure 5.9 for the memory storage scenario, and in Figure 5.10 for the disk storage scenario. In the upper charts from the two figures, we illustrate the evolution of: the average query cost of Sequential Scan (*S search*), the average query costs of R*-tree and X-tree (*R/X search* in memory, *R search* and *X search* on disk), the average query cost of Adaptive Clustering (*A search*), and the average query cost of Adaptive Clustering combined with the cost of cluster restructuring operations (*A search + clustering*). The cost of cluster restructuring operations is averaged between the 100 queries of a query sample. All these costs are expressed in terms of total execution time. In addition to the upper charts, we provide two auxiliary charts showing for Adaptive Clustering the evolution of the numbers of clusters in the storage cases. We accordingly represent the total number of clusters, the number of new clusters, and the number of merged clusters, raised after each global cluster restructuring operation.

Experimental facts and remarks:

- Adaptive Clustering reaches an efficient clustering configuration in less then 10 global cluster restructuring steps, corresponding to the first 10 query samples (1000

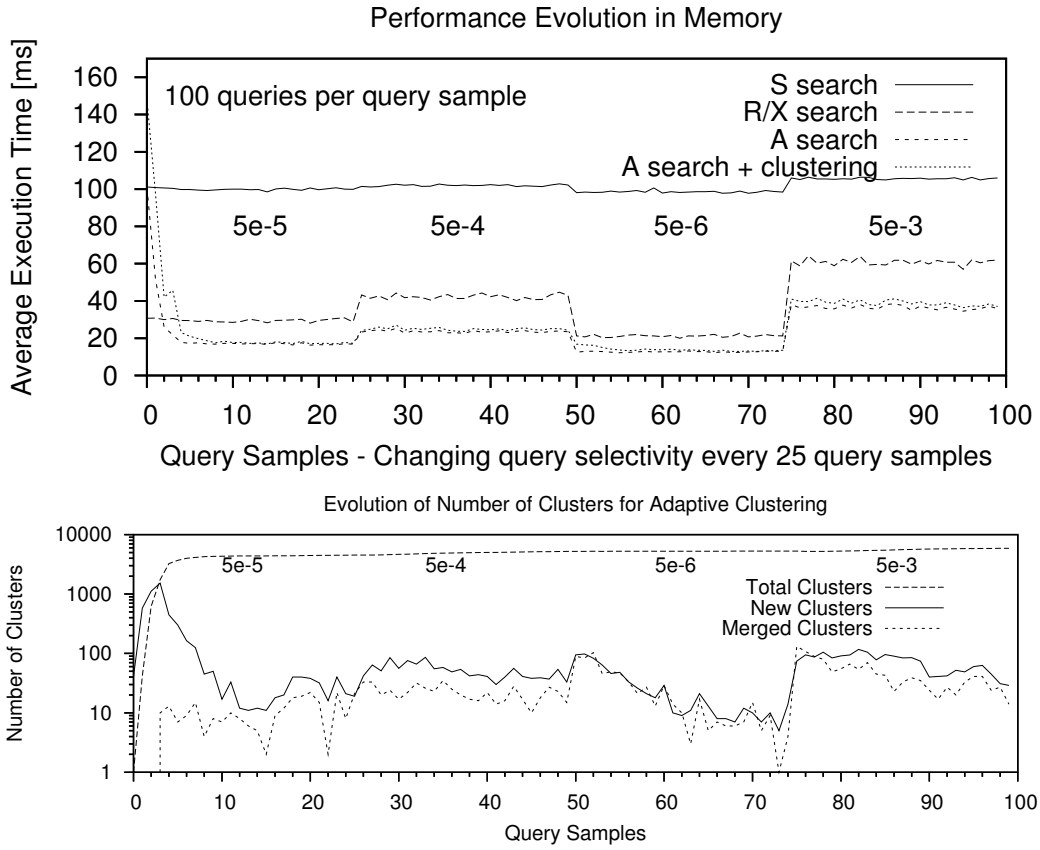


Figure 5.9: Performance evolution in Memory when dynamically changing query selectivity

queries), both in memory and on disk. During the first two query samples, the combined search and clustering cost exceeds the search cost of Sequential Scan. This happens due to the fact that the first cluster restructuring operations involve large sets of objects. Indeed, at the beginning, all the data objects are stored in a single cluster, the root. Because many clusters are rapidly created (see the charts showing the evolution of the number of clusters), the initial clustering cost cannot be immediately counterbalanced by the gain in the search performance. However, starting with the third query sample, the number and the cost of the clustering operations decrease significantly. As a result, the combined search and clustering cost of Adaptive Clustering drops and outperforms the search cost of Sequential Scan. During few more query samples, the search performance continues to improve, until it reaches a minimum, stable level. Similar behaviors can be noticed in memory and on disk. In memory, where R*-tree is globally more efficient than Sequential Scan, the combined search and clustering cost of Adaptive Clustering outperforms the search cost of R*-tree starting with the forth query sample.

- During the query samples following an important change in the average query selectivity (which in our experiment happens every 25 query samples), a higher than usual number of cluster restructuring operations are automatically triggered in a gradual manner (see the charts showing the evolution of the number of clusters).

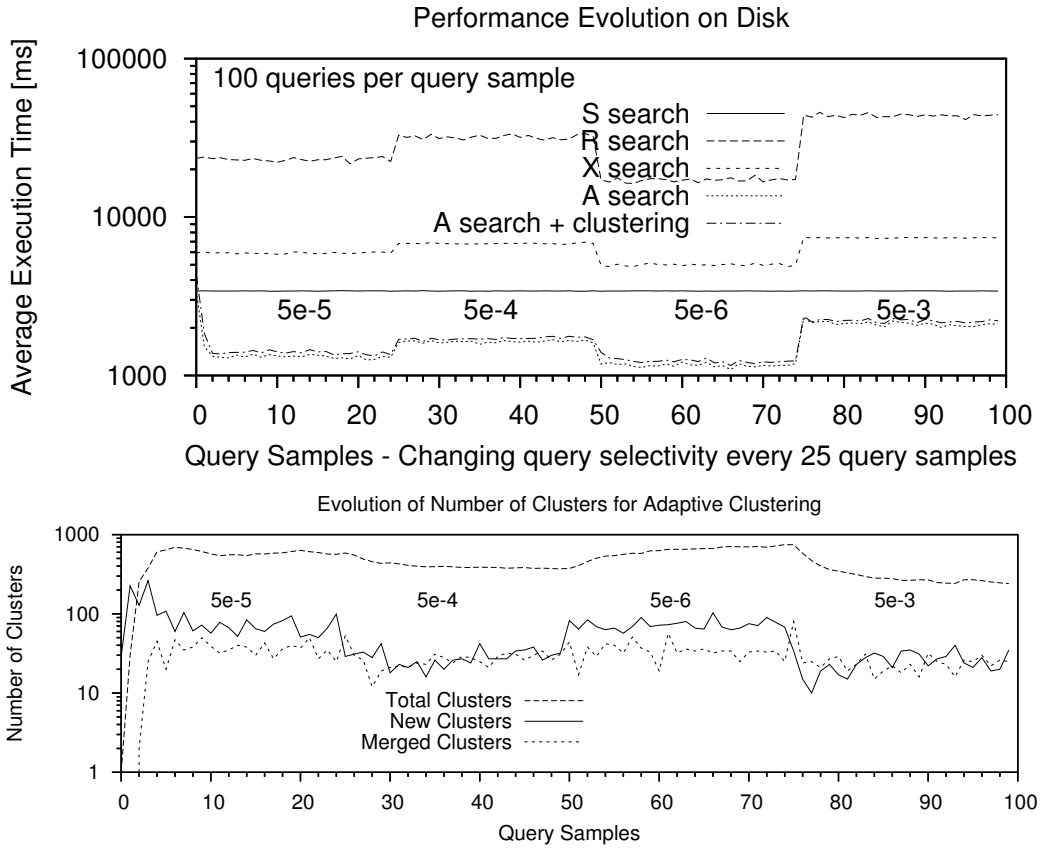


Figure 5.10: Performance evolution on Disk when dynamically changing query selectivity

Indeed, our cost-based query-adaptive clustering method attempts to adapt the clustering configuration to the new query characteristics. This induces a slight increase in the combined search and clustering cost, but the search performance is not significantly affected, continuing to preserve a clear advantage over the competing methods. In time, the additional cost overhead induced by the cluster restructuring operations tends to decrease, as well as the search cost, until a new stable clustering configuration is reached.

- Regarding the evolution of the number of clusters, we notice that in memory the total number of clusters globally increases, even though higher numbers of merges and splits are triggered after each important change in the average query selectivity. Indeed, the cost of accessing a cluster is quite low in memory, which explains why the total number of clusters slightly increases, even when the query selectivity decreases. In contrast, on disk, accessing a cluster is very expensive. For this reason, the total number of clusters follows the query selectivity, increasing when the query selectivity is higher (more cluster splits are triggered than cluster merges), and decreasing when the query selectivity is lower (more cluster merges are triggered than cluster splits).

By this behavior, Adaptive Clustering demonstrates good adaptability to changes in query patterns, while properly taking into account the platform-specific performance

Table 5.3: Cost comparison for different data organization methods

	Adaptive Clustering		Construction Time		
Time	**After 10 samples**	**Total**	**Sequential**	**R*-tree**	**X-tree**
In Memory	18 sec	34 sec	3 sec	491 sec	-
On Disk	4 min	18 min	1 min	93 min	303 min

characteristics.

We also remark that the time required to obtain an efficient clustering configuration is small, and the cost of cluster restructuring operations (cluster splits and cluster merges) is reduced, not significantly affecting the search performance. Table 5.3 shows the clustering time cost of Adaptive Clustering after the first 10 global cluster restructuring operations, as well as the total clustering time cost for all the 100 global cluster restructuring operations triggered in our experiment, and compares them to the insertion and construction times of Sequential Scan, R*-tree and X-tree, in memory and on disk. Note that the time cost to obtain an optimal clustering (after 10 global cluster restructuring operations) is much lower than the construction time of R*-tree or X-tree, in both storage cases.

5. Dynamically changing data distribution

Within this experiment, we investigate the behavior of our clustering approach to important changes in the data distribution, and also examine the cost of database update operations like object insertions and object deletions.

Experiment 5 We consider a number of 1000 regions of interest arbitrarily chosen in a 16-dimensional space. Minimal and maximal interval size constraints are imposed to the regions of interest in order to ensure 4% overlapping probability between any two arbitrary regions. We start with an initial dataset of 2000000 spatial objects uniformly populating only half of the regions of interest. So the first object distribution consists of 500 regions of interest, each one containing 4000 spatial objects. In this experiment, we launch 140 samples of 100 queries, interleaved with database update operations. Each database update operation deletes 20000 objects from the first distribution and inserts 20000 new objects falling in the other 500 regions and representing the second object distribution. The queries uniformly visit all the 1000 regions of interest. Each query sample is followed by a global cluster restructuring operation. The experiment is performed as follows: We first launch 20 query samples in order to produce a first clustering configuration (the first 20 query samples are not interleaved with database update operations). Starting with the 21th query sample, each query sample is followed by a database update operation. The data objects from the first distribution are gradually replaced by data objects from the second distribution. After 100 query samples, all the data objects from the initial distribution are replaced, so the update operations stop. Finally, 20 more query samples are launched to observe the search performance under the new object configuration.

The search performance and the update performance are illustrated in Figure 5.11 for the memory storage scenario, and in Figure 5.12 for the disk storage scenario. In the two upper charts we depict the average search time per query, the combined search and

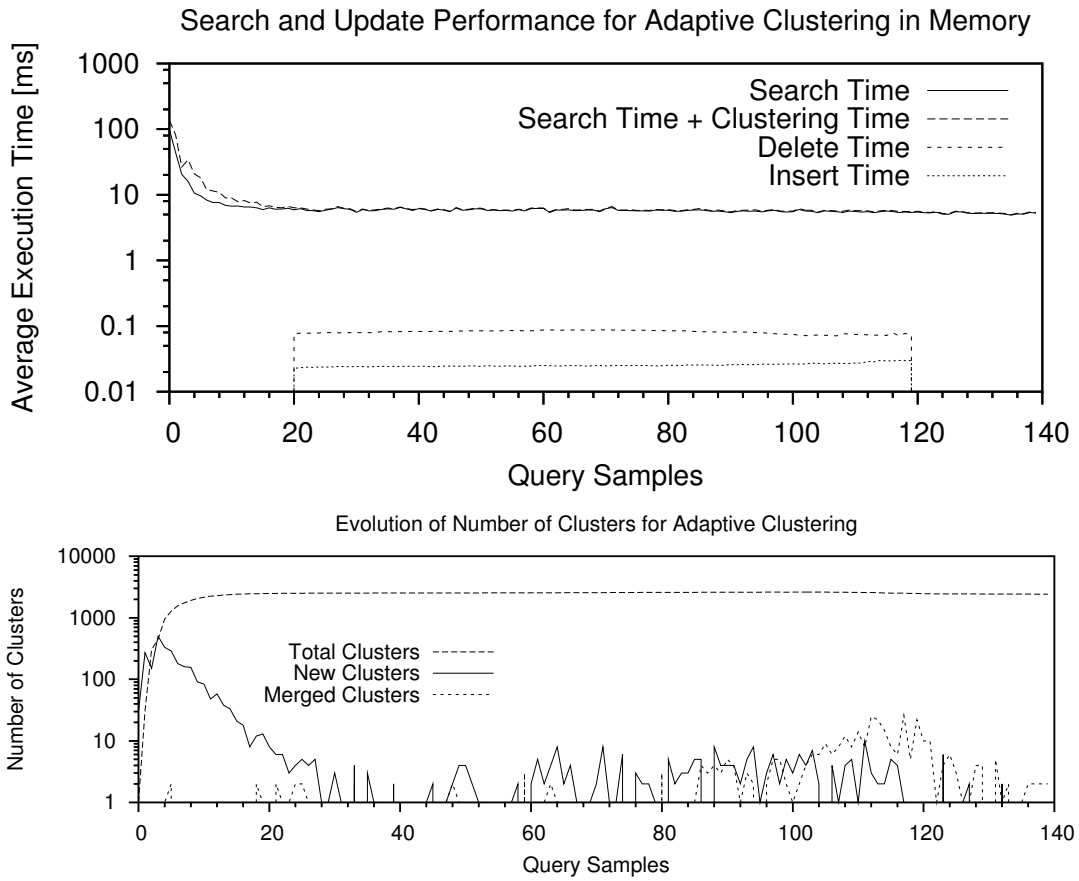


Figure 5.11: Behavior of Adaptive Clustering in Memory when dynamically changing data distribution

clustering time per query, the deletion time per object, and the insertion time per object. In the lower charts from the two figures we depict the evolution of the number of clusters in the two storage cases. Note that we have logarithmic scales in all the charts.

Experimental facts and remarks:

- The search performance of Adaptive Clustering is not visibly affected by the changes from the data distribution, succeeding to preserve its advantage over Sequential Scan both in memory and on disk (16 times better in memory, and 7 times better on disk). This good behavior is due to our insertion strategy which places every new object in the cluster with the lowest access probability, thus taking advantage of the clustering configuration already in place. It is also due to our clustering strategy which gradually adapts the object clustering to the new data characteristics: Some clusters are merged and others are split (see the evolution of the number of clusters). The cost overhead caused by cluster restructuring operations keeps low and does not penalize the search performance in a significant manner. Indeed, the combined search and clustering cost is close to the search cost, in both storage scenarios.

- The database update operations are efficient because they do not involve complex

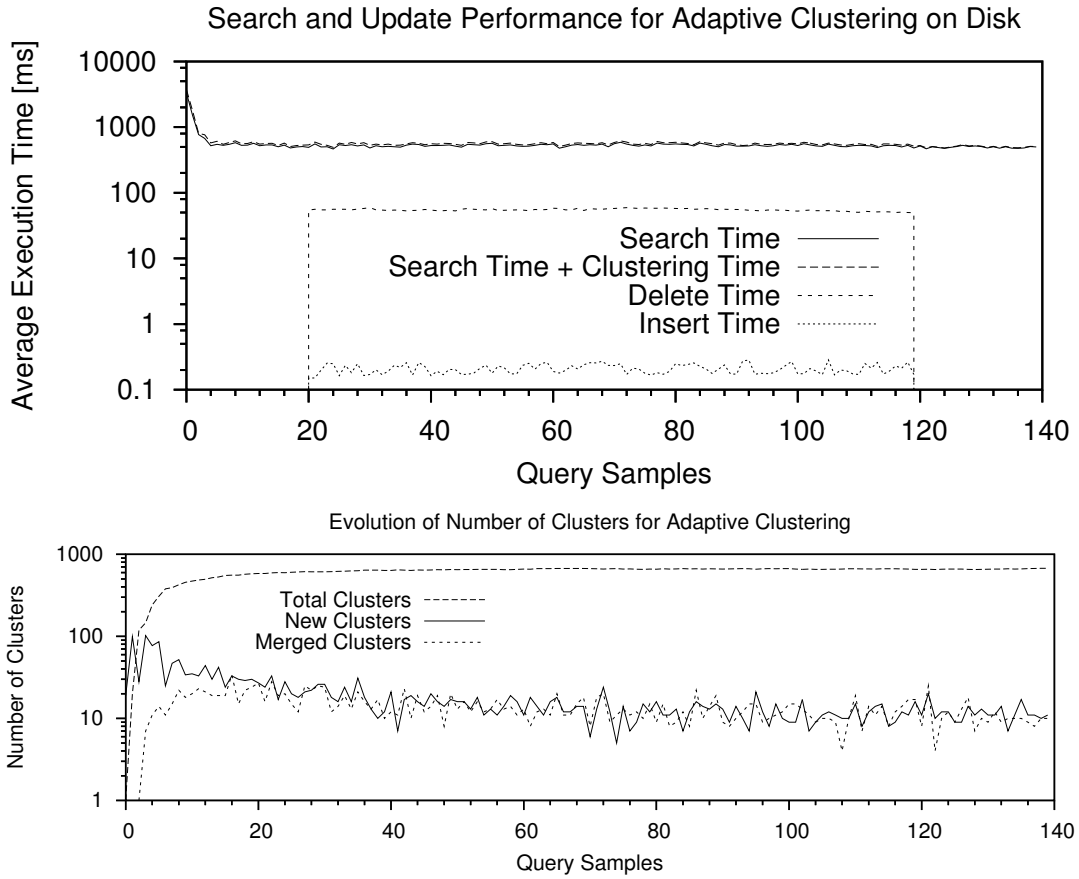


Figure 5.12: Behavior of Adaptive Clustering on Disk when dynamically changing data distribution

database restructuring operations:

– *Object insertions* are very fast. A new object is placed in the cluster with the lowest access probability, whose signature accepts the given object. The target cluster is selected based on cluster signatures and on query statistics maintained in memory for all the database clusters. One data write is only required to place the new object in the selected cluster, either in memory or on disk.

– *Object deletions* are faster than common spatial queries. An object deletion needs to explore a number of clusters in order to retrieve the wanted object. However, the object retrieval is faster than for common spatial queries because only the clusters whose signatures match exactly the wanted object need to be explored. In addition, the cluster explorations stop as soon as the cluster containing the wanted object is found. In contrast, spatial queries have to explore all the clusters whose signatures match the selection criterion, and also have to update the query statistics for all the explored clusters and candidate subclusters.

5.3 Conclusions

The practical relevance of our cost-based query-adaptive clustering solution was demonstrated in a series of experiments where we compared it to R*-tree, X-tree, and Sequential Scan. We examined the search performance and the behavior of our clustering method in main memory and on disk. We investigated the ability of our clustering approach to adapt to query and data distribution changes, and also analyzed the performance of object insertions and object deletions. The following experimental conclusions can be made.

- Regarding Adaptive Clustering:

  – Experimental results demonstrate that our cost-based query-adaptive clustering solution properly takes into account the performance parameters of the execution platform and succeeds to adapt the object clustering to the actual data and query distributions, showing better average search performance than Sequential Scan in all experiments, both in memory and on disk.

  – Our clustering method applies to large collections of spatial objects with many dimensions and has good behavior with increasing dimensionality. It efficiently handles data with skewness at dimension level (spatial objects with different selectivity characteristics over dimensions), and also data and queries with skewed distributions in the multidimensional space (following regions of interests).

  – The cost of maintaining query statistics for the explored clusters/candidate subclusters was counted as part of the query execution cost. Since the search performance of Adaptive Clustering is globally better than alternative solutions, we can conclude that maintaining query statistics is low-cost in terms of execution time and worth significant performance gains.

  – The cost of periodical cluster restructuring operations (cluster merges and cluster splits) is low, not significantly affecting the search performance.

  – The time required to obtain an optimal clustering configuration is much lower than the time required to build an R*-tree or an X-tree, both in memory and on disk.

  – Our clustering strategy proves to be efficient and able to adapt the object clustering to important changes occurring over time in query and data distributions.

  – The object insertions and the object deletions are fast because they do not involve complex restructuring operations.

- Comparing to R*-tree and X-tree:

  – Our object grouping is globally more efficient than the object grouping of R*-tree. Thanks to the cost-based clustering, Adaptive Clustering creates and explores fewer clusters than R*-tree, and verifies smaller amounts of data objects, resulting in better search performance, both in memory and on disk. On disk, due to the cost overhead of numerous I/O accesses, R*-tree fails to outperform Sequential Scan.

  – X-tree behaves like R*-tree in memory, while on disk it degenerates into a tree with only one supernode, the root. On disk, X-tree is more selective than R*-tree,

but considerably more expensive in terms of insertion time. The search performance of X-tree still suffers from numerous I/O accesses, being often outperformed by Sequential Scan. When X-tree shows better performance than Sequential Scan, Adaptive Clustering usually requires fewer random accesses and succeeds to outperform it in most cases.

Chapter 6

Conclusions and perspectives

6.1 Summary

In this thesis, we addressed the problem of efficiently indexing large and dynamic collections of multidimensional objects with spatial extents, to speed-up the execution of spatial range queries. As illustrated in Chapter 1, addressing this problem was motivated by the recent emergence of a wide range of applications from the SDI domain (Selective Dissemination of Information), bringing out new challenges for the multidimensional indexing schemes. In this context, an efficient multidimensional access method is needed to meet a number of general application requirements such as: *scalability* – manage large and dynamic collections of spatial objects with possibly many dimensions and with possibly large extents over dimensions, *search performance* – cope with high rates of spatial range queries, *update performance* – support frequent data object insertions and deletions, and *adaptability* – take into account the spatial distribution of data and query objects, dynamically accommodate important distribution changes in data and query patterns, and take into consideration the performance characteristics of the execution platform. At once meeting all these requirements represents a great challenge for the multidimensional indexing domain.

The state-of-the-art of the multidimensional indexing domain was reviewed in Chapter 2 with emphasis on the indexing solutions suitable for multidimensional objects with spatial extents. We focused on the evolution of the R-tree based methods, specifically designed to support spatial objects. We showed that a number of requirements specific to our target applications are highly impractical for classical R-trees (i.e., deal with many dimensions, handle objects with spatial extents, and cope with spatial range queries). Indeed, in high-dimensional spaces the overlap between minimum bounding rectangles is important at node level, especially when having data objects with spatial extents over dimensions. When answering spatial range queries an excessive number of nodes are explored, notably in a random manner. This inflicts high I/O costs and deteriorates the search performance so much that a naive approach like sequential scan becomes more efficient. To alleviate the performance deterioration, some R-tree extensions were proposed as trade-off solutions between random access and sequential scan. They use cost models embedding the performance characteristics of the execution platform and aiming to adapt the node page sizes to the data distribution (like in X-tree and DABS-tree) and to the query distribution (like in Adaptive R-tree). We analyzed the features and the

cost models of these latter techniques and emphasized their limitations. After studying the limitations of the existing multidimensional indexing solutions, we concluded that an efficient access method should have its object grouping assisted by a cost model embedding the performance parameters of the execution platform and taking into account both data and query distributions.

As the main contribution of our work, we proposed a cost-based query-adaptive clustering solution suitable for multidimensional objects with spatial extents and designed to improve the average performance of spatial range queries. Our object clustering drops many properties of classical tree-based indexing structures (i.e., tree height balance, balanced node splits, and minimum bounding in all dimensions) in favor of a cost-based *clustering strategy* meant to ensure for the set of clusters better average search performance than sequential scan in virtually every case. The *cost model* supporting our clustering strategy takes into account the platform's performance parameters and relies on both data and query distributions to assist the object clustering as means to improve the average performance of spatial range queries. The *object grouping criterion* supporting the object clustering consists of grouping spatial objects with similar intervals (positions and extents) in a reduced subset of dimensions, namely the most selective and discriminatory dimensions and domain regions relative to the query distribution. To identify the best grouping dimensions and domain regions, the grouping criterion is used to deterministically partition each cluster into a number of candidate subclusters representing future cluster candidates. Data and query statistics are maintained for all the clusters and candidate subclusters and employed in the cost model to support creation of new profitable clusters and removal of inefficient clusters. New clusters are created during cluster splits by materializing candidate subclusters that are expected to be profitable for the average search performance. Inefficient clusters are merged to the direct ancestors from the clustering hierarchy. Cluster restructuring operations are periodically triggered to accomplish and refine the object clustering and to adapt it to important changes that might occur over time in data and query distributions. The clustering strategy, the object grouping criterion and the cost model supporting the clustering decisions were detailed in Chapter 3.

Algorithms and execution procedures for the cluster restructuring operations (cluster restructuring invocation, cluster split, and cluster merge) and for standard database manipulation operations (spatial query execution, data object insertion and data object deletion) were presented and discussed in Chapter 4.

The practical relevance of our cost-based query-adaptive clustering solution was demonstrated in Chapter 5 where we presented an advanced experimental evaluation. Experimental results show that our clustering method properly takes into account the system's performance parameters and adapts the object clustering to the actual data and query distributions, showing better average search performance than Sequential Scan in memory and on disk. Our clustering method applies to large collections of spatial objects with many dimensions and has good behavior with increasing dimensionality. It efficiently handles data objects with skewness at dimension level, as well as data objects and queries with skewed distributions in the multidimensional space. The cost of maintaining query and data statistics is low and worth significant performance gains. Our clustering strategy is able to adapt the object clustering to important changes in query and data distributions. The cost of periodical cluster restructuring operations does

not significantly affect the system's availability. The database update operations are fast. Our object grouping is globally more efficient than the object grouping of R*-tree. Adaptive Clustering requires fewer random I/O accesses than X-tree on disk, outperforming it in most cases. The ability to take into account the performance characteristics of the execution platform and to adapt the object clustering to the real data and query distributions are the main advantages of our clustering method.

Parts of the work presented in this thesis were published in [SL04]. We envisage several interesting research issues related to this work:

6.2 Perspectives

- *Advanced control mechanism to trigger cluster restructuring operations*

  To improve the adaptability of our clustering method to data and query distribution changes and to minimize the overhead cost of cluster restructuring operations, we intend to integrate an advanced control mechanism able to monitor the average search performance and to detect significant performance changes, in order to trigger the cluster restructuring operations in a controlled manner, rather than on a periodical basis. For instance, when high fluctuations are detected in the system's performance, the cluster restructuring operations could be put on stand-by until the system's performance stabilizes. The control mechanism could be also used to reset the query statistics associated with clusters and with candidate subclusters when important changes are detected in the query patterns. Such actions would allow to avoid making wrong clustering decisions and to spare the corresponding cluster restructuring costs.

- *Alternative object grouping methods to support the clustering function*

  We plan to evaluate the interest and the maintenance cost of alternative grouping methods as support for the clustering function to deterministically partition a cluster into a known number of candidate subclusters. A possible alternative grouping method could be based on a space division according to several dimensions at once. Such a grouping method might be of interest if the number of possible subclusters remains acceptable. We also envisage to examine a domain partitioning adaptive to the data distribution at dimension level, with smaller size subregions in densely populated areas and higher size subregions in low populated areas. Such a grouping method might be beneficial if the cost of maintaining statistics for irregular space subregions is not too elevated.

- *Caching strategy to improve search performance for applications requiring disk-based storage*

  Because the search performance in memory is much faster than on disk, when dealing with applications requiring disk-based storage, a caching strategy could help to significantly improve the average query execution time. The query statistics that we maintain for clusters can serve to indicate the clusters more frequently accessed in order to cache them in main memory. We plan to develop an advanced cost model able to take into account both possible storage cases (memory and disk),

and thus to enable the implementation of an efficient caching strategy. An idea worth considering is the possible implementation of a dual clustering organization where several smaller clusters suitable for memory storage are grouped in larger clusters more appropriate for disk storage.

- *Approximation-based data representation model to accelerate spatial selections*

  Another research direction is related to the usage of an approximation-based data representation model to accelerate spatial selections in the manner of VA-File [WB97]. In [LS02] we presented an approximation-based filtering technique suitable for multidimensional objects with spatial extents. The domains of values of all dimensions are divided in subregions (possibly in a recursive manner). The interval limits of a spatial object are replaced by references to the qualifying domain subregions. When executing a spatial query, we first annotate the domain subregions that satisfy the selection criterion. Then we filter and only retain the objects pointing to qualifying subregions in all the dimensions. The real intervals of the qualifying objects are finally checked against the spatial selection criterion in order to eliminate the false positives. This approximation-based filtering approach is compatible with our clustering method. The two techniques could be combined to further accelerate the query execution, especially for main memory applications requiring high performance.

Chapter 7

Resumé en Français – French Summary

Groupement d'objets multidimensionnels étendus avec un modèle de coût adaptatif aux requêtes

7.1 Introduction

Le World Wide Web est devenu un énorme entrepôt de données dans pratiquement tous les domaines d'activité. Ces données sont de plus en plus facilement accessibles et disponibles grâce à la démocratisation de l'Internet haut débit. La forte croissance de l'Internet a entraîné une explosion du nombre d'utilisateurs et d'applications en ligne. Le volume important de données déployées sur le Web et le nombre élevé d'utilisateurs ont contribué au développement rapide d'une nouvelle classe d'applications : les applications de dissémination sélective d'informations (ou applications DSI). Le but de ces applications est de distribuer de l'information à un grand nombre de clients. Les entités qui produisent les informations sont appelées *producteurs* ou *publieurs*. Ces informations sont distribuées sous forme de paquets de données appelés *publications*. Les publications sont émises à des moments de temps appelés *événements de publication*. Les entités intéressées par ces publications sont appelées *consommateurs* ou *souscripteurs*. Ces derniers expriment leurs intérêts dans certaines publications au moyen de *souscriptions*. L'interaction entre les publieurs et les souscripteurs est assurée par une application spécifique appelée *système publieur-souscripteur*. Son objectif principal est de fournir rapidement les données appropriées aux souscripteurs concernés. Parmi les applications DSI les plus courantes, on notera celles liées aux bibliothèques électroniques de publications, à la bourse (suivie, échange et vente d'actions), aux enchères, aux systèmes de petites annonces, de publicité, et de forums. Des applications DSI plus complexes peuvent être déployées dans certains systèmes de gestion de réseaux distribués, systèmes de réseaux de capteurs sensoriels, systèmes d'alerte et de notification, systèmes d'intégration d'applications d'entreprise (EAI), systèmes de transactions financières sécurisées, systèmes de services, de recherche, de notification et de livraison de données.

7.1.1 Motivation du travail

Les applications émergentes liées à la dissémination sélective d'informations doivent souvent gérer de larges collections de souscriptions et des taux importants de nouvelles publications portant sur de nombreux attributs. Le caractère multidimensionnel des données manipulées dans ces applications, ainsi que les exigences de performance spécifiques, soulèvent de nouveaux défis dans le domaine de l'indexation multidimensionnelle. Ces défis représentent la principale motivation de notre travail.

Exemple d'application

Comme exemple d'application, nous allons considérer un système de notification de type "publieur-souscripteur" gérant de petites annonces. Par exemple, une demande de souscription peut être : "Notifiez-moi de toutes les offres d'appartements en location dans un rayon de 30 Km autour de Versailles, avec un loyer mensuel entre 800 et 1200 euros, ayant entre 3 et 5 pièces, et 2 salles de bains". La plupart des attributs de cette souscription portent sur des intervalles de valeurs plutôt que sur des valeurs exactes : 0-30 pour la distance de Versailles, 800-1200 pour le loyer, 3-5 pour le nombre de pièces. En pratique, les souscriptions de type intervalle sont plus intéressantes et mieux adaptées pour ce type d'applications. D'abord, parce que la probabilité est très faible de trouver des offres qui satisfont simultanément les valeurs exactes de tous les attributs. Ensuite, parce qu'il n'est pas toujours possible de spécifier des valeurs exactes pour tous les attributs. Si chercher un appartement avec un loyer donné n'est pas vraiment raisonnable, définir un intervalle de valeurs pour le loyer reste tout à fait pertinent. De manière générale, lorsqu'il n'y a aucune offre qui satisfait les critères de recherche exacts, les souscripteurs préfèrent consulter des offres alternatives, à condition que celles-ci soient suffisamment proches des valeurs exactes envisagées au départ. Grâce aux intervalles de valeurs, les souscripteurs peuvent définir des critères de recherche plus flexibles, pouvant établir un degré de variation acceptable au niveau de chaque attribut. De toute évidence, les souscriptions de type intervalle s'avèrent être très utiles pour les applications de dissémination sélective d'informations [LJ04]. Dans un système de notification de type "publieur-souscripteur", des taux importants de nouvelles publications (ou offres) émises par les publieurs ont besoin d'être comparées avec l'ensemble des souscriptions afin d'identifier celles visées pour notifier les souscripteurs concernés. De manière générale, les publications peuvent spécifier pour leurs attributs soit des valeurs exactes, soit des intervalles de valeurs. Par exemple, une publication avec des attributs spécifiant des valeurs exactes peut être énoncé comme suit : "Particulier - Appartement à louer, 10 Km de Versailles, 3 pièces, 1 salle de bains, 1100 euros par mois". Un autre exemple de publication avec des attributs portant sur des intervalles de valeurs peut être : "Agence immobilière - Appartements à louer sur Versailles : 3-5 pièces, 1-2 salles de bain, 1000-1300 euros par mois".

Modèle de représentation des données

Comme montré dans l'exemple précédent, les souscriptions et les publications portent souvent sur des attributs de type intervalle et peuvent être représentées comme des ensembles d'associations <attribut, intervalle de valeurs>. Chaque attribut peut être

vu comme une dimension différente. Dans cette représentation, les souscriptions et les publications deviennent des objets étendus dans un espace multidimensionnel défini par l'ensemble d'attributs. Un objet multidimensionnel étendu, également appelé hyper-intervalle ou hyper-rectangle, comporte un intervalle de valeurs pour chacune de ses dimensions. La collection de souscriptions forme une base d'objets multidimensionnels étendus sur laquelle les nouvelles publications agissent comme des requêtes spatiales de type intervalle : Les souscriptions visées par les nouvelles publications sont identifiées aux moyens d'opérations d'intersection, d'inclusion, ou de couverture spatiale, entre les objets-requêtes représentant les publications et les objets de la base de données représentant les souscriptions. Avec ce modèle de représentation des données, retrouver les souscriptions qui satisfont les nouvelles publications revient à répondre aux requêtes spatiales de type intervalle sur la base d'objets multidimensionnels étendus.

7.1.2 Exigences de performance

Dans le contexte des applications DSI, il est nécessaire d'avoir une méthode d'accès performante qui permet d'exécuter efficacement des requêtes spatiales de type intervalle sur de larges collections d'objets multidimensionnels étendus ; ceci dans le but de pouvoir trouver rapidement les souscriptions satisfaisant les nouvelles publications. Les exigences de la solution d'indexation qui en découlent sont les suivantes :

- Scalabilité

  Les applications DSI comportent souvent des collections larges et dynamiques de souscriptions (millions de souscriptions) portant sur des nombreux attributs (dizaines de dimensions). La méthode d'indexation adoptée doit pouvoir gérer un grand nombre d'objets multidimensionnels étendus avec beaucoup de dimensions et avec potentiellement de longues extensions dans les dimensions. De longues extensions peuvent être associées aux attributs pour lesquels les souscripteurs n'ont pas de préférences particulières.

- Performance de recherche

  Certaines applications de type DSI doivent traiter des centaines voire des milliers de publications par seconde. La réactivité du système peut être cruciale dans par exemple les systèmes d'alerte et de notification temps-réel. Afin de supporter des taux très élevés de nouvelles publications, la méthode d'indexation utilisée doit être efficace, c'est-à-dire capable d'exécuter rapidement des requêtes spatiales de type intervalle sur l'ensemble des objets multidimensionnels étendus représentant les souscriptions.

- Performance de mise à jour et adaptabilité

  Généralement, les souscriptions sont très dynamiques, les intérêts de souscripteurs varient beaucoup dans le temps, ainsi que les publications émises par les publieurs. Les insertions de nouvelles souscriptions et suppressions de souscriptions expirées peuvent intervenir fréquemment. Dans un tel contexte, la solution d'indexation doit pouvoir gérer des mises à jour fréquentes de la base de souscriptions. En effet, les souscripteurs ont souvent l'habitude de suivre certaines zones d'intérêt mais celles-ci ne sont pas forcément suivies par les offres. La méthode d'indexation doit

pouvoir accommoder les variations dynamiques de distribution des objets (suite aux insertions et aux suppressions de souscriptions) et des requêtes (suite aux changements de la nature des publications). Les mises à jour des souscriptions et les opérations de réorganisation de la base de souscriptions doivent être rapides, afin de ne pas affecter de façon significative la disponibilité et la performance de recherche du système.

7.1.3 Formulation du problème

Satisfaire les exigences de performance des applications DSI émergentes représente un important défi pour le domaine de l'indexation multidimensionnelle. La plupart des méthodes d'indexation multidimensionnelles supportant l'exécution de requêtes spatiales de type intervalle sur des collections d'objets multidimensionnels étendus sont dérivées du R-tree. R-tree est une technique d'indexation multidimensionnelle qui utilise des rectangles englobants minimaux (REMs) pour organiser les objets spatiaux dans un arbre hiérarchique afin de diriger les recherches vers les régions spatiales visées et ainsi améliorer l'exécution des requêtes. Plusieurs contraintes de construction sont imposées pour assurer l'équilibrage en hauteur de l'arbre et pour garantir un remplissage minimal des pages associées aux noeuds. Ces contraintes de construction, combinées avec l'aspect multidimensionnel, entraînent une superposition spatiale importante des REMs au niveau des noeuds. Cette superposition spatiale des REMs a pour conséquence de dégrader les performances lors des recherches. En effet, plus de branches de l'arbre devront être explorées, donc plus de noeuds devront être accédés, notamment dans le cas des requêtes spatiales de type intervalle. De plus, la probabilité de superposition spatiale des REMs augmente avec le nombre de dimensions [BKK96, BBK98b].

Au fil des années, plusieurs techniques ont été proposées pour diminuer les effets de ce phénomène connu sous le nom de "la malédiction de la dimensionnalité". Malgré ces efforts, plusieurs évaluations expérimentales [BBK01] ont démontré qu'au delà de 5-6 dimensions une recherche séquentielle naïve de l'ensemble d'objets s'avère plus efficace qu'une recherche basée sur des structures d'indexation complexes telles que les R*-tree, Hilbert R-tree ou X-tree [BBK98b, BK00, BBK01]. Ces faits ont déjà été constatés pour des requêtes de type intervalle sur des collections de points. La performance de recherche s'empire lorsque l'on considère des collections d'objets multidimensionnels étendus, car le degré de superposition spatiale des REMs augmente avec la superposition spatiale des objets.

La mauvaise performance de recherche dans les espaces avec beaucoup de dimensions est la principale limitation des méthodes basées sur la technique R-tree. Une autre limitation importante de ces méthodes est liée au coût des opérations de restructuration de la base d'objets lors des insertions et des suppressions d'objets. En effet, les changements fréquents d'objets entraînent des opérations de restructuration complexes et coûteuses. Vu les exigences des nouvelles applications DSI, il est évident que les méthodes d'indexation basées sur la technique R-tree ne sont pas appropriées. Dans ce contexte, l'objectif de notre travail a été d'étudier leurs limitations et de proposer une méthode d'accès efficace et adaptée pour les applications DSI émergentes.

7.1.4 Contributions

Dans cette thèse nous présentons une méthode de groupement en clusters des collections dynamiques d'objets multidimensionnels étendus, avec un modèle de coût adaptatif aux requêtes, dont l'objectif principal est d'améliorer la performance d'exécution des requêtes d'intersection, d'inclusion et de couverture spatiale. L'intérêt de notre solution de groupement est qu'elle garantit une bonne performance de recherche, meilleure que la recherche séquentielle dans tous les cas de figure, tout en tenant compte des exigences applicatives motivées auparavant : supporter de grandes ensembles d'objets multidimensionnels étendus, avec beaucoup de dimensions et avec de longues extensions possibles dans les dimensions, supporter des taux élevés de requêtes spatiales de type intervalle, des mises à jour fréquentes d'objets, des variations de distribution spatiale des données, ainsi que des requêtes. Les contributions les plus importantes de notre travail sont :

- Une stratégie de groupement basée sur un modèle de coût adaptatif aux requêtes

  Notre groupement d'objets en clusters est basé sur un modèle de coût qui prend en considération la distribution spatiale des objets et la distribution spatiale des requêtes. Le modèle de coût tient compte d'un ensemble de paramètres système influant sur la performance d'exécution des requêtes, i.e., le temps d'accès mémoire et disque, la vitesse de transfert des données entre le disque et la mémoire, la vitesse de vérification du critère de sélection sur les objets en mémoire. Au niveau de chaque cluster d'objets on maintient des statistiques concernant la distribution locale des objets et des requêtes. Ces statistiques sont combinées avec les paramètres système afin de soutenir les décisions de création de nouveaux clusters, plus profitables, et de suppression de clusters qui ne sont plus rentables. Le but de notre stratégie de groupement est de minimiser les coûts d'accès et d'exploration des clusters. Le modèle de coût employé est flexible et peut être facilement adapté à différents scénarios de stockage, assurant une bonne performance de recherche, toujours meilleure que la recherche séquentielle, sur disque ou en mémoire.

- Un critère original pour le groupement spatial d'objets

  Notre groupement d'objets en clusters est basé sur une nouvelle approche de groupement spatial : Un cluster regroupe des objets qui présentent des intervalles "similaires" dans un sous-ensemble de dimensions. Par intervalles "similaires" nous entendons des intervalles de tailles comparables et localisés dans les mêmes régions de l'espace. La notion de similarité est définie par rapport à un découpage régulier et récursif de l'espace en sous-régions, au niveau de chaque dimension. Les dimensions de regroupement et les sous-régions spatiales sont choisies en fonction de la distribution des requêtes afin de minimiser la probabilité d'accès aux clusters au cours des recherches. Pour les objets étendus, ce critère de groupement spatial avec notre stratégie de groupement, s'avère plus efficace que les approches de groupement traditionnelles, la plupart basées sur un englobement minimal dans toutes les dimensions - comme dans le cas des R-trees. Grâce à notre groupement spatial, moins de clusters sont crées et explorés, et moins d'objets sont vérifiés lors des recherches spatiales.

Evaluation expérimentale

Pour évaluer l'efficacité de notre méthode de groupement, nous avons conçu et exécuté un large spectre d'expériences, comportant des objets et des requêtes uniformes et non uniformes, en mémoire et sur disque. Notre solution de groupement a été testé pour des objets et des requêtes uniformément reparties dans l'espace multidimensionnel, pour des objets avec une partie de dimensions plus sélectives que d'autres, et aussi pour des objets et des requêtes distribuées de façon non uniforme dans l'espace multidimensionnel (objets et requêtes reparties dans des régions d'intérêt). Nous avons comparé notre technique avec la recherche séquentielle, avec la technique d'indexation R*-tree, et avec la méthode d'indexation X-tree, pour de larges ensembles d'objets avec un nombre variable de dimensions et stockés en mémoire et sur disque. Notre approche de groupement, basée sur un modèle de coût adaptatif aux requêtes, s'est avérée plus efficace, démontrant de bonnes performances de recherche, de faibles coûts de mise à jours de la base d'objets, et une bonne adaptabilité aux variations de distribution des objets et des requêtes dans tous les cas de figure, notamment dans les cas où le R*-tree et le X-tree ont échoué face à la recherche séquentielle.

7.1.5 Organisation de la thèse

La thèse est organisée en six chapitres de la façon suivante :

Chapitre 1 introduit cette thèse. Il expose d'abord la motivation de notre travail, le type d'applications visées avec leurs caractéristiques et leurs exigences. Ensuite, il introduit le problème adressé et les défis relevés. En conclusion, il mentionne les principales contributions de notre thèse.

Chapitre 2 donne une définition formelle du problème adressé et présente l'état de l'art. Nous montrons que le problème adressé s'inscrit dans le domaine de l'indexation multidimensionnelle. Les caractéristiques et les exigences des applications visées représentent un vrai défi dans ce domaine. Dans ce contexte, les techniques d'indexation existantes, applicables au problème adressé sont présentées, étudiées et leurs limitations révélées. Les principes et les idées directrices de notre solution alternative sont présentés en conclusion de ce chapitre.

Chapitre 3 présente les principaux éléments de la solution de groupement en clusters que nous proposons : la stratégie de groupement, le critère de groupement spatial des objets, et le modèle de coût employé pour assister les décisions de partitionnement. Premièrement, nous expliquons la stratégie de groupement et le processus de partitionnement en clusters. Ensuite, nous présentons le critère de groupement spatial des objets rendant possible l'implémentation de la stratégie de groupement. Enfin, nous détaillons le modèle de coût supportant les décisions de création de nouveaux clusters profitables et de suppression des clusters qui ne sont plus rentables.

Chapitre 4 présente l'implémentation et les algorithmes d'exécution des principales opérations d'organisation et de manipulation de la base d'objets. Le coût et la complexité de ces opérations sont adressés. Premièrement, nous considérons les opérations d'organisation et de restructuration invoquées lors du partitionnement de l'ensemble d'objets : les éclatements et les fusions des clusters. Ensuite, nous présentons et nous étudions les algorithmes d'exécution des opérations de manipulation classiques : exécution

des requêtes spatiales (intersections, inclusions et couvertures spatiales) et opérations de mise à jour (insertions et suppressions d'objets).

Chapitre 5 adresse un nombre d'aspects liés à l'implémentation de la solution proposée et présente une étude expérimentale de performance. Premièrement, nous présentons les structures de données utilisées pour implémenter notre technique de groupement, la gestion de la mémoire centrale et la gestion de l'espace de stockage. Ensuite, nous présentons une série d'expériences spécialement conçues pour évaluer la performance et l'efficacité de notre méthode de groupement, en la comparant avec plusieurs solutions alternatives. Nous démontrons expérimentalement que notre technique est plus efficace que les autres solutions, pour différents jeux de données et de requêtes et pour différents scenarios de stockage.

Chapitre 6 résume les principales contributions de notre travail de thèse, tire des conclusions et introduit plusieurs perspectives de recherches futures.

7.2 Problème et état de l'art

Dans ce chapitre, nous définissons le problème adressé de façon formelle. Puis nous présentons l'état de l'art. Nous nous sommes notamment intéressés aux solutions existantes applicables à ce problème. En conclusion de ce chapitre, nous introduisons les principes et les idées directrices de notre solution alternative.

7.2.1 Spécification du problème

Le problème que nous adressons tient du domaine de l'indexation multidimensionnelle. Le problème consiste à indexer des collections larges et dynamiques d'objets multidimensionnels étendus, avec beaucoup de dimensions et avec potentiellement de longues extensions dans les dimensions, afin d'exécuter efficacement des requêtes spatiales de type intervalle : intersections, inclusions et couvertures spatiales. Un nombre d'exigences applicatives doivent être satisfaites : scalabilité, performance de recherche, performance de mise à jour, adaptabilité aux variations des objets et des requêtes, et prise en compte des paramètres système. Satisfaire simultanément toutes ces exigences représente un défi important pour le domaine de l'indexation multidimensionnelle.

Nous passons en revue l'état de l'art du domaine de l'indexation multidimensionnelle. Pour cela, nous présentons les principales caractéristiques et limitations des solutions d'indexation existantes, insistant sur les techniques applicables aux objets multidimensionnels avec des extensions spatiales. En particulier, nous nous intéressons à l'évolution des méthodes d'indexation basées sur la technique R-tree, spécialement conçue pour gérer des objets multidimensionnels étendus. Dans ce contexte, nous montrons qu'une partie importante d'exigences imposées par les applications visées ne peuvent pas être assurées par les méthodes de la famille R-tree, i.e., supporter un nombre élevé de dimensions, gérer des objets avec potentiellement de longues extensions dans les dimensions, et répondre efficacement aux requêtes spatiales de type intervalle.

En effet, ces exigences applicatives entraînent une superposition spatiale trop importante des rectangles englobants minimaux. Par conséquent, un nombre excessif de noeuds sont explorés lors des sélections spatiales. Les noeuds étant explorés par des accès

aléatoires, les coûts d'entrée/sortie sont très élevés et induisent une forte dégradation de la performance de recherche. Dans beaucoup de cas, la performance de recherche peut s'avérer moins efficace qu'une recherche séquentielle. Pour palier à ce problème, des améliorations des R-trees ont été proposées, elles offrent des compromis entre l'accès aléatoire et l'accès séquentiel. Ces améliorations utilisent des modèles de coût qui prennent en compte les caractéristiques de performance de la plateforme d'exécution. La taille des pages associées aux noeuds peut dépendre, soit de la distribution spatiale des objets (X-tree [BKK96] et DABS-tree [BK00]), ou soit de la distribution spatiale des objets et des requêtes (Adaptive R-trees [TP02]).

7.2.2 Etat de l'art

Durant les trente dernières années, de nombreuses techniques d'indexation ont été proposées pour améliorer la performance d'exécution des requêtes spatiales sur de larges collections d'objets multidimensionnels. [GG98, BBK01, RSV01, Yu02, MNPT03] examinent et comparent la plupart des méthodes d'accès multidimensionnelles existantes. Selon ces études, on peut distinguer deux familles de solutions : les solutions basées sur le partitionnement disjoint de l'espace, et les solutions basées sur une organisation hiérarchique des objets.

Les solutions d'indexation de la première famille s'appuient sur un partitionnement récursif de l'espace dans des régions disjointes. Le partitionnement de l'espace est effectué selon une seule ou plusieurs dimensions à la fois. Quad-tree [FB74, Sam84], K-D-tree [Ben75, BF79], K-D-B-tree [Rob81], Grid file [NHS84] et hB-tree [LS90] utilisent ce partitionnement. Certaines solutions hybrides telles que Pyramid-tree [BBK98b], VA-File [WSB98], et la méthode i-Distance [JOT$^+$05], combinent le partitionnement disjoint de l'espace avec des transformations spatiales de représentation. En général, les méthodes basées sur un partitionnement disjoint de l'espace sont utilisées pour indexer des collections de points multidimensionnels. En effet, les points peuvent facilement être séparés dans des régions disjointes. Par contre, ces méthodes ne sont pas adaptées pour les objets étendus car ces objets se superposent dans l'espace. Il n'est donc pas possible de les séparer de manière équilibrée dans des régions bien disjointes.

La seconde famille permet d'indexer des collections d'objets multidimensionnels étendus. Elle est basée sur la technique R-tree qui a été proposée dans [Gut84] comme une méthode d'accès adaptée aux objets spatiaux étendus. Cette technique conçue comme étant une généralisation multidimensionnelle de B-Tree a été initialement utilisée pour indexer des rectangles 2-d. Elle emploie des rectangles englobants minimaux (REMs), en général des hyper-rectangles pour hiérarchiser les objets spatiaux dans un arbre équilibré en hauteur. Chaque noeud de l'arbre correspond à une page mémoire/disque ayant une taille d'un ou de plusieurs blocs E/S. Une entrée dans un noeud-feuille contient un identificateur d'objet et un rectangle minimal englobant l'objet correspondant. Une entrée dans un noeud intermédiaire (noeud directeur) contient un pointeur vers un noeud fils et un rectangle englobant minimal couvrant de façon minimale tous les REMs situés plus bas dans le sous-arbre correspondant. Des contraintes de construction sont imposées d'une part pour garantir et préserver l'équilibrage en hauteur de l'arbre, c'est-à-dire que tous les noeuds feuilles apparaissent à la même hauteur dans l'arbre, et d'autre part pour assurer un remplissage minimal des pages mémoire/disque associées aux noeuds, c'est-à-dire un partitionnement équilibré des noeuds. Avec l'aspect multidimensionnel,

ces contraintes de construction entraînent une superposition spatiale significative entre les différents REMs au niveau des noeuds. Pour effectuer une recherche, il faut alors explorer de nombreuses branches de l'arbre, ce qui entraîne une sérieuse dégradation de performance. Différentes stratégies ont été proposées pour réduire les effets de ce phénomène : Packed R-tree [RL85], R+-tree [SRF87], et R*-tree [BKSS90].

Packed R-tree La méthode de construction utilisée par Packed R-tree [RL85] permet l'obtention d'une hiérarchie arborescente plus efficace avec une meilleure performance de recherche. Au lieu d'être agrandie graduellement au moyen d'insertions successives d'objets, la structure Packed R-tree est construite de bas en haut en partant de l'ensemble complet d'objets. Evidemment, cette méthode de construction ne fonctionne uniquement que sur des collections statiques d'objets, connues et disponibles à l'avance.

R+-tree R+-tree [SRF87] évite complètement la superposition spatiale et permet aux REMs de découper les objets étendus en plusieurs morceaux. Lorsque le partitionnement d'un noeud nécessite le découpage d'un objet, l'objet correspondant est divisé en deux morceaux. Chaque morceau préserve l'identificateur de l'objet original et est inséré dans le REM qui lui correspond. La superposition spatiale des REMs est ainsi évitée mais la hauteur de l'arbre est plus élevée. R+-tree est plus performant que R-tree dans les espaces avec peu de dimensions. Cependant, dans les espaces de dimensions élevées, R+-tree est peu efficace à cause du nombre important de découpages et réplications d'objets.

R*-tree R*-tree [BKSS90] apporte de nouvelles stratégies et heuristiques de partitionnement pour diminuer la superposition spatiale des REMs. Pour mieux adapter la structure arborescente aux insertions successives, R*-tree introduit le concept de réinsertions forcées. Une partie des objets des noeuds surpeuplés sont extraits et réinsérées dans la structure arborescente. Les réinsertions sont ainsi préférées aux partitionnements de noeuds. Par rapport aux méthodes précédentes, R*-tree s'est révélé être plus performant et s'est rapidement imposé comme une référence de comparaison pour les solutions d'indexation multidimensionnelles.

Considérations de performance

Les R-trees classiques, supportant des objets multidimensionnels étendus, sont en général plus efficaces que la recherche séquentielle dans les espaces avec peu de dimensions. Cependant, ils le sont beaucoup moins dans des espaces à plus de 5-6 dimensions, en raison de la superposition spatiale des rectangles englobants minimaux. Cette superposition augmente avec la dimensionnalité de l'espace, phénomène connu sous le nom de "la malédiction de la dimensionnalité" [BKK96, BBK98b]. A cause des superpositions spatiales des REMs, un grand nombre de noeuds/pages sont accédées lors des recherches spatiales. Tous les types de requêtes sont affectés, notamment les requêtes de type intervalle qui, par définition, explorent des régions vastes de l'espace. Les accès disque sont aléatoires et entraînent des opérations d'entrée/sortie très coûteuses comme les repositionnements de la tête de lecture/écriture. Les accès aléatoires ne peuvent pas bénéficier

de vitesses de transfert élevées entre le disque et la mémoire comme c'est le cas des lectures séquentielles. Les lectures séquentielles sont beaucoup plus rapides, et la recherche séquentielle s'avère souvent plus efficace.

Dans ce contexte, VA-File [WSB98] s'est imposé comme une solution de recherche alternative, plus efficace que les structures d'indexation arborescentes, notamment dans les espaces avec beaucoup de dimensions. VA-File utilise une représentation approximée des données et tire profit de la recherche séquentielle pour accélérer la performance de recherche sur de larges collections de points multidimensionnels. Cependant, cette méthode est conçue pour gérer uniquement des ensembles de points multidimensionnels.

Comme solutions de compromis entre les accès aléatoires et la recherche séquentielle, des modifications structurelles ont été apportées aux méthodes d'indexation de la famille R-tree : X-tree [BKK96], DABS-tree [BK00], et Adaptive Trees [TP02].

X-tree X-tree [BKK96] introduit le concept de super-noeuds : en échange d'un partitionnement qui entraînerait une superposition spatiale trop importante des REMs générés, les noeuds directeurs sont élargis et deviennent des super-noeuds. Plusieurs pages disque consécutives sont assignées aux super-noeuds. Une valeur de seuil représentant la superposition spatiale maximale admise est utilisée pour décider le partitionnement des noeuds ou leur élargissement. Elle est fixée en tenant compte de paramètres de performance du système : disque, processeur et mémoire. Le modèle prend en considération la distribution spatiale des objets, mais assume que la distribution des requêtes suit la distribution des objets.

DABS-tree Une approche basée sur un modèle de coût est aussi utilisée dans le DABS-tree [BK00]. DABS-tree calcule dynamiquement les tailles de pages/noeuds en fonction de la distribution spatiale des objets. Le modèle de coût utilisé prend en considération les paramètres de performance du système et la distribution des objets, mais ne prend pas en considération la distribution des requêtes. DABS-tree utilise des REMs, mais aussi un partitionnement disjoint de l'espace. Pour cette raison, DABS-tree ne peut gérer que des collections de points multidimensionnels.

Adaptive Trees Dans la pratique, la distribution des requêtes ne suit pas toujours la distribution des objets. [TP02] propose une méthodologie pour convertir des structures d'indexation traditionnelles en des structures d'indexation adaptatives. L'objectif principal est de tenir compte à la fois de la distribution des objets et à la fois de la distribution de requêtes. Cette méthodologie est utilisée pour obtenir des B-trees Adaptatifs. Une généralisation multidimensionnelle est aussi proposée pour construire des R-trees Adaptatifs. Comme pour les X-trees, un gain de performance est obtenu en permettant aux noeuds d'occuper un nombre variable de pages disque consécutives. La taille d'un noeud, en nombre de pages, est décidée à sa création, et recalculée à chaque fois que la capacité du noeud est dépassée ou sous-occupée. La taille optimale est déterminée grâce aux statistiques associées à l'espace couvert par le noeud. Ces statistiques concernent la distribution spatiale des objets et des requêtes. Les statistiques sont maintenues dans un histogramme global divisant l'espace multidimensionnel dans des cellules de volume égal. Les statistiques d'objets et de requêtes sont intégrées dans un modèle de coût analytique

avec les paramètres de performance du système (disque, processeur, mémoire) pour calculer les tailles optimales des noeuds en fonction de leurs coûts d'accès. Cette approche semble efficace pour les B-trees, mais elle ne l'est pas pour les R-trees, notamment dans les espaces de dimensions élevés, pour trois raisons. Premièrement, le nombre des cellules utilisées pour maintenir des statistiques dans l'histogramme global augmente de façon exponentielle avec le nombre de dimensions. Deuxièmement, l'histogramme déployé est inadapté pour des objets étendus dont les extensions peuvent s'étendre sur plusieurs cellules. Enfin, les tailles des noeuds sont ajustées uniquement quand les noeuds deviennent surpeuplés ou sous-peuplés. Cela entraîne une dégradation de performance lorsque les variations de distribution des requêtes ne sont pas suivies par des variations équivalentes dans la distribution spatiale des objets.

Conclusion

Après l'étude approfondie des limitations des solutions d'indexation existantes, nous avons tiré la conclusion qu'une méthode d'accès efficace doit être impérativement assistée par un modèle de coût intégrant les caractéristiques de performance du système, et aussi qui tient compte de la distribution réelle des objets et des requêtes. Nous avons donc proposé une méthode de groupement des objets en clusters avec un modèle de coût adaptatifs aux requêtes, spécialement conçu pour améliorer la performance moyenne d'exécution des requêtes spatiales de type intervalle. Les idées directrices et les principales caractéristiques de notre méthode de groupement sont exposées dans la dernière section de ce chapitre.

Notre stratégie de groupement est basée sur un modèle de coût qui, d'une part, intègre les paramètres de la plateforme d'exécution et, d'autre part tient compte de la distribution des objets et aussi de la distribution des requêtes. Le but de notre stratégie de groupement est d'optimiser le groupement des objets et d'assurer pour les requêtes spatiales une performance de recherche moyenne meilleure que la recherche séquentielle, et ceci dans tous les cas de figure.

Dans le chapitre suivant, nous décrirons la stratégie de partitionnement en clusters, le critère de groupement spatial des objets, et le modèle de coût employé pour assister les décisions de restructuration des clusters.

7.3 Groupement en clusters avec un modèle de coût adaptatif aux requêtes

Notre groupement en clusters est basé, d'une part, sur un critère de groupement spatial adapté pour des objets multidimensionnels étendus, et d'autre part, sur un modèle de coût tenant compte de la distribution spatiale des objets et des requêtes, et intégrant les caractéristiques de performance de la plateforme d'exécution. Le critère de groupement spatial est utilisé pour partitionner, de manière déterministe, chaque cluster dans un nombre prédéfini de sous-clusters candidats représentant des futurs clusters possibles. Des statistiques d'objets et de requêtes sont maintenues au niveau des clusters et des sous-clusters candidats. Ces statistiques sont intégrées dans le modèle de coût avec les paramètres système pour évaluer la performance de recherche des clusters existants, et

pour estimer la performance de recherche des sous-clusters candidats. L'évaluation de la performance de recherche des clusters existants et des clusters candidats possibles est nécessaire pour supporter les décisions de création de nouveaux clusters profitables et les décisions de suppression des clusters qui ne sont plus rentables.

Deux types d'opérations de restructuration sont utilisés pour accomplir le processus de partitionnement des objets en clusters : l'éclatement de clusters et la fusion de clusters. L'éclatement d'un cluster est effectué en matérialisant une partie de ses sous-clusters candidats. De nouveaux clusters sont ainsi créés, mais uniquement si l'on estime que ceux-ci vont contribuer à l'amélioration de la performance moyenne de recherche. Les décisions d'éclatement des clusters sont assistées par la fonction de bénéfice de matérialisation dérivée du modèle de coût. Quand la profitabilité d'un cluster diminue suite aux changements de distribution des objets ou des requêtes, le cluster en question peut fusionner avec son ancêtre direct dans la hiérarchie de clusters. Grace aux opérations de fusion, les clusters non-rentables sont retirés de la base et leurs objets sont transférés aux ancêtres directs dans la hiérarchie de clusters. Les décisions de fusion des clusters sont assistées par la fonction de bénéfice de fusion également dérivée du modèle de coût.

7.3.1 Définition du cluster

Un cluster représente un groupe d'objets qui sont stockés, accédés et vérifiés ensembles lors des recherches spatiales. Chaque cluster est représenté par sa signature résumant les propriétés de groupement. La signature est utilisée pour vérifier :

- Si un objet peut être membre du cluster

  Seuls les objets qui vérifient la signature du cluster peuvent être membres du cluster.

- Si le cluster doit être exploré lors d'une sélection spatiale

  Seuls les clusters dont les signatures satisfont le critère spatial de sélection par rapport à l'objet-requête (intersection, inclusion, ou couverture spatiale) sont explorés.

Stockage du cluster

La création de nouveaux clusters profitables et la suppression de clusters non rentables sont basées sur le modèle de coût qui assiste notre stratégie de groupement. Les clusters n'ont pas de restrictions de taille. Le nombre d'objets membres d'un cluster est déterminé par la distribution spatiale des objets. Le modèle de coût est utilisé pour assurer une bonne performance de recherche pour le cluster. Pour des raisons de performance, les objets appartenant au même cluster sont stockés de façon séquentielle. Le placement séquentiel des objets est employé dans les deux cas de stockage envisagés : un stockage des objets dans la mémoire centrale et un stockage des objets sur le disque dur. En mémoire, ce placement assure une bonne performance de recherche parce qu'il augmente la localité des données. L'accès séquentiel des objets bénéficie ainsi des technologies de cache et de lecture prédictive disponibles sur les systèmes modernes. Sur disque, le placement contigu augmente aussi la localité des données et permet d'éviter des opérations coûteuses de repositionnement de la tête de lecture/écriture, et ainsi de bénéficier d'une vitesse de transfert élevée entre le disque et la mémoire centrale. Au moment de la création ou du

déplacement de clusters, un nombre de places libres sont réservées à la fin des clusters pour de nouveaux objets. Cela permet d'éviter des opérations coûteuses de replacement de clusters lors d'insertions de nouveaux objets.

Indicateurs de performance

Afin d'évaluer la performance d'exécution des requêtes, au niveau des clusters et des sous-clusters candidats, nous maintenons deux statistiques appelées indicateurs de performance :

1. le nombre d'objets membres de chaque cluster

   Avec les paramètres de performance du système (temps d'accès et de transfert des objets, coûts de vérification), cette statistique permet d'estimer le coût d'exploration du cluster. En effet, l'exploration d'un cluster comporte l'accès et la vérification individuelle de chaque objet membre du cluster.

2. le nombre de requêtes explorant chaque cluster pendant une période de temps

   Cette statistique permet d'estimer la probabilité d'accès du cluster. Celle-ci peut être vue comme le rapport entre le nombre de requêtes qui explorent le cluster durant une période de temps et le nombre total de requêtes adressées au système pendant ce temps.

7.3.2 Processus de groupement en clusters

Le processus de groupement des objets en clusters est récursif. Périodiquement, les clusters existants sont éclatés pour former de nouveaux clusters, plus profitables. Les clusters qui ne sont plus rentables sont fusionnés avec leurs clusters parents. Les opérations de regroupement d'objets, d'éclatements et de fusions de clusters, sont effectuées uniquement si elles sont considérées comme étant profitables pour la performance moyenne de recherche.

Initialement, la collection d'objets forme un seul cluster, appelé le cluster racine. La signature du cluster racine accepte tout objet spatial. Toutes les requêtes spatiales vont explorer le cluster racine, donc la probabilité d'accès à ce cluster est toujours égale à 1. A la création du cluster racine, nous utilisons la fonction de groupement, qui implémente le critère de groupement spatial afin de partitionner ce cluster dans un nombre prédéfini de sous-clusters candidats. Les signatures des sous-clusters candidats sont données par la fonction de groupement en fonction de la signature du cluster d'origine (dans ce cas le cluster racine). Des indicateurs de performances sont maintenus pour le cluster racine et pour tous ses sous-clusters candidats.

Des opérations d'éclatement et de fusion de clusters sont effectuées périodiquement. Elles sont prises en considération après avoir exécuté un nombre suffisant de requêtes pour mettre à jour les indicateurs de performance.

La création de nouveaux clusters est décidée par la fonction de bénéfice de matérialisation. Cette fonction s'applique aux sous-clusters candidats des clusters existants pour évaluer les profits potentiels de leurs matérialisations. Les sous-clusters candidats avec les meilleurs

profits potentiels sont sélectionnés pour être matérialisés. La matérialisation d'un sous-cluster candidat comporte deux actions. Premièrement, un nouveau cluster est créé, reprenant la signature du sous-cluster candidat correspondant. Tous les objets du cluster initial qui satisfont cette signature sont ainsi déplacés vers le nouveau cluster. Deuxièmement, la fonction de groupement est appliquée sur la signature du nouveau cluster pour obtenir ses sous-clusters candidats. Des indicateurs de performance sont attachés aux sous-clusters candidats du nouveau cluster afin de cumuler des statistiques d'objets et de requêtes pour de futures opérations de restructuration.

Les opérations d'éclatement sont appliquées sur tous les clusters, de façon périodique. Après un nombre d'opérations de regroupement d'objets, un arbre de clusters est obtenu. Chaque cluster est représenté par sa signature, par ses indicateurs de performance, et par un ensemble de sous-clusters candidats avec leurs indicateurs de performance. Quand un cluster n'est plus rentable en raison des changements survenant dans la distribution spatiale des objets ou des requêtes, il est retiré de la base et ses objets sont transférés dans le cluster parent (ancêtre direct dans la hiérarchie de clusters). Cette opération de restructuration, appelée fusion, permet au groupement des objets de s'adapter aux variations de distribution des objets et des requêtes.

L'opération de fusion entre un cluster et son cluster parent est décidée par la fonction de bénéfice de fusion qui évalue son impact sur la performance moyenne des requêtes spatiales. Pour rendre possibles les opérations de fusion, chaque cluster matérialisé garde une référence (un pointeur) vers son parent direct, ainsi qu'une liste de références vers les clusters enfants. Le cluster racine n'a pas de parent et ne peut être jamais enlevé de la base de données.

7.3.3 Fonctions supportant la stratégie de groupement

Par la suite nous allons présenter brièvement les rôles des trois fonctions supportant notre stratégie de groupement : la fonction de groupement implémentant le critère de groupement spatial des objets, la fonction de bénéfice de matérialisation, et la fonction de bénéfice de fusion ; les deux dernières étant dérivées du modèle de coût.

- Fonction de groupement

  La fonction de groupement implémente le critère de groupement spatial des objets. A partir de la signature d'un cluster, la fonction de groupement produit un ensemble prédéterminé de sous-signatures représentant les signatures des sous-clusters candidats du cluster initial. Formellement, les sous-signatures de la signature initiale satisfont la propriété suivante : tout objet spatial qui satisfait une sous-signature satisfait aussi la signature initiale. De plus, un objet spatial membre du cluster initial pourra satisfaire les signatures de plusieurs de ses sous-clusters. La fonction de groupement assure une compatibilité en arrière dans la hiérarchie de clusters. Cette compatibilité permet d'effectuer des opérations de fusion entre les clusters enfants et parents.

  La fonction de groupement doit résoudre le compromis suivant : d'une part, le nombre de sous-clusters candidats doit être assez élevé pour offrir un nombre significatif d'alternatives de groupement des objets. D'autre part, le nombre de sous-clusters candidats ne peut pas être trop élevé parce que le coût de maintenance de statis-

tiques devient trop important. Il faut tenir compte du fait que les indicateurs de performance sont maintenus pour tous les sous-clusters candidats des clusters existants. La fonction de groupement donne le nombre de sous-clusters candidats de chaque cluster et détermine aussi le nombre de sous-clusters candidats satisfaits par un objet membre du cluster considéré.

- Fonction de bénéfice de matérialisation

  La fonction de bénéfice de matérialisation est dérivée du modèle de coût. Chaque cluster est associé avec un ensemble de sous-clusters candidats aux matérialisations. Le rôle de la fonction de bénéfice de matérialisation est d'estimer pour chacun des sous-clusters candidats l'impact de sa matérialisation sur la performance moyenne des requêtes spatiales. A cet effet, la fonction de bénéfice de matérialisation prend en compte les indicateurs de performance du sous-cluster candidat, ainsi que ceux du cluster initial, et l'ensemble des paramètres système influant sur le temps de réponse. Les clusters pour lesquels on attend les meilleures augmentations de performance moyenne de recherche sont choisis pour être matérialisés. L'opération de matérialisation présente un intérêt quand le sous-cluster candidat a une probabilité d'accès inferieure à celle du cluster d'origine, et quand le nombre d'objets à transférer vers le nouveau cluster est assez élevé pour que le coût d'un cluster supplémentaire soit rentable.

- Fonction de bénéfice de fusion

  La fonction de bénéfice de fusion est également dérivée du modèle de coût. Le rôle de la fonction de bénéfice de fusion est d'évaluer l'impact d'une opération de fusion entre un cluster et son cluster parent sur la performance moyenne de recherche. A cet effet, la fonction de bénéfice de fusion prend en compte les indicateurs de performance du cluster considéré, du cluster parent, et l'ensemble des paramètres influant sur le temps de réponse des requêtes. Si l'impact de l'opération de fusion est bénéfique pour la performance moyenne de recherche, alors la fusion peut être effectuée. L'opération de fusion présente un intérêt quand la probabilité d'accès du cluster enfant devient proche de la probabilité d'accès du cluster parent et quand le nombre d'objets du cluster enfant diminue significativement.

7.4 Algorithmes d'organisation et de manipulation de la base d'objets

7.4.1 Restructuration des clusters

Deux opérations de restructuration de base sont utilisées pour améliorer la performance moyenne de recherche :

1. L'éclatement des clusters

   L'éclatement d'un cluster est effectué par des matérialisations d'une partie de ses sous-clusters candidats.

2. Les fusions des clusters avec leurs clusters parents

Lors d'une fusion, les objets du cluster initial sont transférés dans le cluster parent et le cluster initial est supprimé de la base de données.

Les Figures 7.1, 7.2 et 7.3 illustrent les procédures invoquées lors de la restructuration d'un cluster.

RestructureClusters()

1. **if** (q(system) % period_q = 0) **then**
2. RecursiveMergeClusters(root);
3. RecursiveSplitClusters(root);

End.

Figure 7.1: Restructuration des clusters

La Figure 7.1 présente le processus général de restructuration des clusters.

Fusion de cluster

La procédure de fusion d'un cluster avec son cluster parent est détaillée dans la Figure 7.2. Les objets du cluster enfant sont transférés dans le cluster parent. Cette opération nécessite la mise à jour du nombre d'objets dans le cluster parent et dans ses sous-clusters candidats. Pour préserver la hiérarchie de clusters, le cluster parent du cluster à supprimer devient le nouveau parent des clusters enfants du dernier. Finalement, le cluster à supprimer est enlevé de la base de données.

La complexité de l'opération de fusion est dominée par (1) le coût de transfert des objets du cluster enfant au cluster parent, et par (2) le coût de mise à jour des indicateurs de performance associés au cluster parent et à ses sous-clusters candidats. Le cluster parent peut ne pas être assez large pour accueillir tous les objets réacquis du cluster enfant. En pratique, nous créons un nouveau cluster et nous y transférons tous les objets du cluster parent et du cluster enfant. L'ancien cluster parent sera enlevé de la base de données en même temps que le cluster enfant. Sa place sera prise par le nouveau cluster. Le transfert des objets nécessite une lecture et une écriture des objets des deux clusters impliqués, parent et enfant. Comme les objets sont placés séquentiellement, leur transfert est effectué de façon efficace par des opérations de lecture et d'écriture en bloc. Le transfert des objets en bloc est important surtout dans le cas d'un stockage sur disque où ce type de transfert évite de nombreuses opérations d'entrée/sortie très coûteuses. En termes de nombre d'opérations de lecture/écriture en bloc, le coût de transfert des objets dépend des tailles de deux clusters impliqués. La mise à jours des statistiques d'objets associées au cluster parent et à ses sous-clusters candidats est effectuée quand les objets du cluster enfant sont lus et écrits dans le cluster parent. Cette tâche comporte l'incrémentation du nombre d'objets membres du cluster parent et l'incrémentation du nombre d'objets pour chacun de ses sous-clusters candidats dont la signature accepte un objet acquis du cluster enfant. Le coût de cette tâche est proportionnel au nombre d'objets dans le cluster enfant, multiplié par le nombre de sous-clusters candidats capables

ClusterMerge ($\mathsf{c} \in \mathcal{C}$, $\mathsf{a} \in \mathcal{C} \mid \mathsf{a} \leftarrow \mathit{parent}(\mathsf{c})$)

// Move data objects from child cluster c to parent cluster a:
1. **let** $\mathit{objects}(\mathsf{a}) \leftarrow \mathit{objects}(\mathsf{a}) \cup \mathit{objects}(\mathsf{c})$;

// Update data statistics for parent cluster a:
2. **let** $n(\mathsf{a}) \leftarrow n(\mathsf{a}) + n(\mathsf{c})$;

// Update data statistics for candidate subclusters of parent cluster a:
3. **for each** s **in** $\mathit{candidates}(\mathsf{a})$ **do**
4. **let** $\mathcal{M}(\mathsf{s}, \mathsf{c}) \leftarrow \{\mathsf{o} \in \mathit{objects}(\mathsf{c}) \mid \mathsf{o}$ matches $\sigma(\mathsf{s})\}$;
5. **let** $n(\mathsf{s}) \leftarrow n(\mathsf{s}) + \mathit{card}(\mathcal{M}(\mathsf{s}, \mathsf{c}))$;

// Set parent reference for child clusters of c:
6. **for each** s **in** $\mathit{children}(\mathsf{c})$ **do**
7. **let** $\mathit{parent}(\mathsf{s}) \leftarrow \mathsf{a}$;

// Update list of child references for parent cluster a:
8. **let** $\mathit{children}(\mathsf{a}) \leftarrow \mathit{children}(\mathsf{a}) \cup \mathit{children}(\mathsf{c})$;
9. **let** $\mathit{children}(\mathsf{a}) \leftarrow \mathit{children}(\mathsf{a}) \setminus \{\mathsf{c}\}$;

// Remove c from database:
10. **let** $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\mathsf{c}\}$;

End.

Figure 7.2: Fusion de cluster

d'accueillir un objet membre du cluster. Le nombre de sous-clusters candidats pouvant accueillir un objet membre du cluster est donné par la fonction de groupement.

Eclatement de cluster

Le processus d'éclatement d'un cluster est illustré dans la Figure 7.3. Les sous-clusters candidats pour lesquels les matérialisations potentielles montrent les meilleurs profits sont identifiés à la première étape : Beta désigne l'ensemble des sous-clusters candidats promettant le meilleur profit positif. Si Beta n'est pas vide, un de ses membres devient le sujet de la prochaine matérialisation : Un nouveau cluster est créé et rajouté à la base de données. Les objets du cluster initial satisfaisant la signature du sous-cluster candidat à matérialiser sont identifiés et transférés du cluster d'origine vers le nouveau cluster. La configuration du nouveau cluster est établie : signature, nombre d'objets membres, cluster parent. Le nombre d'objets du cluster initial est mis à jour (réduit), ainsi que les nombres d'objets associés aux sous-clusters candidats du cluster initial. On sait qu'un même objet satisfait la signature de plusieurs sous-clusters candidats. Cependant, une fois enlevé du cluster initial et transféré dans un cluster matérialisé, l'objet ne peut plus compter pour les sous-clusters candidats du cluster d'origine. L'étape suivante initialise les statistiques d'objets associées aux sous-clusters candidats du nouveau cluster.

La procédure d'éclatement du cluster initial peut ensuite continuer avec la sélection du prochain meilleur sous-cluster candidat en revenant à la première étape. Le processus

ClusterSplit ($\mathsf{c} \in \mathcal{C}$)

// Find best candidate subclusters for materialization:
1. **let** $\mathcal{B} \leftarrow \{\mathsf{b} \in \mathit{candidates}(\mathsf{c}) \mid \beta(\mathsf{b},\mathsf{c}) > \mathsf{min\_}\beta \wedge \beta(\mathsf{b},\mathsf{c}) \geq \beta(\mathsf{d},\mathsf{c}),\ \forall \mathsf{d} \neq \mathsf{b} \in \mathit{candidates}(\mathsf{c})\}$;

2. **if** ($\mathcal{B} \neq \emptyset$) **then**

// One of best candidate subclusters is materialized:
3. **let** $\mathsf{b} \in \mathcal{B}$;

// Create new database cluster d;
4. **let** $\mathsf{C} \leftarrow \mathsf{C} \cup \{\mathsf{d}\}$;

// Move qualifying data objects from cluster c to new cluster d;
5. **let** $\mathcal{M}(\mathsf{b},\mathsf{c}) \leftarrow \{\mathsf{o} \in \mathit{objects}(\mathsf{c}) \mid \mathsf{o}$ matches $\sigma(\mathsf{b})\}$;
6. **let** $\mathit{objects}(\mathsf{d}) \leftarrow \mathcal{M}(\mathsf{b},\mathsf{c})$;
7. **let** $\mathit{objects}(\mathsf{c}) \leftarrow \mathit{objects}(\mathsf{c}) \setminus \mathcal{M}(\mathsf{b},\mathsf{c})$;

// Set configuration for new cluster d:
8. **let** $\sigma(\mathsf{d}) \leftarrow \sigma(\mathsf{b})$;
9. **let** $\mathit{parent}(\mathsf{d}) \leftarrow \mathsf{c}$;
10. **let** $n(\mathsf{d}) \leftarrow n(\mathsf{b})$;

// Set data object statistics for candidate subclusters of cluster d:
11. **for each** s **in** $\mathit{candidates}(d)$ **do**
12. **let** $\mathcal{M}(\mathsf{s},\mathsf{d}) \leftarrow \{\mathsf{o} \in \mathit{objects}(\mathsf{d}) \mid \mathsf{o}$ matches $\sigma(\mathsf{s})\}$;

// Update data object statistics for cluster c:
13. **let** $n(\mathsf{c}) \leftarrow n(\mathsf{c}) - n(\mathsf{d})$;

// Update data object statistics for candidate subclusters of cluster c:
14. **for each** s **in** $\mathit{candidates}(\mathsf{c})$ **do**
15. **let** $\mathcal{M}(\mathsf{s},\mathsf{d}) \leftarrow \{\mathsf{o} \in \mathit{objects}(\mathsf{d}) \mid \mathsf{o}$ matches $\sigma(\mathsf{s})\}$;
16. **let** $n(\mathsf{s}) \leftarrow n(\mathsf{s}) - \mathit{card}(\mathcal{M}(\mathsf{s},\mathsf{d}))$;
17. **let** $n(\mathsf{s}) \leftarrow \mathit{card}(\mathcal{M}(\mathsf{s},\mathsf{d}))$;

// Consider next candidate subcluster for materialization:
18. **go to** 1.

// If the cluster c was split, then reset associated query statistics:
19. **if** c *was split* **then**
20. **let** $q(\mathsf{c}) \leftarrow 0$;
21. **for each** s **in** $\mathit{candidates}(\mathsf{c})$ **do**
22. **let** $q(\mathsf{s}) \leftarrow 0$;

End.

Figure 7.3: Eclatement de cluster

de matérialisation est repris et répété jusqu'à ce qu'il n'y a plus de sous-clusters candidats profitables. La sélection du prochain sous-cluster à matérialiser est faite d'une manière

"greedy" : le candidat le plus profitable est matérialisé en premier. Pour tenir compte des changements d'objets entraînés par les matérialisations successives, l'ensemble des meilleurs sous-clusters candidats, Beta, doit être réévalué chaque fois. A la fin, le cluster initial contient uniquement les objets qui ne satisfont les signatures d'aucun de nouveaux sous-clusters matérialisés. Evidemment, quand aucun sous-cluster candidat n'est profitable pour la performance moyenne de recherche, le cluster initial reste inchangé.

L'éclatement d'un cluster comporte une ou plusieurs opérations de transfert d'objets du cluster initial aux sous-clusters candidats matérialisés. Lorsqu'un sous-cluster candidat est matérialisé, les objets qui y qualifient sont déplacés. Grâce aux statistiques d'objets associées aux sous-clusters candidats, le nombre exact d'objets qui satisfont la signature du nouveau cluster est connu. Les objets dans les deux clusters, le nouveau cluster et le cluster initial, doivent être stockés de façon séquentielle. En pratique, nous créons deux nouveaux clusters : un pour matérialiser le sous-cluster candidat, et un autre pour recevoir les objets restant dans le cluster initial. A la fin du transfert d'objets, le deuxième cluster prend la place du cluster initial. Chaque objet du cluster initial est lu, comparé avec la signature du sous-cluster à matérialiser, et écrit, soit dans le nouveau cluster correspondant au sous-cluster candidat, soit dans le nouveau cluster censé remplacer le cluster d'origine. Comme exigé, les objets seront placés de façon séquentielle dans les deux clusters. En termes d'opérations de lecture/vérification/écriture, le coût de transfert des objets est proportionnel au nombre d'objets du cluster initial. Pour des raisons de performance, les opérations de lecture/écriture sont effectuées en bloc. Le transfert des objets en bloc est important surtout pour le stockage sur disque. Pendant le transfert des objets, nous mettons aussi à jour les statistiques d'objets associées aux clusters impliqués ainsi que leurs sous-clusters candidats. Chaque fois qu'un objet convenant pour le sous-cluster à matérialiser est identifié, nous décrémentons le nombre d'objets membres du cluster initial dans lequel il se trouvait, ainsi que celui associé à ses sous-clusters candidats qui satisfont la signature de l'objet. En même temps, nous incrémentons les statistiques d'objets du sous-cluster matérialisé dans lequel l'objet va être ajouté ainsi que de ses sous-clusters candidats correspondants. Le coût de cette tâche est proportionnel au nombre d'objets qui matchent la signature du sous-cluster à matérialiser, multiplié par le nombre de sous-clusters candidats satisfaisant la signature d'un objet. Ce dernier est donné par la fonction de groupement.

7.4.2 Exécution des requêtes spatiales

Une requête spatiale désigne un objet de référence (objet requête) et une relation spatiale, d'intersection, d'inclusion, ou de couverture ; cette relation doit être vérifiée entre l'objet requête et les objets de la base de données faisant partie du résultat. Répondre à une requête spatiale consiste à explorer les clusters matérialisés dont les signatures satisfont la relation spatiale avec l'objet requête. Tous les objets des clusters explorés sont individuellement vérifiés par rapport au critère de sélection spatiale. L'algorithme d'exécution d'une requête spatiale est illustré dans la Figure 7.4. Il est simple : les signatures des clusters sont comparées avec l'objet requête, et les objets des clusters dont les signatures satisfont la relation spatiale exigée, sont lus et vérifiés. Comme indicateur de performance pour la probabilité d'accès, les statistiques sur le nombre de requêtes sont incrémentées pour tous les clusters explorés, ainsi que pour les sous-clusters candidats susceptibles d'être explorés s'ils étaient matérialisés.

```
SpatialQuery (query object ρ, spatial selection criterion ∇) : data object set

//    Initialize the query answer set:
1.    let R ← ∅;

//    Determine the clusters to be explored:
2.    let X ←ClustersToExplore(root, ρ, ∇);

//    Exploration of qualifying clusters:
3.    for each cluster c ∈ X do

//        Check all data objects against the selection criterion:
4.        for each object o in objects(c) do
5.            if (ρ ∇ o) then
6.                let R ← R ∪ {o};

//        Update query statistics for cluster c:
7.        let q(c) ← q(c) + 1;

//        Update query statistics for candidate subclusters of cluster c:
8.        let S ← {s ∈ candidates(c) | ρ ∇ σ(s)};
9.        for each s in S do
10.           let q(s) ← q(s) + 1;

//    Return the query result:
11.   return R;

End.
```

Figure 7.4: Exécution des requêtes spatiales

La complexité d'exécution d'une requête spatiale est donné par : (1) la vérification des signatures de clusters pour déterminer les clusters à explorer; (2) la vérification individuelle des objets des clusters explorés pour déterminer les objets faisant partie du résultat de la requête; et (3) la mise à jour des statistiques de requêtes associées aux clusters explorés et à leurs sous-clusters candidats. En termes de vérifications de signatures, le coût de la première tâche est, dans le pire des cas, proportionnel au nombre total de clusters. Cependant, en pratique, l'organisation hiérarchique des clusters permet d'éviter la vérification exhaustive des signatures. En effet, les clusters descendant d'un cluster dont la signature ne demande pas son exploration n'ont pas besoin d'être explorés. Plus la sélectivité d'une requête est élevée, moins le coût de la première tâche est important. En termes de coûts d'accès aux objets, le coût de la deuxième opération est proportionnel au nombre de clusters explorés. En termes de coûts de lecture/vérification, le coût de la deuxième opération est proportionnel au nombre d'objets des clusters explorés. Si les clusters sont stockés sur disque, des opérations d'entrée/sortie coûteuses sont nécessaires : repositionnement de la tête de lecture/écriture du disque au bon endroit et transfert des objets entre le disque et la mémoire centrale. Ces opérations sont effectuées en bloc, car les objets sont stockés séquentiellement au niveau des clusters. La troisième tâche comporte l'incrémentation des statistiques de requêtes exploratoires associées aux clusters explorés et à leurs sous-clusters candidats dont les signatures matchent l'objet requête.

Le coût de cette dernière tâche est proportionnel au nombre de clusters explorés, multiplié par le nombre de sous-clusters candidats satisfaisant la signature d'un objet. Ce dernier est donné par la fonction de groupement.

7.4.3 Opérations de mise à jour d'objets

L'insertion et la suppression des objets dans la base de données sont les deux opérations de mise à jour qui sont présentées ci-après.

Insertion d'objet

Quand un nouvel objet doit être inséré dans la base de données, à part le cluster racine dont la signature, la plus générale, matche tout objet spatial, d'autres clusters peuvent également accueillir le nouvel objet. Les clusters capables de recevoir le nouvel objet sont identifiés grâce à leurs signatures. Parmi ces clusters, nous choisissons de placer le nouvel objet dans le cluster qui a la plus petite probabilité d'accès. Notre stratégie d'insertion vise directement à minimiser la probabilité d'accéder et de vérifier le nouvel objet quand il ne fait pas partie du résultat d'une sélection spatiale. La Figure 7.5 illustre la procédure d'insertion. L'insertion d'un nouvel objet doit mettre à jour les statistiques d'objet du cluster choisi ainsi que de ses sous-clusters candidats pouvant accueillir le nouvel objet.

ObjectInsertion (data object ρ)

// Determine and select best cluster accepting object ρ:
1. **let** $\mathcal{B} \leftarrow \{\mathsf{b} \in \mathcal{C} \mid \rho$ matches $\sigma(\mathsf{b}) \ \wedge \ p(\mathsf{b}) \leq p(\mathsf{c}), \forall \mathsf{c} \neq \mathsf{b} \in \mathcal{C}\}$;
2. **let** $\mathsf{b} \in \mathcal{B}$;

// Insert data object ρ into selected cluster b:
3. **let** $objects(\mathsf{b}) \leftarrow objects(\mathsf{b}) \cup \{\rho\}$

// Increment data statistics for cluster b:
4. **let** $n(\mathsf{b}) \leftarrow n(\mathsf{b}) + 1$;

// Update data statistics for candidate subclusters of cluster b:
5. **let** $\mathcal{S} \leftarrow \{\mathsf{s} \in candidates(\mathsf{b}) \mid \rho$ matches $\sigma(\mathsf{s})\}$;
6. **for each** s **in** $\mathcal{S}$ **do**
7. **let** $n(\mathsf{s}) \leftarrow n(\mathsf{s}) + 1$;

End.

Figure 7.5: Insertion d'objet

La complexité de l'opération d'insertion d'objet est donnée par : (1) la vérification de signatures pour identifier les clusters capables d'accueillir le nouvel objet; (2) la sélection du cluster le moins accédé parmi les clusters candidats; (3) l'insertion du nouvel objet dans le cluster choisi; et (4) la mise à jour des statistiques d'objets des clusters/sous-clusters affectés par l'opération d'insertion. Le coût de la première tâche est dans le pire des cas proportionnel au nombre total de clusters, en termes de coûts de vérification de signatures. Cependant, en pratique, l'organisation hiérarchique des clusters permet

d'éviter la vérification exhaustive des signatures. En effet, les clusters descendant d'un cluster dont la signature n'accepte pas le nouvel objet ne peuvent pas non plus accueillir cet objet. La seconde tâche concerne seulement les clusters capables de recevoir le nouvel objet. Leurs probabilités d'accès sont évaluées et le cluster ayant la plus petite probabilité d'accès est retenu. Le coût de cette évaluation est linéaire avec le nombre de clusters capables d'accepter le nouvel objet. La troisième tâche demande une opération d'écriture pour stocker le nouvel objet dans la première place libre à la fin du cluster choisi. Lorsque les clusters sont stockés sur le disque, cette tâche comporte un repositionnement de la tête de lecture/écriture du disque au bon endroit suivi par une écriture de l'objet. Enfin, le coût de la quatrième tâche est proportionnel au nombre de sous-clusters candidats pour lesquels il est nécessaire d'incrémenter les nombres d'objets membres. Le nombre de sous-clusters candidats dont les statistiques doivent être mises à jour est fixe et dépend de la fonction de groupement. Comme seulement un seul objet est inséré à chaque fois lors de l'opération d'insertion, le coût de la dernière tâche est négligeable par rapport au coût initial de vérification des signatures de clusters ou par rapport au coût d'écriture du nouvel objet sur disque.

Suppression d'objet

Pour enlever un objet de la base de données, nous devons d'abord retrouver le cluster qui contient l'objet recherché. Tous les clusters pouvant accueillir l'objet à enlever doivent être explorés. Ces clusters sont identifiés grâce à leurs signatures. La Figure 7.6 illustre l'algorithme de suppression d'objet. Les objets des clusters dont les signatures satisfont celle de l'objet recherché sont tous comparés à ce dernier, jusqu'à ce que le cluster correspondant soit trouvé. L'objet est ensuite enlevé du cluster correspondant. Evidemment, la suppression d'un objet demande la mise à jour des statistiques d'objets associées au cluster affecté ainsi qu'à ses sous-clusters candidats. Une fois que l'objet est retrouvé et enlevé du cluster correspondant, l'opération de suppression prend fin.

Lors de la suppression d'un objet, le dernier objet du cluster prend la place de l'objet enlevé. Ce choix de placement augmente la complexité de la procédure de suppression, mais facilite la gestion des places libres au niveau des clusters. Les places libres sont toujours à la fin des clusters, ce qui assure le placement séquentiel des objets au niveau des clusters et simplifie également la lecture et l'insertion des objets.

La complexité de l'opération de suppression d'objet est déterminée par l'exécution d'une requête spatiale simplifiée, auquelle il faut ajouter le coût de suppression d'un objet du cluster correspondant. En effet, ce cluster est retrouvé en explorant uniquement les clusters dont les signatures satisfont de façon exacte l'objet recherché. De ce fait, la suppression d'un objet est plus rapide qu'une requête spatiale qui elle doit explorer tous les clusters satisfaisant le critère spatial de sélection (insertion, inclusion ou couverture spatiale). De plus, la recherche se termine dès que le cluster contenant l'objet à enlever est découvert. La place de cet objet est prise par le dernier objet du cluster. Par conséquent, la suppression d'un objet demande deux opérations d'accès : une lecture et ensuite une écriture de l'objet.

```
ObjectDeletion ( data object ρ )

    //      Determine the clusters to be explored:
    1.      let 𝒳 ← {c ∈ 𝒞 | ρ matches σ(c)};

    //      Cluster explorations:
    2.      for each cluster c ∈ 𝒳 do

    //          Consider all data objects from cluster c:
    3.          for each object o in objects(c) do

    //              Look for the wanted object ρ:
    4.              if (ρ equals o) then

    //                  Remove data object ρ from the cluster c:
    5.                  let objects(c) ← objects(c) \ {ρ}

    //                  Decrement data statistics of cluster c:
    6                   let n(c) ← n(c) − 1;

    //                  Update data statistics of candidate clusters of cluster c:
    7.                  let 𝒮 ← {s ∈ candidates(c) | ρ matches σ(s)};
    8.                  for each s in 𝒮 do
    9.                      let n(s) ← n(s) − 1;

    //                  End deletion procedure:
    10.                 go to End;

End.
```

Figure 7.6: Suppression d'objet

7.5 Implémentation et évaluation expérimentale

7.5.1 Gestion de la mémoire et de l'espace de stockage

L'arbre de signatures des clusters et les indicateurs de performance associés aux clusters matérialisés et aux sous-clusters candidats sont maintenus en mémoire centrale. Pour les objets, nous considérons deux scénarios de stockage : un stockage en mémoire centrale et un stockage sur le disque dur. Les objets du même cluster sont placés et stockés de façon séquentielle. Pour éviter des replacements fréquents des clusters lors des insertions de nouveaux objets, un certain nombre de places est réservé à la fin de chaque nouveau cluster. Nous considérons que le nombre de places réservés représente 25% de la taille du cluster. Ainsi, prenant en compte la distribution des objets, les grands clusters ont plus de places libres que les petits clusters. Un facteur de remplissage de l'espace de stockage d'au moins 75% est dans tous les cas assuré.

7.5.2 Evaluation expérimentale

Afin de vérifier l'efficacité de notre solution de groupement, nous avons premièrement effectué un nombre important d'expériences. Puis nous avons évalué la performance de recherche des requêtes spatiales sur de larges collections d'objets étendus avec beaucoup de dimensions, suivant des distributions spatiales uniformes et non uniformes. Ensuite, nous avons comparé notre solution de groupement avec la Recherche Séquentielle et avec les structures d'indexation R*-tree et X-tree. Puis, nous avons mesuré le temps d'exécution des requêtes, le nombre des clusters/noeuds explorés, et le taux d'objets vérifiés en moyenne. Ensuite, nous avons aussi évalué le temps d'exécution des opérations de mise à jour comme les insertions et suppression d'objets. Enfin nous avons analysé le comportement de notre méthode de groupement et son adaptabilité aux variations dynamiques des requêtes et des objets.

Conclusions Expérimentales

Notre solution de groupement démontre une meilleure performance de recherche que la Recherche Séquentielle, le R*-tree, et le X-tree, dans la plupart des cas, pour les deux scenarios de stockage considérés : en mémoire centrale et sur disque. Les évaluations expérimentales valident les points suivants : Notre méthode de groupement suit la distribution réelle des objets et de requêtes. Notre système est scalable en nombre d'objets et présente un bon comportement dans les espaces avec beaucoup de dimensions : jusqu'à 40 dimensions dans nos tests. Notre approche de groupement gère bien les objets avec différentes caractéristiques au niveau des dimensions et les objets et requêtes repartis dans des régions d'intérêt (sélectivités différentes selon les dimensions, ou des objets et des requêtes avec des distributions non uniformes dans l'espace multidimensionnel). Le surcoût des opérations de réorganisation et de restructuration des clusters (matérialisations des sous-clusters candidats et fusions des clusters) reste faible. Les opérations de mise à jour de la base d'objets (les insertions et les suppressions d'objets) sont rapides. Notre méthode de groupement s'adapte bien à l'évolution dynamique des requêtes et des objets. Elle démontre également une bonne flexibilité et une bonne prise en compte des paramètres de performance de la plateforme d'exécution.

7.6 Conclusions et perspectives de recherche

Les nouvelles applications de dissémination sélective des informations ont fait apparaitre de nouveaux défis et exigences pour le domaine de l'indexation multidimensionnelle. Une application avancée doit supporter des taux élevés de requêtes spatiales sur de larges collections d'objets multidimensionnels étendus avec beaucoup de dimensions et avec de longues extensions dans les dimensions. Elle doit également pouvoir gérer des objets et des requêtes dynamiques, évoluant dans le temps. Les insertions et les suppressions des objets doivent être rapides afin de ne pas affecter de façon significative les performances de recherche. Cependant, les structures d'indexation existantes ne sont pas adaptées pour ce type d'applications. Dans notre thèse, nous avons présenté une solution de groupement alternative, basée sur un modèle de coût. Le groupement des objets prend en compte la distribution réelle des objets et des requêtes, ainsi que les paramètres de performance de la

plateforme d'exécution. Notre méthode de groupement utilise un critère de groupement spatial original et s'avère plus efficace que les méthodes d'indexation traditionnelles.

Plusieurs perspectives de recherche sont exposées dans le dernier chapitre de la thèse : un mécanisme de control avancé pour déclencher les opérations de restructuration des clusters ; des méthodes de groupement d'objets alternatives pour supporter l'implémentation de la fonction de groupement ; une stratégie de cache pour améliorer la performance de recherche des applications basées sur un stockage disque ; et un modèle de représentation approximée des objets multidimensionnels étendus pour accélérer les sélections spatiales en mémoire.

Bibliography

[AF00] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, Cairo, Egypt, 2000.

[AG01] L. Amsaleg and P. Gros. Content-based retrieval using local descriptors: problems and issues from a database perspective. *Pattern Analysis and Applications*, 4(2/3):108–124, 2001.

[AHVV02] L. Arge, K. Hinrichs, J. Vahrenhold, and J. S. Vitter. Efficient bulk-operations on dynamic r-trees. *Algorithmica*, 33(1):104–128, 2002.

[APR99] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Philadelphia, PA, USA, 1999.

[AT97] C. H. Ang and T. C. Tan. New linear node splitting algorithm for R-trees. In *Proceedings of the 5th SSD Conference, pages 339-349*, Berlin, Germany, 1997.

[BBK98a] S. Berchtold, C. Böhm, and H.-P. Kriegel. Improving the query performance of high-dimensional index structures using bulk-load operations. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, Valencia, Spain, 1998.

[BBK98b] S. Berchtold, C. Böhm, and H.-P. Kriegel. The Pyramid-technique: Towards breaking the curse of dimensionality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, WA, USA, 1998.

[BBK01] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[BBKK97] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *ACM PODS Symposium*, Tucson, AZ, USA, 1997.

[BCG01] N. Bruno, S. Chaudhuri, and L. Gravano. Stholes: A multidimensional workload-aware histogram. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, USA, 2001.

[Ben75] J. L. Bently. Multidimensional binary search tree used for associative searching. *Communications of ACM*, 18(9):509–517, 1975.

[BF79] J. L. Bently and J. H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, 1979.

[BF95] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB), pages 299-310*, Zurich, Switzerland, 1995.

[BK00] C. Böhm and H.-P. Kriegel. Dynamically optimizing high-dimensional index structures. In *Proceedings of the 7th International Conference on Extending Database Technology (EDBT)*, Konstanz, Germany, 2000.

[BKK96] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, Bombay, India, 1996.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Atlantic City, NJ, USA, 1990.

[BM72] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972.

[CCR98] L. Chen, R. Choubey, and E. A. Rundensteiner. Bulk-insertions into R-trees using the Small-Tree-Large-Tree approach. In *Proceedings of the ACM GIS Conference*, Washington, DC, USA, 1998.

[Com79] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[CR94] C. Chen and N. Roussoupoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Minneapolis, MA, USA, 1994.

[dMOG02] M. deBerg, Hammar M, M. H. Overmars, and J. Gudmundsson. On r-trees with low stabbing number. *Computational Geometry - Theory and Applications*, 24(3):179–195, 2002.

[FB74] R. Finkel and J. Bently. Quad-trees: A data structure for retrieval on composite keys. *ACTA Informatica*, 4(1):1–9, 1974.

[FB93] C. Faloutsos and P. Bhagwat. Declustering using fractals. *PDIS, Journal of Parallel and Distributed Information Systems*, pages 18–25, 1993.

[FJL+01] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithm and implementation for very fast publish/subscribe systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, USA, 2001.

[FR91] C. Faloutsos and Y. Rong. DOT: A spatial access method using fractals. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Kobe, Japan, 1991.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[GLL98a] Y. Garcia, M. Lopez, and S. Leutenegger. A greedy algorithm for bulk loading R-trees. In *Proceedings of the 6th ACM GIS Conference*, Washington, DC, USA, 1998.

[GLL98b] Y. Garcia, M. Lopez, and S. Leutenegger. On optimal node splitting for R-trees. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB), pages 334-344*, New York, NY, USA, 1998.

[GLR00] V. Ganti, M. Lee, and R. Ramakrishnan. Icicles: self-turning samples for approximate query answering. In *Proceedings of the 26nd International Conference on Very Large Data Bases (VLDB)*, Cairo, Egypt, 2000.

[GM98] P. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Seattle, WA, USA, 1998.

[GS04] T. Gevers and A. W. M. Smeulders. Content-based image retrieval: An overview. In G. Medioni and S. B. Kang, editors, *Emerging Topics in Computer Vision*. Prentice Hall, 2004.

[Gün89] O. Günther. The cell tree: An object-oriented index structure for geometric databases. In *Proceedings of the 5th International Conference on Data Engineering (ICDE), pages 598-605*, Los Angeles, CA, USA, 1989.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, 47-57*, 1984.

[HLL01] P. W. Huang, P. L. Lin, and H. Y. Lin. Optimizing storage utilization in R-tree dynamic index structure for spatial databases. *Journal of Systems and Software*, 55:292–299, 2001.

[HS92] P. Haas and A. Swami. Sequential sampling procedures for query size estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, CA, USA, 1992.

[HSW89] A. Henrich, H.-W. Six, and P. Widmayer. The LSD-tree: Spatial access to multidimensional point and non-point objects. In *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB), pages 45-53*, Amsterdam, The Netherlands, 1989.

[Int05] Internet World Stats. World Internet usage and population statistics. On-line source: http://www.internetworldstats.com, July 2005.

[IP95] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose, CA, USA, 1995.

[JOT+05] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)*, 30(2):364–397, 2005.

[KF93] I. Kamel and C. Faloutsos. On packing R-trees. In *Proceedings of the 2nd CIKM Conference, pages 490-499*, Washington, DC, USA, 1993.

[KF94] I. Kamel and C. Faloutsos. Hilber R-tree: An improved R-tree using fractals. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, Santiao, Chile, 1994.

[KS97] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997.

[LEL97] S. Leutenegger, J. M. Edgington, and M. A. Lopez. STR - a simple and efficient algorithm for R-tree packing. In *Proceedings of the 13th International Conference on Data Engineering (ICDE), pages 497-506*, Birmingham, England, 1997.

[LJ04] H. Liu and H. A. Jacobsen. Modelling uncertainties in publish/subscribe systems. In *Proceedings of the 20th International Conference on Data Engineering (ICDE)*, Boston, MA, USA, 2004.

[LS90] D. Lomet and B. Salzberg. The hB-tree: A multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, 1990.

[LS02] F. Llirbat and C.-A. Saita. RVA-clustering: An approximation-based indexing approach for multidimensional objects. In *Technical Report, No. 4670, INRIA-Rocquencourt, France*, December 2002.

[MHN84] T. Matsuyama, L. V. Hao, and M. Nagao. A file organization for geographic information systems based on spatial proximity. *International Journal of Computer Vision, Graphics and Image Processing*, 26(3):303–318, 1984.

[MNPT03] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis. R-trees have grown everywhere. *Submitted to ACM Computing Surveys*, 2003.

[NHS84] J. Nievergelt, H. Hinterberger, and K. Sevcik. The grid file: An adaptable symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.

[OSDM87] B. C. Ooi, R. Sacks-Davis, and K. J. McDonell. Spatial k-d-tree: An indexing structure mechanism for spatial databases. In *Proceedings of the IEEE COMPSAC Conference*, 1987.

[Per02] J. Pereira. *Algorithmes de filtrage efficace pour les systèmes de diffusion d'information à base de notifications*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, September 2002.

[PSW95] B.-U. Pagel, H.-W. Six, and M. Winter. Window query-optimal clustering of spatial objects. In *Proceedings of the ACM PODS Conference*, San Jose, CA, USA, 1995.

[RL85] N. Roussopoulos and D. Leifker. Direct spatial search on pictorial databases using Packed R-Trees. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1985.

[Rob81] J. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1981.

[RSV01] P. Rigaux, M. Scholl, and A. Voisard. *Spatial databases with application to GIS*. Morgan Kaufmann, 2001.

[Sag94] H. Sagan. *Space-filling curves*. Springer-Verlag, New York, 1994.

[Sam84] H. Samet. The quadtree and related hierarchical data structure. *ACM Computing Surveys*, 16(2):187–260, 1984.

[Sam89] H. Samet. *The design and analysis of spatial data structures*. Reading, MA: Addison-Wesley, 1989.

[SC00] T. Schrek and Z. Chen. Branch grafting method for R-tree implementation. *Journal of Systems and Software*, 53:83–93, 2000.

[SL04] C.-A. Saita and F. Llirbat. Clustering multidimensional extended objects to speed up execution of spatial queries. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT)*, Heraklion, Crete, Greece, 2004.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloustos. The R+-tree: A dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, Brighton, England, 1987.

[SYUK00] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, Cairo, Egypt, 2000.

[TP02] Y. Tao and D. Papadias. Adaptive index structures. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, Hong Kong, China, 2002.

[vSW97] J. van den Bercken, B. Seeger, and P. Widmayer. A general approach to bulk loading multidimensional index structures. In *Proceedings of the 23th International Conference on Data Engineering (ICDE)*, Athens, Greece, 1997.

[WAE01] Y. Wu, D. Agrawal, and A. El Abaddi. Applying the golden rule of sampling for query estimation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Santa Barbara, CA, USA, 2001.

[WAE02] Y. Wu, D. Agrawal, and A. El Abaddi. Query estimation by adaptive sampling. In *Proceedings of the International Conference on Data Engineering (ICDE)*, San Jose, CA, USA, 2002.

[WB97] R. Weber and S. Blott. An approximation based data structure for similarity search. In *Technical Report 24, ESPRIT Project HERMES (No. 9141)*, 1997.

[WJ96] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proceedings of the 12th International Conference on Data Engineering (ICDE)*, New Orleans, LA, USA, 1996.

[WSB98] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, New York, NY, USA, 1998.

[Yu02] Cui Yu. *High-dimensional indexing. Transformational approaches to high-dimensional range and similarity searches.* Lecture Notes in Computer Science, LNCS 2341, Springer, 2002.

Abstract We propose a cost-based query-adaptive clustering solution for multidimensional objects with spatial extents to speed-up execution of spatial range queries (e.g., intersection, containment). Our work was motivated by the emergence of many SDI applications (Selective Dissemination of Information) bringing out new real challenges for the multidimensional data indexing. Our clustering method aims to meet several application requirements such as scalability (many objects with many dimensions and with spatial extents), search performance (high rates of spatial range queries), update performance (frequent object insertions and deletions), and adaptability (to object and query distributions and to system parameters). In this context, the existing indexing solutions (e.g., R-trees) do not efficiently cope with most of these requirements. Our object clustering drops many properties of classical tree-based indexing structures (tree height balance, balanced splits, minimum object bounding) in favor of a cost-based clustering strategy. The cost model takes into account the performance characteristics of the execution platform and relies on both data and query distributions to improve the average performance of spatial range queries. Our object clustering is based on grouping spatial objects with similar intervals (positions and extents) in a reduced subset of dimensions, namely the most selective and discriminatory ones relative to the query distribution. The practical relevance of our clustering approach was demonstrated by a series of experiments involving large collections of multidimensional spatial objects and spatial range queries with uniform and skewed distributions.

Keywords*: multidimensional indexing, multidimensional extended objects, clustering, spatial range queries, query-adaptive cost model*

Resumé Nous proposons une méthode de groupement en clusters d'objets multidimensionnels étendus, basée sur un modèle de coût adaptatif aux requêtes, pour accélérer l'exécution des requêtes spatiales de type intervalle (e.g., intersection, inclusion). Notre travail a été motivé par l'émergence de nombreuses applications de dissémination sélective d'informations posant de nouveaux défis au domaine de l'indexation multidimensionnelle. Dans ce contexte, les approches d'indexation existantes (e.g., R-trees) ne sont pas adaptées aux besoins applicatifs tels que scalabilité (beaucoup d'objets avec des dimensions élevées et des extensions spatiales), performance de recherche (taux élevés de requêtes), performance de mise à jour (insertions et suppressions fréquentes d'objets) et adaptabilité (à la distribution des objets et des requêtes, et aux paramètres systèmes). Dans notre méthode, nous relâchons plusieurs propriétés spécifiques aux structures d'indexation arborescentes classiques (i.e. équilibrage de l'arbre et du partitionnement, englobement minimal des objets) en faveur d'une stratégie de groupement basée sur un modèle de coût adaptatif. Ce modèle de coût tient compte des caractéristiques de la plateforme d'exécution, de la distribution spatiale des objets et surtout de la distribution spatiale des requêtes. Plus précisément, la distribution des requêtes permet de déterminer les dimensions les plus sélectives et discriminantes à utiliser dans le regroupement des objets. Nous avons validé notre approche par des études expérimentales de performance impliquant de grandes collections d'objets et des requêtes d'intervalles avec des distributions uniformes et non-uniformes.

Mots clé *: indexation multidimensionnelle, objets multidimensionnels étendus, groupement, requêtes spatiales de type intervalle, modèle de coût adaptatif aux requêtes*