Design and Implementation of On-Line Analytical Processing (OLAP) of Spatial Data

by

Nebojša Stefanović

B.Sc., University of Belgrade, Yugoslavia, 1993

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the Department
of
Computing Science

SIMON FRASER UNIVERSITY
September 1997

APPROVAL

Name: Nebojša Stefanović

Degree: Master of Science

Title of thesis: Design and Implementation of On-Line Analytical Processing (OLAP) of Spatial Data

Examining Committee: Dr. Binay Bhattacharya
Chair

Dr. Jiawei Han
School of Computing Science
Senior Supervisor

Dr. Tiko Kameda
School of Computing Science
Supervisor

Dr. Qiang Yang
School of Computing Science
SFU External Examiner

Date Approved: ______________________

Abstract

On-line analytical processing (OLAP) has gained its popularity in database industry. With a huge amount of data stored in spatial databases and the introduction of spatial components to many relational or object-relational databases, it is important to study the methods for spatial data warehousing and on-line analytical processing of spatial data. This thesis investigates methods for spatial OLAP, by integration of nonspatial on-line analytical processing (OLAP) methods with spatial database implementation techniques. A spatial data warehouse model, which consists of both spatial and nonspatial dimensions and measures, is proposed. Methods for computation of spatial data cubes and analytical processing on such spatial data cubes are studied, with several strategies proposed, including approximation and partial materialization of the spatial objects resulting from spatial OLAP operations. Some techniques for selective materialization of the spatial computation results are worked out, and the performance study has demonstrated the effectiveness of these techniques. Spatial OLAP has been partially implemented as a part of GeoMiner, a system prototype for spatial data mining.

Keywords: Data warehouse, data mining, on-line analytical processing (OLAP), spatial databases, spatial data analysis, spatial OLAP.

Acknowledgments

I would like to thank my senior supervisor, professor Dr. Jiawei Han, for introducing me to data mining and data warehousing concepts and for directing my research. I am very grateful for many inspiring discussions and for confidence that he has in me. He has been available and helpful throughout the preparation of this thesis, and his support and supervision have been invaluable. Thanks also goes to my supervisor, professor Dr. Tiko Kameda, and external examiner, professor Dr. Qiang Yang, for reading this thesis and making very valuable suggestions.

Additionally, I would like to thank Krzysztof Koperski for very thoughtful comments and suggestions that strengthened and focused my research work. Moreover, it was great fun working with him on the design and implementation of the GeoMiner system prototype.

I would like to thank my parents and my brother Veljko for their love, encouragement and support. The potpourri of my gratitude, appreciation, and feelings are more than words can say. Not even thousands of kilometers could ever make a gap between us. Many thanks go to my uncle and aunt for their generosity and heart-warming acceptance to their family. If it were not for them, it is unlikely that I would have come to Canada and studied at Simon Fraser University.

Finally, I wish to thank Ya Ling (Donna) Hsiao whose emotional support sustained me through periods of loneliness and self-doubt. Without her trust, love, and care, I would have never overcome numerous problems that I came across. She has been my greatest inspiration and I am delighted that she knows that.

Dedication

To Mom, Dad, and Veljko.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the rapid growth of enterprise data, it is essential to develop techniques that summarize such voluminous data. In the last few years, there has been a substantial effort on creation, usage, and maintenance of data warehouses. A data warehouse is designed to manage large volumes of business data and to provide a foundation for analytical processing. It can be defined as a subject-oriented, integrated, time varying, non-volatile collection of data that is used primarily in organizational decision making [39]. The goal of data warehouses is to provide a single image of business reality for the organization. Typically, data warehouse systems consist of a set of programs that extract data from the operational environment, a repository that maintains the warehouse data, and systems that provide information to users.

In this chapter we give a brief summary of the conventional OLAP and stress the importance of its extension for handling spatial data. Then, we present the outline of this thesis.

1.1 On-Line Analytical Processing

The notion of extracting useful knowledge from collected data is not a new idea in information systems technology. Only with the explosive growth of the quantity of data has it become crucial to systematically examine techniques for data analysis. Many organizations possess a wealth of data that is maintained, and stored, but they are unable to capitalize on the nuggets of information hidden in the data. The primary goal of data warehouses is to improve quality of decision making process in the enterprise. Years of research have produced state-of-the-art technology. Furthermore, in parallel with the investigation into design of data warehouses, various techniques for analyzing large amounts of data have been proposed. Accordingly, a new term *OLAP (On-Line Analytical Processing)* [11] was coined.

The data warehousing supports on-line analytical processing (OLAP), the functional and performance requirements of which are very different from those of on-line transaction processing (OLTP) applications traditionally supported by the operational databases. Whereas transaction processing systems are judged on their ability to collect and manage data, analytical processing systems are judged on their ability to extract information from data. These two types of data processing differ in a number of aspects, and the differences are summarized below.

- users
  While OLTP is performed mainly by clerks, OLAP is used by management people in a decision support process.

- data
  Data in OLTP is current, accurate, and very detailed. In contrast, data stored in data warehouses and manipulated by OLAP is historical, multidimensional and often summarized.

- unit of operation
  Transactions in OLTP are usually short SQL statements, as opposed to OLAP where a knowledge worker deals with very complex nested queries. This creates a necessity for a flexible user interface and even more importantly for an efficient query optimization.

- number of accessed records
  In most cases the number of records accessed by an OLAP server is at least by an order of magnitude larger than in the case of OLTP.

- metrics

  Transaction throughput is the main performance indicator in OLTP applications; however, query throughput and response time are critical for OLAP applications. Only if response time is adequate (a few seconds) can OLAP be fruitful and appealing for data analysts.

All these characteristics are strong arguments for physical separation of data warehouses from operational data. Moreover, it is often the case that data warehouses contain data consolidated from heterogeneous sources including legacy data. The different sources may contain data of varying quality, and/or may use inconsistent representations, codes, formats, which have to be reconciled. In most cases OLTP is developed using E-R model [10], that is application-oriented. Such a model cannot effectively serve for decision support, since a database model for OLAP is to be subject-oriented. *Star* and *snowflake* schemas [8, 39, 46] have emerged as the main candidates for models for efficient OLAP.

Two basic OLAP operations are *roll-up* (decreasing the level of details) and *drill-down* (increasing the level of details) along one or more dimensions. Roll-up/drill-down is often considered as a process of ascending/descending concept hierarchies. A concept hierarchy provides valuable information for inductive learning. It is related to a specific attribute in a database and is partially ordered according to general-to-specific ordering [32]. However, above OLAP operations are not necessarily associated with the existence of hierarchies. To solve this ambiguity, a concept *any* is introduced for each dimension. Rolling-up a dimension to *any* is equal to dropping it. If selection and projection are applied together with a drill-down operation, one gets *slice-and-dice* OLAP operation. Finally, the operation that changes the orientation of a multidimensional view of data is known as *pivoting*.

Data warehouses can be implemented on standard or extended relational DBMSs, called Relational OLAP (ROLAP) servers [8]. These servers assume that data is stored in relational databases, and they support extensions to SQL that facilitate implementation of a multidimensional data model. In contrast, Multidimensional OLAP (MOLAP) servers directly store multidimensional data in a special data structures (e.g., multidimensional arrays) and implement the OLAP operations over these data structures [8].

Defining a schema and selecting an OLAP server is only one step in the process of building and maintaining a data warehouse. It is very important that the architecture which fits the needs of knowledge workers be chosen carefully. Ideally, creating an integrated enterprise warehouse that collects information about all subjects (e.g., customers, products, revenues, personnel) spanning the whole organization would be the best choice. The problem is that building such a warehouse is a long and a complex process. Many different kinds of metadata, including *administrative*, *business*, and *operational* metadata, have to be managed. Consequently, many organizations are settling for *data marts* instead. A data mart in an integrated data resource is a subset of the data resource, usually oriented to a specific purpose or major data subject, that may be distributed to support local business needs [5]. Data marts enable faster roll out, since they do not require the enterprise-wide consensus, but they may lead to complex integration problems in the long run [8].

1.2 Motivations for Spatial OLAP

Being recognized as a crucial task in information technology, the OLAP phenomenon has become interesting from both an academic and an industrial point of view. We are witnessing a tremendous burst of OLAP-related research activity [8, 59, 61, 62, 65]. However, the research interests have been mainly directed towards OLAP of relational data, while neglecting the importance of consolidating, integrating and summarizing other types of more complex data.

With the popular use of satellite telemetry systems, remote sensing systems, medical imaging, and other computerized data collection tools, a huge amount of spatial data has been stored in spatial databases, geographic information systems, spatial components of many relational or object-relational databases, and other spatial information repositories. It is an imminent task to develop efficient methods for the analysis and understanding of such huge amount of spatial data and utilize them effectively [62].

Following the trend of the development of data warehousing and data mining techniques [8, 22, 40, 41, 46, 61, 65], we propose to construct *spatial data warehouses* to facilitate on-line spatial data analysis and spatial data mining [17, 18, 19, 21, 36, 45, 48, 49, 53, 54]. Similar to nonspatial data warehouses [8, 39, 41, 61, 65], we consider that a *spatial data warehouse* is a *subject-oriented, integrated, time-variant*, and *non-volatile* collection of both spatial and nonspatial data in support of management's decision making process.

In this thesis, we study how to construct a spatial data warehouse and how to implement efficiently *on-line analytical processing of spatial data* (i.e., *spatial OLAP*) in such a warehouse environment. To motivate our study of spatial data warehousing and spatial OLAP operations, we examine the following application examples.

Example 1 Regional weather pattern analysis

There are about 3,000 weather probes scattered in British Columbia, each recording daily temperature and precipitation for a designated small area and transmitting signals to a provincial weather station. A user may like to view weather patterns on a map by month, by region, and by different combinations of temperature and precipitation, or may even like to dynamically drill-down or roll-up along any dimension to explore desired patterns, such as wet and hot regions in Fraser Valley in July, 1997. □

Example 2 Overlay of multiple thematic maps

There often exist multiple thematic maps in a spatial database, such as altitude map, population map, and daily temperature maps of a region. By overlaying multiple thematic maps, one may find some interesting relationships among altitude, population density and temperature. For example, *flat low land in B.C. close to the coast is characterized by mild climate and dense population.* One may like to perform data analysis on any selected dimension, such as drill down along a region to find the relationships between altitude and temperature. □

Example 3 Maps containing objects of different spatial data types

Maps may contain objects with different spatial data types. For example, one map could contain highways and roads of a region, the second about sewage network, and the third about the altitude of the region. To choose an area for housing development, one should consider many factors, such as road network connection, sewage network connection, altitude, etc. One may like to drill-down and roll-up along some dimension(s) in a spatial data warehouse which may require overlay of multiple thematic maps of different spatial data types, such as regions, lines, and networks. □

The above examples show some interesting applications of spatial data warehouses but also indicate that there are many challenging issues in implementing spatial data warehouses.

The first challenge is the construction of spatial data warehouses by integration of spatial data from heterogeneous sources and systems. Spatial data is usually stored in different industrial firms and government agencies using different data formats. Data formats are not only structure-specific (e.g., raster- vs. vector-based spatial data, object-oriented vs. relational models, different spatial storage and indexing structures, etc.), but also vendor-specific (e.g., ESRI, MapInfo, Intergraph, etc.). Moreover, even with a specific vendor like ESRI, there are different formats like Arc/Info and ArcView (shape) files. There have been a lot of work on data integration and data exchange. In this thesis, we are not going to address data integration issues and we assume that a spatial data warehouse can be constructed either from a homogeneous spatial database or by integration of a collection of heterogeneous spatial databases with data sharing and information exchange using some existing or future techniques. Methods for incremental update of such spatial data warehouses to make it consistent and up-to-date will not be addressed in this thesis either.

The second challenge is the realization of fast and flexible on-line analytical processing in a spatial data warehouse. This is the theme of our study.

In spatial database research, spatial indexing and accessing methods have been studied extensively for efficient storage and access of spatial data [15, 26, 30, 58]. Unfortunately, these methods alone cannot provide sufficient support for on-line analytical processing of spatial data because spatial OLAP operations summarize and characterize a large set of spatial objects in different dimensions and at different levels of abstraction, which requires fast and flexible presentation of collective, aggregated, or general properties of spatial objects. New models and techniques should be developed for on-line analysis of voluminous spatial data.

In this thesis, we propose the construction of a spatial data warehouse using a *spatial data cube* model (also called a *spatial multidimensional database* model). A *star/snowflake model* is used to model a spatial data cube which consists of spatial dimensions and/or measures together with nonspatial ones. Methods for efficient implementation of spatial data cubes are examined with some interesting techniques proposed, especially on precomputation and selective materialization of spatial OLAP results.

We will show that the precomputation of spatial OLAP results (i.e., spatial measures), such as merge of a number of spatially connected regions, is beneficial not only for fast response in result display but also, and often more importantly, for further spatial analysis and spatial data mining, such as spatial association, clustering, classification, etc [36].

1.3 The role of Spatial OLAP in Spatial Data Mining

Spatial data mining, i.e., knowledge discovery from large amounts of spatial data, is a highly demanding field because voluminous data have been collected in various applications, including remote sensing, medical imaging, environmental assessment and planning, geographical information systems (GIS), etc [50]. Moreover, most of business data contains, at least implicitly, spatial dimension (e.g., postal code) that can be easily geocoded.

Since most data mining systems can work with data stored in flat files or operational databases, neither a data warehouse nor OLAP is required. Yet, mining a data warehouse usually results in better information, because data is usually cleansed before being stored there. Furthermore, one of the premises for fruitful data mining is to be able to perform it at different levels of abstraction. Interactive approach in knowledge discovery is reflected by frequent usage of various OLAP operations. When integrated with data mining modules such as associator, classifier, or clustering module, the OLAP engine can serve as a backbone of an interesting and powerful data mining system [35, 36]. Thus, we see spatial OLAP as a prerequisite for spatial data mining. However, the linkage between OLAP and data mining is not one-directional.

Dealing with large volumes of data involves a high probability of having errors. Errors, both spatial and nonspatial, are often caused by inconsistent field lengths, inconsistent value alignments, inconsistent descriptions, missing entries, and violation of integrity constraints. In all these cases, data cleaning is an absolutely necessary step in the process of building a data warehouse. Data cleaning is a problem that is reminiscent of heterogeneous data integration, a challenging problem that has been studied for years. But here the emphasis is on data inconsistencies rather than schema inconsistencies [8]. Data cleaning is much more than simply updating a record with the correct data because the detection of errors is a crucial part in this process. Although the data cleaning process can hardly be fully automated, by using data mining tools such as clustering, trend, or deviation analysis one can identify data anomalies. After detected errors are corrected, one may proceed with a creation of a data warehouse.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 contains a review of previous related work on data warehousing, OLAP, and spatial data mining. Chapter 3 describes the model of a spatial data warehouse. The emphasis is put on spatial measures and their materialization. Chapter 4 addresses challenges for selective materialization of spatial measures and presents three algorithms. Chapter 5 presents the OLAP component of the GeoMiner system and the performance study of the proposed algorithms. Finally, Chapter 6 summarizes this study and discusses the future research issues.

Chapter 2

Related Work

In this chapter we briefly summarize previous work that has influenced our design and implementation of spatial OLAP. It includes techniques for the design and implementation of data warehouses, and the work on spatial data mining.

2.1 Logical Design of a Data Warehouse

The fundamental characteristic of the data warehouse technology is its multidimensional paradigm. In contrast, operational data is mainly stored in a form of flat relational tables. Thus, a new multidimensional model creates a number of challenges.

Most data warehouses are built using a *star schema* (also called *star join schema*) to represent the multidimensional data model [8, 39, 46, 61]. The reason behind adopting the name star schema is quite clear: the database contains a central *fact table* and a number of radially organized *dimension tables*. While the fact table is large, especially in terms of the number of tuples, the dimensional tables are usually relatively small. This asymmetric architecture is very different from what the entity-relationship model is built on. Each tuple in the fact table contains a pointer, in the form of a foreign key, to each of the dimension tables. On the other hand, each dimension table consists of columns that represent attributes of the dimension.

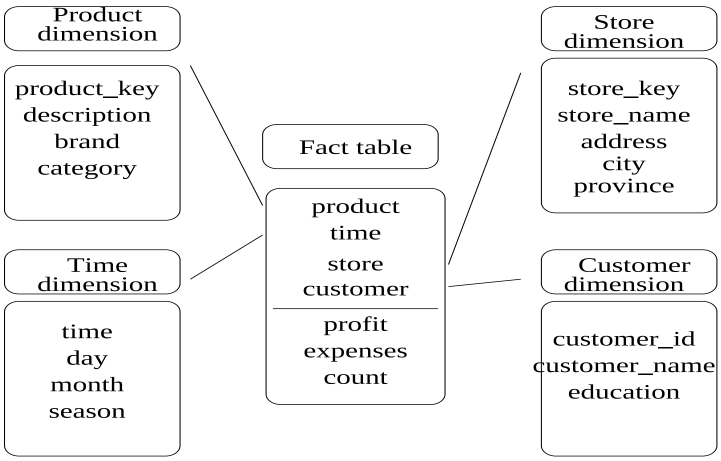

Figure 2.1: A star schema

These attributes may or may not correspond to the concept hierarchy of the dimension. An example of a star schema architecture is shown in Figure 2.1. While the fact table is highly normalized, the attendant dimension tables are kept denormalized. For example, *Store* dimension table is denormalized since the following functional dependency [16, 64] holds: $city \rightarrow province$ (violating the third normal form). This table can be normalized, by eliminating column *province* and creating another table that contains only *city* and *province* columns. *Store* table may contain tuples like {#6445, Sears, 5483 Main Street, Vancouver, B.C.} and {#4485, Eatons, 2304 First Avenue, Vancouver, B.C.}. The normalized table would contain tuples: {#6445, Sears, 5483 Main Street, Vancouver} and {#4485, Eatons, 2304 First Avenue, Vancouver}, and the additional dimension table would contain tuple {Vancouver, B.C.}

In addition to dimensions, the star schema collects measures of the business. Since the main motivation for the whole data warehouse technology is to enhance decision support process, obtaining useful measures presents a pivotal issue. All measures of the business are stored in the fact table. $Profit$, $expenses$, and $count$ are measures shown in Figure 2.1.

Due to their denormalized dimension tables, star schemas do not provide an explicit support for concept hierarchies, so that many enterprise data warehouses are

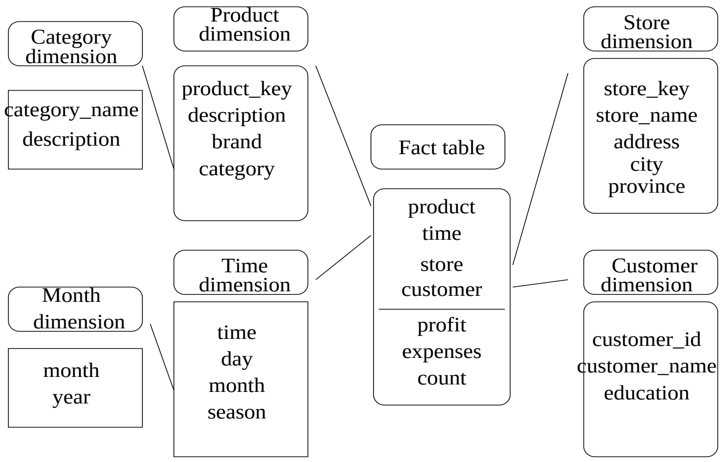

Figure 2.2: A snowflake schema

designed using a *snowflake schema*, shown in Figure 2.2, where some or all dimension tables are fully normalized. It results in advantages in schema maintenance and handling of concept hierarchies. However, the denormalized structure of the dimension tables in a star schema offers easier browsing of the dimensions. For this reason Kimball, one of the leading experts in data warehouse technology, strongly discourages using a snowflake schema. "The dimension tables must not be normalized but should remain as flat tables. Normalized dimensions destroy the ability to browse" [46]. The main reason for normalization of attendant dimension tables is to minimize storage needs, albeit, the amount of storage used by all dimension tables is negligible comparing to the storage used by the fact table [46].

Finally, some data warehouses are built around *fact constellations*, a complex structure in which multiple fact tables share dimension tables [8]. For instance, in order to keep track of both *projected profit* and *actual profit* one may form a fact constellation because many dimensions are shared by two fact tables.

Using any of the above structures for modeling multidimensional data leads to large benefits during decision support process. However, these models have certain limitations in handling spatial data. Although many existing applications deal with dimensions that contain spatial information (e.g., *Store dimension, Geography*), spatial objects are not considered. Introducing spatial objects creates a number of challenges. New types of dimensions and measures have to be added to the model. Consequently, techniques for creating, using and maintaining a spatial data warehouse will significantly differ from those for a traditional (nonspatial) data warehouse. The detailed discussion on the necessary extensions is presented in Chapter 3.

2.2 Physical Design of a Data Warehouse

2.2.1 Architecture of OLAP Servers

Clearly, there are two major directions in the implementation of a data warehouse, namely MOLAP and ROLAP, and most corporations have followed one of the approaches, or a mixture of both. ROLAP servers extend traditional relational servers with specialized middleware to efficiently support multidimensional OLAP queries, and they are typically optimized for specific back-end relational DBMS servers. The main strength of ROLAP is in exploiting the scalability, reliability and the transactional features of relational systems. However, the mismatch between OLTP and OLAP style querying may present the bottleneck for ROLAP servers.

MOLAP servers provide a direct support for a multidimensional view of data. This approach has the advantage of excellent indexing properties, but often suffers from poor space utilization especially when data is sparse. A number of techniques for handling sparse data have been proposed. We will give a short overview of the most often used indexing methods in the following subsection. The current state of MOLAP is very chaotic and there are several reasons for that:

- Unlike in the relational model, there is no standard multidimensional model.
- There are no standard access methods or API's.
- The products range from narrow to broad in addressing the aspects of decision support.
- Companies do not reveal their design strategies.

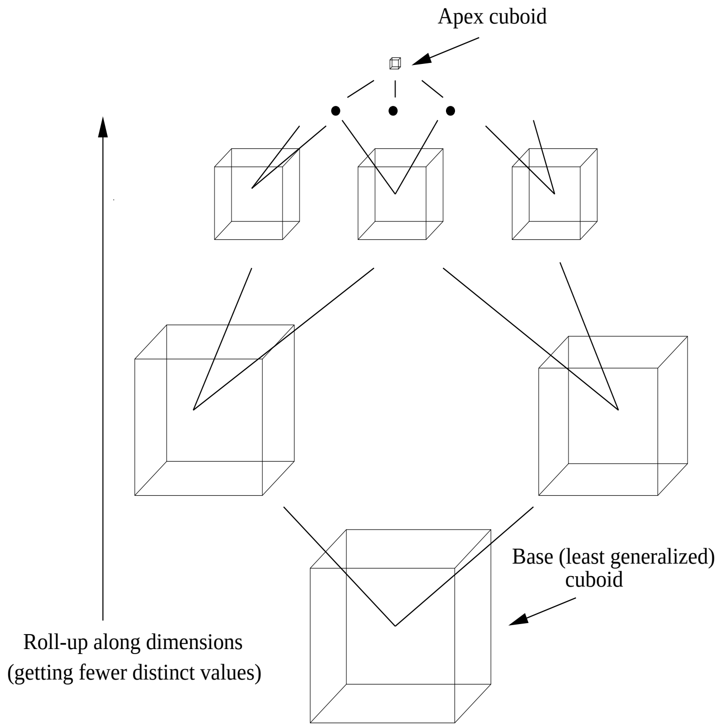

Figure 2.3: A data cube

There is an on-going debate about advantages of MOLAP over ROLAP and vice versa. Two approaches are compared in [67], with MOLAP getting a significant edge. In the same paper, the authors proposed an efficient algorithm that first converts the relational table into an array, cube the array, and then convert the results back to a relational table.

2.2.2 Materialization of Views

One of distinctive characteristics of OLAP operations is that the queries deal with summarized data, or aggregates. Hence, materialization of summary data can accelerate many common queries. Multidimensional aggregates, that serve as measures of the business, are usually stored in a *data cube*.

A data cube consists of a number of *views*. For that reason, views are often called

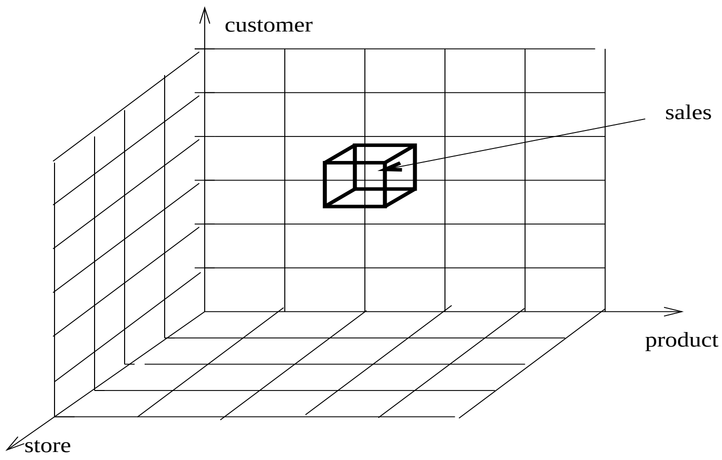

Figure 2.4: A cuboid

subcubes, or *cuboids*. In the rest of this thesis we will use these terms interchangeably. A lattice of cuboids that constitute one data cube is shown in Figure 2.3. A 3-dimensional cuboid is shown in Figure 2.4. In this cuboid, *product*, *store*, and *customer* are dimensions while *sales* is a measure. Roll-up/drill-down along any of the dimensions leads to decrease/increase in the the number of distinct values for the dimension. However, a cuboid is associated with a single level of concept hierarchies for all dimensions. Note that some dimensions may be at concept *any*, i.e, dropped dimension. In order to explain the meaning of a measure, we for the moment ignore the concept hierarchies for dimensions. If there were not the concept hierarchies, cell *sales* would contain the total sales of a particular *product* sold in a particular *store* and to a particular *customer*. The existence of concept hierarchies means that the dimension values can be at abstraction levels higher that that of raw data (particular product ids, store ids, or customer names). Each cell in the cuboid corresponds to one combination of dimension values. The main incentive for materialization is to shorten on-line processing time, a crucial metric for data warehouse performance. Suppose that the cuboid (view) shown in Figure 2.4 is materialized, that is all measures (cells) are computed off-line. Then, answering an OLAP query that asks for this cuboid would need no scan of the database table. The major challenges in exploiting materialized views are listed in [8] as follows.

- Identifying the views to materialize

  There are three obvious approaches to materialization of cuboids: (1) materialize all cuboids (2) materialize none of the cuboids (3) materialize only selected cuboids. While the first approach suffers from the explosion of consumed space, the second one results in a slow response time. Thus, selective materialization seems as the only reasonable solution.

  Harinarayan, Rajaraman, and Ullman proposed a scalable greedy algorithm [38] that was shown to have a good performance. In the rest of this thesis we will refer to this algorithm as HRU algorithm. The algorithm recognizes that cuboids can be organized in a hierarchical lattice structure. Accordingly, it uses the dependency relationship among cuboids to determine which cuboids should be selected for materialization in the preprocessing phase. The objective of the algorithm is to minimize the average time taken to evaluate a view (cuboid) while materializing a fixed number of views, regardless of the space they use. The authors assume the cost of answering a query is proportional to the number of rows examined. Then, they observe that a view containing dimensions A and B can be computed using view A, B, and C without scanning the original relational table. Thus the problem is to select a set of optimal views that leads to the minimal average time taken to evaluate any view. The authors show that the problem is intractable and propose a greedy algorithm. In each round, the algorithm chooses a view to materialize considering what was materialized in earlier rounds. The benefit of materialization is defined as the decrease in the number of rows to be scanned. For detailed explanation of this heuristic, we refer to [38]. The total benefit of the algorithm is at least 63% of the benefit of the optimal algorithm.

  Later, the same research group augmented this algorithm by proposing a set of greedy algorithms that select cuboids and indices in parallel [29]. These algorithms have granularity on the cuboid level. We will show that this characteristic makes the algorithms inadequate for applying on databases with complex data types, such as spatial data. Furthermore, the algorithms do not take access frequency into account.

- Exploiting the materialized views to answer OLAP queries
  It is very important to make the precomputed aggregates transparent to the user of the OLAP engine. In other words, the user should pose queries to base tables rather than to aggregates. On-line processing should take as much benefit as possible from precomputed aggregations. In general, there may be several candidates that can be used in answering a query, and it is a non-trivial task to determine which of the candidate(s) is/are most suitable for the query. Some work on this problem have been reported in [9, 27, 52, 63].

- Efficient updating of materialized views
  Although OLAP applications are mainly read-only, all materialized views have to be precomputed when a new batch of data is loaded. There are two main streams in this area of research: updating the values of precomputed aggregations, and updating the schema. Arguably, incremental update is the only reasonable approach. There has been a lot of on-going research work in addressing this issue [2, 28, 57].

2.2.3 Indexing of OLAP Data

Speeding up the access to data is often a critical concern for relational DBMS. The proliferation of end user-oriented tools, the availability of sophisticated applications for relational DBMSs, and especially the growing interest in data warehousing and on-line analytical processing (OLAP) applications have contributed to the complexity of workloads that today's databases must support. OLAP applications require viewing data from many different angles (dimensions). It is crucial that these applications have fast interactive response time to a variety of large aggregate queries on huge amounts of data. Techniques for deciding which aggregate views to materialize certainly improve response time to OLAP queries, without introducing a significant performance degradation due to storage overhead. However, only if the indexing structure is adequate can one fully exploit the benefits of OLAP applications. According to [59], existing indexing methods in OLAP data can be classified into the following four classes.

- Multidimensional array-based methods
  Arguably, the most natural indexing schema for the OLAP data cube is a *multidimensional array*. This would be the ideal model, if the data cube were dense. However, in most applications with large number of dimensions, the cube sparsity is a huge problem. Typically, only 20% of data in the data cube is non-zero [12]. An interesting solution for handling sparse data is used in *Arbor Essbase* [13] in which dimensions are divided into *dense* and *sparse* dimensions. An index tree ($B+$ tree) is formed using combinations of values for sparse dimensions. Leaves of the tree point to multidimensional arrays of dense dimensions. However, with a large number of sparse dimensions and many distinct values for them, the number of leaf nodes in the $B+$ tree grows rapidly. Only if the sparse index fits in the memory can this method produce satisfactory performance. Thus, the success of the above method heavily depends on the ability to find enough dense dimensions. Moreover, only queries that specify values for all sparse dimensions have adequate performance.

- Bitmap indices
  The increased focus on complex queries for data warehousing and OLAP has revived the interest in *bitmap indices*. The basic idea behind a bitmap is to use a single bit (instead of multiple bytes of data) to indicate that a specific value of an attribute is associated with an entity. For example, instead of storing eleven character long string "programmer" as a skill of a particular employee, the skill value is attributed to the employee by using a single bit. The relative position of the bit within the string of bits is mapped to the relevant tuple. Queries can be answered by applying bitwise *OR* operation for different values of the same dimension and bitwise *AND* operation among different dimensions [55]. The major advantages of this technique are that: (1) For low cardinality data, both

storage space and response time are low. (2) Sparse data is handled in the same way as dense data. (3) All dimensions are treated symmetrically. On the other hand, some clear disadvantages of bitmap indices are that: (1) There is increased storage space overhead for storing bitmaps, especially for high cardinality data. (2) Answering range queries may be expensive, because it involves a number of bitwise *OR* operations. (3) Updates are costly, because all indices have to be updated for even a single row insertion. Several approaches for handling high cardinality data have been proposed. Compression of bitmaps indices [23] can significantly reduce storage overhead; however, the efficiency of performing bitwise *AND/OR* operations may deteriorate. Some products [14] use a hybrid approach that combines B-tree with bitmap indices.

- Hierarchical indexing methods
  Hierarchical indexing methods exploit hierarchical nature of data to save space. An interesting study on *cube forests* is presented in [42]. The authors first define a *cube tree* as a tree whose nodes are search structures (e.g., B-trees or multidimensional structures). Each node represents an index on one attribute (or collection of attributes). In order to create a cube tree, attributes have to be ordered and the relational table indexed in such an order. This order defines a *template*. This method favors some queries over others, i.e., queries that form a prefix of the template are answered more quickly than others. In order to overcome this disadvantage, cube trees are organized in cube forests. Details of this study can be found in [42].

- Conventional multidimensional indices
  A number of multidimensional indices have been proposed for handling spatial data [30, 58]. Data structures like R-trees, Quad-trees and their variations are primarily used for two (or three) dimensional data, but they do not scale well if applied to OLAP applications [4]. However, modifications proposed in [60] optimize R-trees for efficient access to OLAP data. The method allows for two types of nodes: (1) rectangular dense regions that contain more points than that specified by a threshold, and (2) points in sparse regions. Note that for dense

regions only boundaries are stored. This method relies on the ability to detect dense regions in the multidimensional space. In most cases, such regions can be identified by domain experts or by using clustering algorithms that retrieve clusters of rectangular shape. R-trees and bitmap indices are compared in [59], and the study shows that R-trees are preferred unless the cardinality is low and data is very sparse.

2.2.4 SQL Extensions

We believe that the success of relational databases [16, 64] should be credited in part to the creation of the standardized relational query language - SQL. Thus, a number of researchers have studied extensions to SQL that would facilitate the expression and processing of OLAP queries. Some extensions as listed in [8] are.

- Extended family of aggregate functions
  Traditional SQL aggregate functions are not sufficient for efficient decision support process. Thus, *rank*, *percentile*, and a number of functions for financial analysis are being added to SQL standard. However, one should note that results of some aggregate functions are more maintainable than those of the others. Accordingly, aggregate functions can be classified into three categories: distributive, algebraic, and holistic [25].

- Multiple Group-By
  OLAP applications require grouping by different sets of attributes. This could be achieved by a set of SQL statements but the data set would be scanned multiple times, that would lead to poor performance. Let us go back to Figure 2.4. The cuboid shown on that figure corresponds to the following SQL query:

  ```
  SELECT      product, store, customer, SUM(sales)
  FROM        Our_sales
  GROUP BY    product, store, customer
  ```

  In addressing the problem of a multiple scan two new operators *Cube* and *Rollup* have been proposed [24]. The Cube operator is the n-dimensional generalization of the group-by operator. It computes group-bys corresponding to all possible combinations of a list of dimensions (attributes). However, in some cases, it is not necessary to compute the full cube. In such cases, Rollup operator that generates only super-aggregates can be used instead. While the order of specified dimensions is irrelevant in the Cube clause, it plays an important role in Rollup clause. Algorithms for efficient implementation of Cube operator are described in [1]. These algorithms extend sort-based and hash-based grouping methods with several optimizations, like combining common operations across multiple group-bys, caching, and using precomputed group-bys for computing other group-bys. Similar techniques could be applied for efficient implementation of Rollup operator.

- Comparisons
  Comparing differences among different portions of data sets is a common operation in decision support process. Although the current version of SQL cannot handle comparisons [47], a recent research paper [7] suggests extensions to SQL that could meliorate this problem. A challenging implementation issue is how to avoid multiple sequential scans of the database tables.

Spatial database systems lack a standardized query language and currently, the most promising option seems to be *Spatial SQL* [15]. Extensions similar to these outlined above would definitely facilitate spatial OLAP.

2.3 Spatial Data Mining

Recent years have seen a rapid progress of research into data mining and data warehousing of relational and transactional data [22]. Similarly, but to a smaller extent, there have been a number of promising results in spatial data mining. Spatial data mining is a subfield of data mining that deals with extraction of implicit knowledge, spatial relationships, or other interesting patterns not explicitly stored in a database [36]. An excellent survey on state-of-the-art work in spatial data mining can be found in [50].

Research into spatial data mining started with a paper by Lu et al. [53]. The authors suggest two methods of generalization, namely *nonspatial-data-dominant generalization* and *spatial-data-dominant generalization*. The algorithms use attribute-oriented induction method [32] to generalize nonspatial dimensions. Spatial generalization is conducted by spatial merging and/or spatial approximations. This study showed the importance of discovery of general knowledge from large spatial databases.

Spatial data clustering has been recognized as a very useful data mining method in recent years. Accordingly, there has been reported substantial amount of research in this field. A distance-based clustering method *CLARANS*, based on randomized search is proposed in [54]. In CLARANS, a cluster is represented by its *medoid*, the most centrally located data point within the cluster. The clustering process is formalized in terms of searching a graph in which each node is a potential solution. Unfortunately, being an I/O extensive algorithm, CLARANS has serious drawbacks with respect to efficiency. The method proposed in [20] augments CLARANS by clustering only a sample of the data set that is drawn from corresponding R^*-tree [3] data pages. While the efficiency is greatly improved, there is no significant degradation of clustering quality. However, the scalability problem gets fully addressed in [66], where the authors propose a distance-based *BIRCH* method. BIRCH makes full use of available memory to derive the finest possible clusters by minimizing I/O costs. It exploits the important observation that the data space is usually not uniformly occupied, and that not every point is equally important for the clustering process. Thus, BIRCH performs well on skewed data and is insensitive to the data input order. Finally, *DBSCAN* clustering method [18] that relies on density-based notion of clusters discovers clusters of arbitrary shapes and handles noise well.

The discovery of association rules from relational and transactional databases has attracted a large number of researchers. As a result, several interesting methods have been proposed. An interesting algorithm suggested in [49] proposes an extension of transaction association rules, by taking into consideration spatial properties of objects in a spatial database. For example, a spatial association rule may show that "golf courses that are *close to* resorts generate large profit in Spring months". Note that spatial predicate *close to* can be defined as a certain distance (e.g., 5 kilometers).

The method explores efficient mining of spatial association rules at multiple approximation and abstraction levels. It consists of two major steps: filtering and refining. Such a two-step method facilitates mining at multiple concept levels by a top-down, progressive deepening technique.

Recently, in [17] the authors introduced a formal framework for spatial data mining by proposing a set of basic operations which should be supported by a spatial database system to express algorithms for knowledge discovery. A concept of neighborhood graphs and paths, together with a small set of operations for their manipulation is introduced. In addition, the authors outline algorithms for spatial classification and spatial trend detection. Furthermore, the authors claim that materialization of neighborhood indices and paths significantly speeds up proposed operations. This claim implicitly suggests the importance of spatial OLAP as a prerequisite step for a fruitful spatial data mining.

Chapter 3

Model of a Spatial Data Warehouse

In this chapter we describe the model of a spatial data warehouse and emphasize its distinctive characteristics from the relational counterpart. We explain the limitations of a conventional (nonspatial) data warehouse in handling spatial data, and stress the importance for its necessary extensions. Consequently, we recognize a need for different algorithms for creation, usage, and maintenance of a spatial data warehouse.

3.1 Logical Design of a Spatial Data Warehouse

To model a spatial data warehouse, the star schema model is still considered to be a good choice because it provides a concise and organized data warehouse structure and facilitates OLAP operations and easy browsing. However, applying the star schema in its original form [8, 39, 46, 61] would lead to inefficiency in performing spatial OLAP. To show the importance of having a different model we revive the examples from Chapter 1.

Example 1 Regional weather pattern analysis

There are about 3,000 weather probes scattered in British Columbia, each recording daily temperature and precipitation for a designated small area and transmitting signals to a provincial weather station. A user may like to view weather patterns on a map by month, by region, and by different combinations of temperature and precipitation, or may even like to dynamically drill-down or roll-up along any dimension to explore desired patterns, such as wet and hot regions in Fraser Valley in July, 1997. □

Here, the obvious question that one may pose is: "Why a conventional data warehouse and conventional OLAP cannot handle on-line analysis described in the above example?". By answering this question, we show which extensions should be added to a conventional data warehouse model.

Each weather station is associated with a region on the map. It is likely that neighboring regions have similar weather patterns. Moreover, when a user rolls-up dimensions, the likelihood of having same descriptions for neighboring regions increases. Consequently, one should get a number of compound (merged) regions. From a decision support perspective, these merged regions are measures of the business. Note that this measure type is very different from those discussed in Chapter 2, such as $profit$, $expenses$ and $count$ since it represents spatial objects. In addition, a user may wish to perform OLAP operations directly on map regions. None of these can be done by applying conventional OLAP methods, because they deal only with measures that are numeric aggregations.

Region merge is only one of the reasons for exploring a new model. There are a number of applications that deal with multiple thematic maps (Example 2) and on-line analysis of such maps involves frequent map overlay operations. The regions resulting from overlays are treated as measures. Additionally, maps may contain spatial objects of different types (Example 3) and overlay of such objects often has to be calculated too. Note that in all these cases, a user may want to drill-down/roll-up along both spatial and nonspatial dimensions.

We now present the model of a spatial data warehouse. From the above discussion, it is clear that both dimensions and measures may contain spatial components. Furthermore, a spatial data cube can be constructed according to the dimensions and the measures modeled in the spatial data warehouse.

3.1.1 Dimensions in a Spatial Date Warehouse

We recognize three types of *dimensions* in a spatial data warehouse:

1. Nonspatial dimension

   A *nonspatial dimension* is a dimension containing only nonspatial data. For example, two nonspatial dimensions, *temperature* and *precipitation*, can be constructed for the data warehouse in Example 1, each is a dimension containing nonspatial data whose generalizations are nonspatial, such as *hot*, and *wet*.

2. Spatial-to-nonspatial dimension

   A *spatial-to-nonspatial dimension* is a dimension whose primitive level data is spatial but whose generalization, starting at a certain high level, becomes nonspatial. For example, *state* in the U.S. map is spatial data. However, each state can be generalized to a nonspatial value, such as *pacific_northwest*, or *big_state*, and its further generalization is nonspatial, and thus playing a similar role as a nonspatial dimension.

3. Spatial-to-spatial dimension

   A *spatial-to-spatial dimension* is a dimension whose primitive level and all of its high-level generalized data are spatial. For example, *equi-temperature-region* in Example 2 is spatial data, and all of its generalized data, such as regions covering *0-5_degree*, *5-10_degree*, and so on, are also spatial.

Note that the last two dimension types indicate that a spatial attribute, such as *county*, may have more than one way to be generalized to high level concepts, and the generalized concepts can be spatial, such as *map representing larger regions*, or nonspatial, such as *area* or *general description of the region*.

3.1.2 Measures in a Spatial Date Warehouse

We distinguish two types of *measures* in a spatial data warehouse.

1. Numerical measure

   A *numerical measure* is a measure containing only numerical data. For example, one measure in a spatial data warehouse could be *monthly revenue* of a region, and a roll-up may give the total revenue by year, by county, etc.

Numerical measures can be further classified into *distributive*, *algebraic*, and *holistic* [25]. A measure is *distributive* if it can be computed by cube partition and distributed aggregation, such as *count, sum, max*; it is *algebraic* if it can be computed by algebraic manipulation of distributed measures, such as *average, standard deviation*; and it is *holistic* if there is no constant bound on the size of the storage needed to describe a sub-aggregate, such as *median, most_frequent, rank*. The scope of our discussion related to numerical measures is confined to *distributive* and *algebraic* measures.

2. Spatial measure

   A *spatial measure* is a measure which contains a collection of pointers to spatial objects. For example, during the generalization (or roll-up) in a spatial data cube of Example 1, the regions with the same range of *temperature* and *precipitation* are grouped into the same cell, and the measure so formed contains a collection of pointers pointing to those regions.

A computed measure can be used as a dimension in a data warehouse, which we call a *measure-folded dimension*. For example, the measure *monthly average temperature in a region* can be treated as a dimension and can be further generalized to a value range or a descriptive value, such as *cold*. Moreover, a dimension can be specified by experts/users based on the relationships among attributes or among particular data values, or be generated automatically based on spatial data analysis techniques, such as spatial clustering [18, 54], spatial classification [17], or spatial association analysis [49].

We will focus on computation and storage of *spatial measures* in the remainder of this thesis.

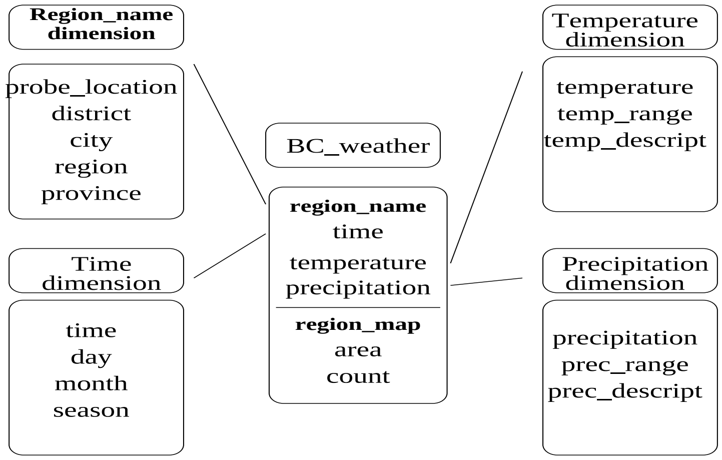

Figure 3.1: Star model of a spatial data warehouse: BC_weather

3.1.3 An Example of a Spatial Data Warehouse

In this subsection we continue examining Example 1 by presenting the star schema model that corresponds to it. The star schema model of Example 1 with its dimensions and measures is illustrated as follows.

Example 4 Spatial data warehouse for BC_weather

A star model can be constructed, shown in Figure 3.1, for the *BC_weather* data warehouse of Example 1, where the B.C. map with regions covered by weather probes is shown in Figure 3.2. Note that neither the map nor the data used in this example is real. Nevertheless, we believe that this fact does not weaken our study.

The spatial data warehouse model consists of four dimensions: *temperature*, *precipitation*, *time*, and *region_name*, and three measures: *region_map*, *area*, and *count*. The concept hierarchy for each dimension, shown in Figure 3.3, can be created by users or experts or generated automatically by data clustering or data analysis. While the first three dimensions are *nonspatial*, the fourth one (*region_name*) is *spatial-to-spatial* dimension. Of the three measures, *region_map* is a *spatial* measure which contains a collection of spatial pointers pointing to the corresponding regions, *area* is a *numerical* measure which represents the sum of the total areas of the corresponding spatial

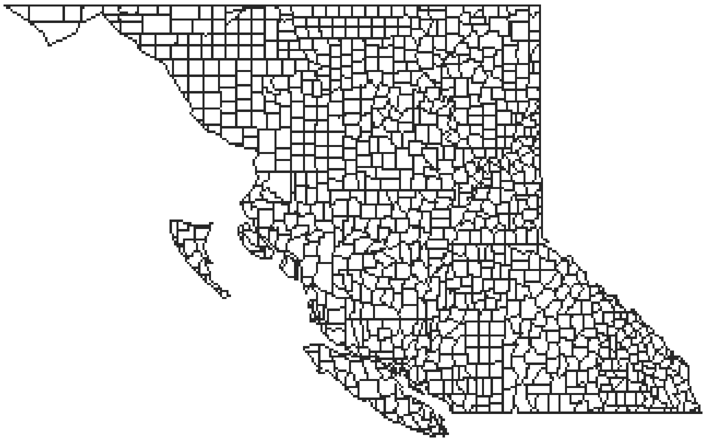

Figure 3.2: Weather probes map

objects, and *count* is a *numerical* measure which represents the total number of base regions (probes) accumulated in the corresponding cell.

Table 3.1 shows a data set that may be collected from a number of weather probes scattered in British Columbia. Notice that *region_name* at the primitive level is a spatial object name representing the corresponding spatial region on the map. For example, region_name *AM08* may represent an area of *Burnaby mountain*, and whose generalization could be *North_Burnaby*, and then *Burnaby, Greater_Vancouver, Lower_Mainland*, and *Province_of_BC*, each corresponding to a region on the B.C. map. □

With these dimensions, OLAP operations can be performed by stepping up and down along any dimension shown in Figure 3.1. Let us now revive popular OLAP operations and analyze how they are performed on a spatial data cube.

1. *Slicing* and *dicing*, each of which selects a portion of the cube based on the constant(s) in one or a few dimensions. For example, one may be interested only in *cold* and *dry* regions located in the *Northern part of British Columbia* This can be realized by transforming the selection criteria into a query against the spatial data warehouse and be processed by query processing methods [6, 25, 30, 64].

Region_name:
probe location $\subset$ district $\subset$ city $\subset$ region $\subset$ province

Time:
day $\subset$ month $\subset$ season

Temperature:
any $\subset$ (cold, mild, hot)
cold $\subset$ (below -20, -20 to -10, -10 to 0)
mild $\subset$ (0 to 10, 10 to 15, 15 to 20)
hot $\subset$ (20 to 25, 25 to 30, 30 to 35, above 35)

Precipitation:
any $\subset$ (dry, fair, wet)
dry $\subset$ (0 to 0.05, 0.05 to 0.2)
fair $\subset$ (0.2 to 0.5, 0.5 to 1.0, 1.0 to 1.5)
wet $\subset$ (1.5 to 2.0, 2.0 to 3.0, 3.0 to 5.0, above 5.0)

Figure 3.3: Concept hierarchies in a spatial data warehouse: BC_weather

Region_name	Time	Temperature	Precipitation
AA00	01/01/97	−4	1.5
AA01	01/01/97	−7	1.0
AA02	01/01/97	−7	2.5
...	...	...	...
AA00	01/02/97	−6	2.5
AA01	01/02/97	−8	1.0
AA02	01/02/97	−9	2.0
...	...	...	...

Table 3.1: Weather probes table

2. *Pivoting*, which presents the measures in different cross-tabular layouts. This can be implemented in a similar way as in nonspatial data cubes. For example, a spreadsheet table containing *temperature* and *precipitation* as row and columns respectively may be presented to a user. The values (e.g., cells) in the table may contain *area* of the corresponding region(s).

3. *Roll-up*, which generalizes one or a few dimensions (including the removal of some dimensions when desired) and performs appropriate aggregations in the corresponding measure(s). For example, one may roll-up on *temperature* dimension to get summarized information. For nonspatial measures, aggregation is implemented in the same way as in nonspatial data cubes [1, 25, 67]. However, for spatial measures, aggregation takes a collection of a spatial pointers in a map or map-overlay and performs certain *spatial aggregation* operation, such as region merge, or map overlay. It is challenging to efficiently implement such operations since it is both time and space consuming to compute spatial merge or overlay and save the merged or overlaid spatial objects. This will be discussed in detail in later sections.

4. *Drill-down*, which specializes one or a few dimensions and presents low-level objects, collections, or aggregations. This can be viewed as a reverse operation of *roll-up* and can often be implemented by saving a low level cuboid and performing appropriate generalization from it when necessary.

From this analysis, one can see that a major performance challenge for the implementation of spatial OLAP is the efficient construction of spatial data cubes and the implementation of roll-up/drill-down operations.

Example 5 Spatial OLAP on BC_weather data warehouse

The roll-up of the data cube of B.C. weather probe of Example 4 can be performed as follows.

The roll-up on the *time* dimension is performed to roll-up values from day to month. Since temperature is a *measure-folded* dimension, the roll-up on the *temperature* dimension is performed by first computing the *average temperature grouped by*

Time	Temperature	Precipitation	Region_map
January	below -20	dry	{AK04,AK07,...,VS67}
January	below -20	fair	{AG10, AG05,...,TP90}
...	...	...	...

Table 3.2: Result of a roll-up operation

month and by spatial region and then generalizing the values to ranges (such as -10 to 0) or to descriptive names (such as "cold"). Notice that one may also obtain *average daily high/low temperature* in a similar manner as long as the way to compute the measure and transform it into dimension hierarchy is specified. Similarly, one may roll-up along the *precipitation* dimension by computing the average precipitation grouped by month and by spatial region. The *region_name* dimension can be dropped if one does not want to generalize data according to specified regions.

Thus, the generalized data cube contains three dimensions, *Time (in month)*, *Temperature (in monthly average)*, and *Precipitation (in monthly average)*, and one spatial measure *Region_map* which is a collection of spatial object ids, as shown in Table 3.2. Moreover, roll-up and drill-down can be performed dynamically which may produce another table as shown in Table 3.3.

Two different roll-ups from the B.C. weather map data (Figure 3.2) produce two different generalized region maps, as shown in Figure 3.4, each being the result of merging a large number of small (probe) regions from the map shown in Figure 3.2. Computing such spatial merges of a large number of regions flexibly and dynamically poses a major challenge to the implementation of spatial OLAP operations. Note that hundreds of small regions may need to be merged together. If such an operation were performed on-the-fly, all these regions would have to be fetched (likely from a disk) and then merged. Thus, only if appropriate precomputation is performed, can the response time be satisfactory to end users.

□

Time	Temperature	Precipitation	Region_map
March	cold	0.1 to 0.3	{AL04,AM03,...,XN87}
March	cold	0.3 to 1.0	{AM10, AN05,...,YP90}
...	...	...	...

Table 3.3: Result of another roll-up operation

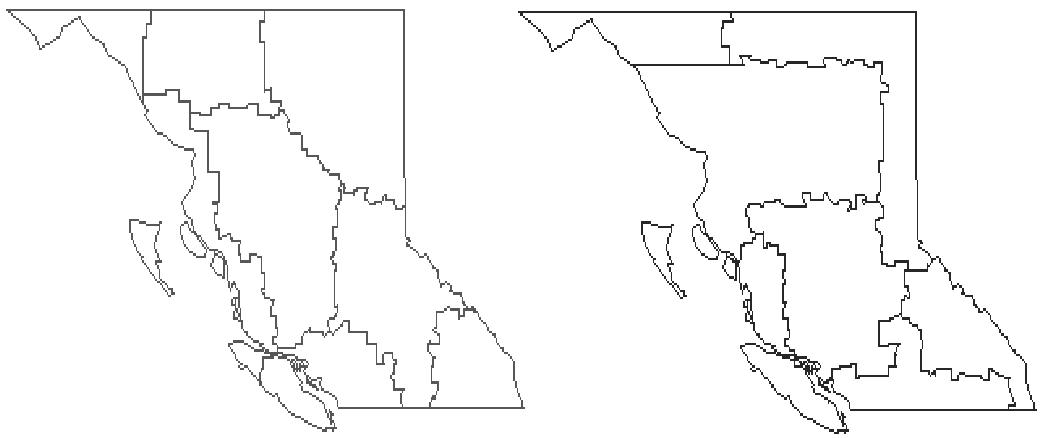

Figure 3.4: Generalized regions after different roll-up operations

3.2 Implementation of a Spatial Data Cube

A nonspatial data cube contains only nonspatial dimensions and numerical measures. If a spatial data cube contained spatial dimensions but not spatial measures, its OLAP operations could be implemented in a way similar to that of nonspatial data cubes. However, the introduction of spatial measures raises challenging issues on efficient implementation, which is the focus of this study. In this section we present the challenges in computation of spatial measures and propose some methods.

3.2.1 Challenges in Implementation of a Spatial Data Cube

Similar to the structure of a nonspatial data cube [8, 67], a spatial data cube consists of a lattice of cuboids, where the lowest cuboid (*base point*) references all the dimensions at the primitive abstraction level (i.e., group-by all the dimensions), and the highest cuboid (*apex point*) summarizes all the dimensions at the top-most abstraction level (i.e., no group-by's in aggregation). Thus, ascending/descending the lattice corresponds to roll-up/drill-down operation. A lattice structure for a cube with three dimensions is shown in Figure 3.5, where A, B, and C represent dimension names, and subscripts annotate concept hierarchy levels (0 for the lowest level). Note that rolling-up to *"any"* (highest level) for a dimension is same as dropping the dimension (i.e., dimension reduction).

Drill-down, roll-up, and dimension reduction in a spatial data cube result in different cuboids in which each cell contains the aggregation of measure values or clustered spatial object pointers. While, the aggregation (such as sum, average, etc.) of numeric values results in a new numeric value, the clustering of spatial object pointers may not lead to a single, new spatial object. Only if all the objects pointed to by the spatial pointers are connected, can they be merged into one large region; otherwise, they will be represented by a set of isolated regions.

A numeric measure usually takes only about two- to eight-byte storage space and requires a relatively small amount of computation time. However, being a spatial object, a spatial measure may take kilo- to mega-bytes in storage space and it is much more costly to compute the merge or overlay of a set of spatial regions than its

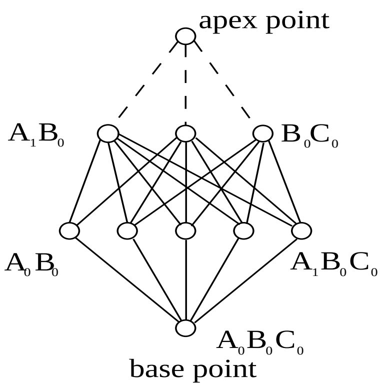

Figure 3.5: A lattice of cuboids

numerical counterpart.

Furthermore, one may expect that OLAP operations, such as drilling or pivoting, be performed flexibly in a spatial data warehouse with fast response time since a user may like to interact with the system to obtain necessary statistics for decision making. Instead of computing such aggregations on-the-fly, it is often necessary to precompute some high-level views (*cuboids*) [1] and save them as *materialized views* (*computed cuboids*) to facilitate fast response to OLAP operations. Can all the possible results of spatial OLAP operations be precomputed and saved for efficient OLAP? Let us perform a simple analysis.

There are different products in (*nonspatial*) data warehouse industry: some materialize every cuboid, some none, but some only part of the cube (i.e., some of the cuboids). There are interesting studies on efficient implementation of data cubes [1, 67].

In the implementation of spatial data cubes, we face a dilemma of balancing the cost of on-line computation and the storage overhead of storing computed spatial measures: the substantial computation cost of on-the-fly computation of spatial aggregations calls for precomputation but substantial overhead for storing aggregated spatial values discourages it. Obviously, we should not materialize every cuboid with limited storage space but we cannot afford to compute all the spatial aggregates on-the-fly. We first clarify terms that will be extensively used in the rest of this thesis.

A *cuboid of a spatial data cube* corresponds to a single table whose columns are dimensions and measures of the spatial data cube. A *cell of a cuboid* corresponds to a tuple in such a table.

To show the importance of selective materialization of cuboids we examine the data cube structure in more detail. Let the data cube consist of m measures, M_1, $\ldots$, M_m and n dimensions, D_1, $\ldots$, D_n, where the i-th dimension D_i has k_i levels of hierarchy, and the top level has only one special node *"any"* which corresponds to the removal of the dimension.

For an n-dimensional data cube, if we allow new cuboids to be generated by climbing up the hierarchies along each dimension (note that the removal of a dimension is equivalent to generalizing to the top level "any"), the total number of cuboids that can be generated is,

$$N = \prod_{i=1}^{n} k_i - 1.$$

This is a big number. For example, if the cube has 10 dimensions and each dimension has 5 levels, the total number of cuboids that can be generated is $5^{10} - 1 \approx 9.8 \times 10^6$. Therefore, it is recommended to materialize only some of all the possible cuboids that can be generated.

Three factors may need to be considered when judging whether a cuboid should be selected for materialization: (1) the potential access frequency of the generated cuboid, (2) the size of the generated cuboid, and (3) how the materialization of one cuboid may benefit the computation of other cuboids in the lattice. A greedy cuboid-selection algorithm HRU has been presented in [38] based on the analysis of the latter two factors. A minor extension to the algorithm may take into consideration the first factor, the potential access frequency of the generated cuboid, where the potential access frequency can be estimated by an expert or a user, or calculated based on the cube access history. The analysis of whether a cuboid should be selected for precomputation in a spatial data cube is similar to a nonspatial one although an additional factor, the cost of on-line computation of a particular spatial cuboid, should be considered in the cost estimation since the spatial computation, such as region merging, map overlaying, spatial join, could be expensive when involving a large number of spatial objects.

The above analysis motivates us to propose interesting techniques for selective materialization of spatial data cubes. In the previous OLAP studies, granularity of data cube materialization has been at the cuboid level, that is, either completely materialize a cuboid or not at all. However, for materialization of the spatial measure, it is often necessary to consider finer granularity and examine individual cells to see whether a group of spatial objects within a cell should be precomputed.

3.2.2 Approaches to Computation of Spatial Measures

In this discussion, we assume that the computation of spatial measures involves *spatial region merge* operation only. The principles discussed here, however, are also applicable to other kinds of spatial operations, such as *spatial map overlay*, *spatial join* [26, 30], and *intersection* between lines and regions [56].

There are at least three possible choices regarding the computation of spatial measures during the spatial data cube construction:

1. Collection of Spatial Pointers

   For some applications, it is not mandatory to merge similar neighboring objects. For example, if one deals with objects that are compact enough (e.g., provinces, states) additional merge would not convey useful information. In this case, pointers to corresponding spatial objects can be collected as shown in Table 3.2 and Table 3.3. This can be implemented easily by storing, in the corresponding cube cell, an indirect pointer to a collection of spatial object pointers. This choice clearly indicates that the (region) merge of a group of spatial objects, when necessary, has to be performed on-the-fly. Nonetheless, this is still a good choice if only a map display is required (i.e., no need for spatial merge), or if there are relatively few regions to be merged in any pointer collection (thus, on-line merge is not so costly an operation). The storage space overhead is relatively small, and similar to that for nonspatial measures. Thus, if the OLAP results are produced only for viewing, display-only mode is useful. However, as we elaborated in previous chapters, OLAP can be integrated with spatial data mining modules, such as spatial association, clustering, classification, etc [36]. In such cases OLAP results are used for further analysis and it is important to merge a number of spatially connected regions (with same nonspatial descriptions).

2. Approximate Computation of Spatial Measures

   We believe that not all portions of a map are equally interesting for the user of a spatial data warehouse. Moreover, different users may have different preferences. It is thus, plausible to precompute rough approximation/estimation of spatial measures and store them in a spatial data cube. If higher precision is needed for specific cells, the system can either fetch precomputed high quality results, if available, or compute them on-the-fly. This choice is good for a rough view or coarse estimation of spatial merge results under the assumption that it takes little storage space to store the coarse estimation results. For example, the minimum bounding rectangle (MBR) of the spatial merge result (representable by two spatial points) can be taken as a rough estimation of the merged region. This estimated measure, Figure 3.6, occupies as little space as a nonspatial measure and can be presented quickly to users. Another possible choice is to use a convex hull instead of an MBR.

3. Selective Materialization of Spatial Measures

   Selective materialization seems to be the best choice but the challenge is how to select a set of spatial measures for precomputation. A previous study [38] shows that materializing every cuboid requires a huge amount of disk space, whereas not materializing any cuboid requires a great deal of on-the-fly, and often redundant, computation. That study promotes partial materialization as a desired solution and algorithms have been proposed to determine what should be precomputed for partial materialization of data cubes.

   The selection can be performed at the cuboid level, i.e., either precompute and store *every* set of mergeable spatial regions for *every* cell of a selected

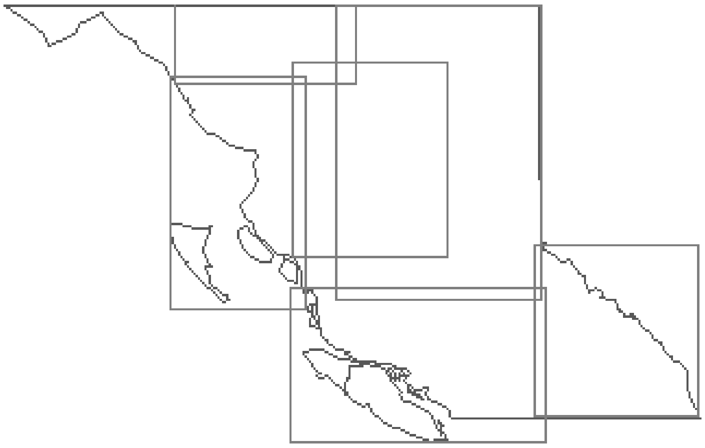

Figure 3.6: Rough approximation of the spatial measures

cuboid, or precompute none if the cuboid is not selected. Since a cuboid usually covers the whole map, it may involve precomputation and storage of a large number of mergeable spatial objects although some of them could be rarely used. Therefore, it is recommended to perform selection at a finer granularity level by examining each group of mergeable spatial objects in a cuboid to determine whether such a merge should be precomputed. Since this approach examines individual cells, we also refer to it as *selective materialization at the cell level.* In addition to selective materialization, a special care should be taken during on-line processing, i.e. using the spatial data warehouse. Best candidates for a target query should be chosen from a set of precomputed merged regions.

Our further discussion is focused on how to select groups of mergeable spatial objects for precomputation from a set of spatial data cuboids chosen by cuboid-selection algorithm HRU [38].

Chapter 4

Materialization of Spatial Measures

In the previous chapter, we underlined the importance of spatial measures and their precomputation and elucidated that, due to high processing requirements, it is not feasible to perform all computations on-the-fly. We enumerated three approaches for collecting information of spatial measures. Since the first one, Collection of spatial pointers, is self-explanatory we concentrate on the remaining two approaches, namely Approximate computation of spatial measures and Selective materialization of spatial measures. This chapter addresses the latter two approaches in more detail and proposes some algorithms.

While for the first of the the three approaches selectivity is expressed at the cuboid level, the remaining two approaches exploit cell-level selectivity. However, the motivations for cell-level selectivity differ for these two approaches. While Approximate computation of spatial measures method materializes only the spatial objects that convey reasonably high quality information, Selective materialization of spatial measures method materializes the objects (aggregations) that are expected to provide significantly shorter response time for spatial OLAP queries. Various heuristics are used and they will be described throughout this chapter.

4.1 Approximate Computation of Spatial Measures

We expressed primary motivations for approximate computation of spatial measures earlier in this thesis. We now explore this approach more specifically and propose a simple, yet effective algorithm.

Each region of the map can be represented by using a *minimum bounding rectangle* (*MBR*). MBRs have been used extensively to approximate spatial objects in spatial analysis because they need only two points for their representation; in particular each object p is represented as an ordered pair (p'_l, p'_u) of points that correspond to the lower left and upper right points of the MBR p' that completely covers p, while having minimal area.

The MBRs (one for each region) can be organized using the R^*-tree structure [3]. Note that in our case this data structure is mainly read-only since map objects seldom (or never) change. Table 3.2 and Table 3.3 show collections of spatial objects that have same nonspatial descriptions after some OLAP operations are performed. In this approximation method, spatial objects whose MBRs are neighbors are considered for merging into a larger MBR. In the rest of this discussion we refer to this class of regions as *merged MBR*, or *MMBR*.

We encounter inaccuracy problem here, i.e., an MMBR that fully contains two or more MBRs may be covered only in a small portion with the associated spatial objects. In that case, the precomputed MMBR provides misleading information (see Figure 4.1). To cope with this problem, we introduce *area_weight* as the percentage of the area covered by the original spatial objects.

Definition 4.1.1 Let $R = \{r_1, r_2, \ldots, r_n\}$ be a set of regions, $M = \{m_1, m_2, \ldots, m_n\}$ be a set of MBRs such that m_i is the MBR for r_i ($i \in (1, n)$), i.e., $m_i = MBR(r_i)$, and A be the MMBR that includes all MBRs in M. Then, $area\_weight(A)$ is defined as following:

$$area\_weight(A) = \frac{\sum_{i=1}^{n} area(r_i)}{area(A)}$$

□

In order to mitigate the inaccuracy problem, we introduce threshold *min_weight*

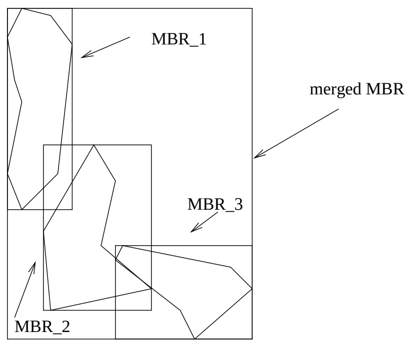

Figure 4.1: A merged MBR with a low area_weight

that filters out MMBRs with low *area_weight*. We believe that only the objects that reveal "good enough" information should be stored. *Rough measures* algorithm is presented as follows.

Algorithm 4.1.1 (Rough Measures Algorithm) An approximation algorithm for precomputation and storage of the merge results during the construction of a spatial data cube.

Input:

- A cube lattice which consists of a set of selected cuboids obtained by running an extended cuboid-selection algorithm similar to HRU [38].
- A group of spatial pointers in each cell of cuboids in the lattice.
- A region map which delineates the neighborhood of the regions.
- An R^*-tree whose leaves are MBRs of regions from the map.
- *min_weight*: A threshold which represents the minimum *area_weight* for an MMBR to be stored in a spatial data cube as a spatial measure.

Output: A spatial data cube selectively populated with spatial measures (MMBRs).

Method:

- The main program is outlined as follows, where *max_cuboid* is the maximum number of cuboids selected for materialization.

(1) FOR $cuboid = 1$ TO max_cuboid DO
(2) FOR EACH $cell$ IN $cuboid$ DO {
(3) $get\_intersecting\_mbrs(cell, candidate\_list)$;
(4) $filter\_area(candidate\_list)$;
(5) $populate\_cube(candidate\_list, cell, cuboid)$;
(6) }

- The procedure *get_intersecting_mbrs(cell, candidate_list)* is outlined as follows. Each *cell* contains a set of pointers to MBRs (one for each map region). This procedure breaks the MBRs into a number of intersecting groups. Each member (MBR) of an intersecting group must intersect with at least one other member from the group. Finally, each intersecting group gets represented with an MMBR that contains all members of the group and all the MMBRs are put into the *candidate_list*.

- The procedure *filter_area(candidate_list)* is outlined as follows.

(1) PROCEDURE $filter\_area(candidate\_list)$ {
(2) FOR EACH $candidate$ IN $candidate\_list$ DO
(3) IF $area\_weight(candidate) < min\_weight$
(4) THEN *remove candidate from candidate_list*;
(5) }

- The procedure *populate_cube(candidate_list, i, j)* is outlined as follows. If *candidate_list* is non-empty, cell i of cuboid j is populated by storing pointers to MMBRs. The MMBRs themselves are stored in the R^*-tree. Note that the position of a cell in a cuboid is determined by values of its dimensions (see Table 3.2 and Table 3.3).

Rationale of the algorithm. *Rough measures* algorithm is to compute and store approximate spatial measures in a spatial data cube. The algorithm is applied to the cuboids selected by the well-known cuboid-selection algorithm HRU [38]. It scans all pointer groups (spatial measures) in selected cuboids, and detects potential merged regions (MMBRs) (Line (3)). Line (4) is to filter out candidates whose *area_weight* is below *min_weight*. This threshold can be chosen by users or experts and/or adjusted dynamically. Candidates that pass the threshold are stored in cells of the spatial data cube (Line (5)). Note that in addition to the spatial measure (e.g., *region_map*), a numerical measure *area* has to be computed for the filtering step. However, its computation does not differ from the computation of measures in a nonspatial data cube. Essentially, the *area* measure is very similar to the *sum* measure (it belongs to distributive numerical measures).

We are well aware of limitations of this algorithm. It provides only rough estimation of spatial measures, and it is often necessary to fetch (or compute) refined measures. Furthermore, since the algorithm works on MBRs, as opposed to map objects, it does not use a precise neighborhood information, i.e., two or more spatial objects may be disjoint, but their MBRs may still intersect.

In spite of the above limitations, we believe that *rough measures* algorithm provides users of a spatial data warehouse with a useful coarse grain information. If more precise information is needed for certain sections of the map, users can either fetch them (if available) or compute on-the-fly. Although MBRs demonstrate some disadvantages when approximating non-convex or diagonal objects, they are still most commonly used approximations in spatial applications. The algorithm is fast and generated spatial measures do not have high storage demands. Finally, since cycles of the loop (Lines (3) to (5)) are mutually independent, the algorithm can be easily parallelized in a multi-threading environment. □

An example of the execution of the algorithm is presented below.

Example 6 Suppose that a set of cuboids has been selected from lattice shown in Figure 4.2 using an extended cuboid-selection algorithm like HRU [38]. Each cuboid contains a set of cells, each containing a group of spatial pointers, representing the spatial measure associated with a particular set of dimension values. Since the focus of our discussion is on the selective materialization of *spatial measures*, only the groups of spatial pointers are shown in Table 4.1, without presenting the corresponding dimension values. Suppose that *min_weight* is 45%. The map is shown in Figure 4.3.

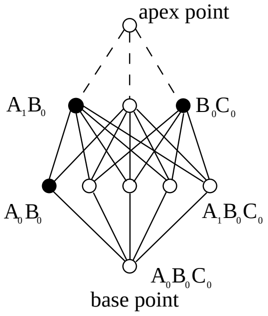

Figure 4.2: A lattice showing selected cuboids

Cuboid_1	Cuboid_2	Cuboid_3
{1, 4, 7, 8, 11, 16, 20}	{1, 4, 7, 13, 17, 20}	{1, 4, 13}
{2, 3, 5, 6, 9, 12, 17}	{2, 3, 6, 9, 16}	{2, 3}
{10, 13, 14, 15, 18, 19}	{5, 11, 18}	{5, 18}
	{8, 12, 15}	{6, 9, 16}
	{10, 14, 19}	{7, 17, 20}
		{8, 12, 15}
		{10, 19}
		{11}
		{14}

Table 4.1: Sets of pointers for selected cuboids

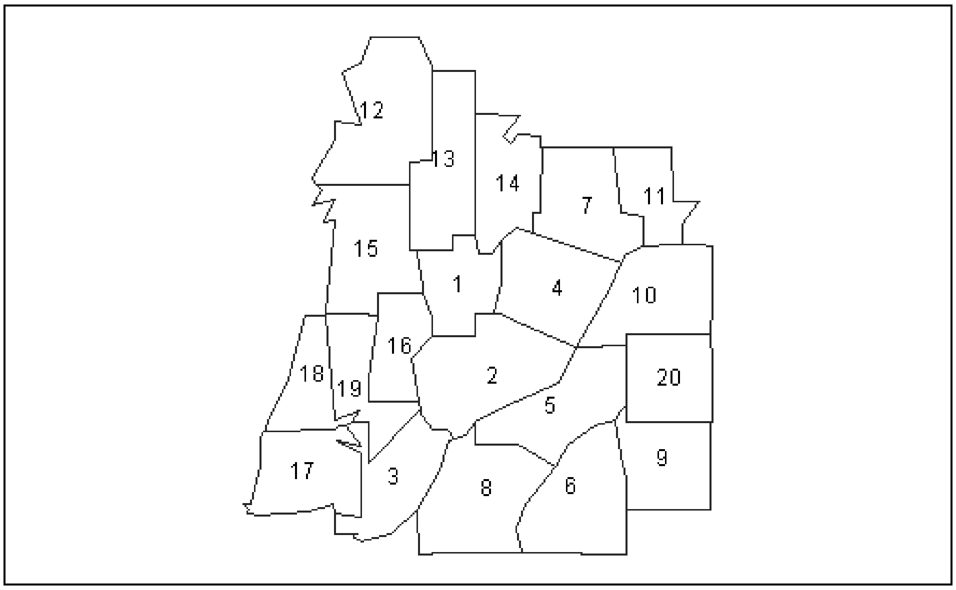

Figure 4.3: An example map 1

The algorithm analyzes one group of spatial measures at a time. In other words, processing of one group is not interleaved with that of any other group. Thus, in order to illustrate the execution of the algorithm, we assume that all iterations are executed in parallel.

After applying steps depicted in Lines (3) and (4) of the algorithm, we get a set of candidates (MMBRs) shown in Table 4.2, where *area_weight* is associated with each candidate. Only the candidates that pass *min_weight* threshold (Table 4.3) are stored in the spatial data cube.

Cuboid_1	Cuboid_2	Cuboid_3
{1, 4, 7, 11, 16}(41%)	{1, 4, 7, 13}(50%)	{1, 4, 13}(40%)
{2, 3, 5, 6, 9, 17}(49%)	{2, 3, 6, 9, 16}(43%)	{2, 3}(35%)
{13, 14, 15, 18, 19}(35%)	{12, 15}(62%)	{6, 9}(61%)
		{12, 15}(62%)

Table 4.2: Candidates for merged MBRs

{12, 15}(62%)
{6, 9}(61%)
{1, 4, 7, 13}(50%)
{2, 3, 5, 6, 9, 17}(49%)

Table 4.3: Merged MBRs that are stored in the spatial data cube

□

4.2 Selective Materialization of Spatial Measures

In this section we examine three algorithms for the selective materialization of spatial measures that are improvement from the *rough measures* algorithm. Here, instead of storing merged MBRs as rough approximations of spatial measures, we strive for accuracy. The process of getting precise spatial measures carries two challenges: increased computation time and substantial storage overhead. While merging two MBRs takes small and constant amount of time (i.e., four comparisons), merging two regions takes $O(n)$ time, where n is the total number of vertices (with the assumption that the vertices were sorted off-line). Similarly, the storage requirements for a merged MBR are negligible comparing to that for merged regions.

Thus, even when we decide to materialize a cuboid, it is still unrealistic to compute and store every spatial measure for every cell because it may consume a substantial amount of computation time and disk space, especially considering that many of them may not be examined in detail, or may only be examined a small number of times. In the following subsections, we introduce *spatial greedy* algorithm, *pointer intersection* algorithm, and *object connection* algorithm, that selectively materialize spatial measures. The algorithms select cells that will be materialized, i.e., merged during spatial data cube construction time. Therefore, in the remainder of this thesis we refer to this process as a *premerge*.

4.2.1 The Problem Statement

The goal of selective materialization of spatial measures is to select and merge groups of connected spatial objects that will, given storage space constraints, provide shortest time to evaluate results of spatial OLAP queries. The groups can be organized in a partial order, i.e., if a group that contains objects {1, 3, 8} is merged it can help in merging the group {1, 3, 6, 8}. We now define the partial order more formally.

Consider two groups that contain connected spatial objects G_i and G_j. We say that $G_i \preceq G_j$ if and only if $G_i \subseteq G_j$. Similarly to [38], the operator $\preceq$ imposes a partial ordering of the groups. We now state input, output, and benefit of selective materialization of spatial measures.

Input:

- N regions on a map, where each of the regions has 0 to $N-1$ neighbors. The ids for map regions form a set M.
- A lattice L where each node of the lattice contains a set S described as follows,

$$S = \{s \mid s \subset M\}$$

$$(\forall s_i, s_j \in S)(i \neq j \Rightarrow s_i \cap s_j = \emptyset)$$

  The lattice L corresponds the lattice of cuboids chosen by HRU algorithm, while a set S within a node (i.e., cuboid) of such lattice corresponds to a set of mergeable (connected) groups for a cuboid.
- weight w for each node in lattice L. In our case, the weight of a node corresponds to the access frequency of a cuboid.
- Allocated storage space for merged objects.

Output: A set T of selected mergeable groups, described as follows.

Groups in T provide minimum on-line computation time to compute average merge of all groups within the lattice.

Analysis: The selection of group G_i provides a benefit for all groups G_j, such that $G_i \preceq G_j$. Suppose that $G_i = \{a_1, \ldots, a_n\}$ and $G_j = \{a_1, \ldots, a_n, a_{n+1}, \ldots, a_{n+m}\}$.

The benefit $B_{i,j}$ of merging group G_i with respect to group G_j is expressed as follows.

$$B_{i,j} = merge\_time(\{a_1, \ldots, a_{n+m})\}) - merge\_time(\{\{a_1, \ldots a_n\}, a_{n+1}, \ldots, a_{n+m}\}$$

Given a lattice L of dimensions, each being associated with a benefit value B and a weight value W, find a fixed number K of dimensions to merge, such that the total computational time using the merged dimensions as a basis in the computation, is minimized.

This problem is intractable because it can be reduced from set-covering problem. Thus, we propose several heuristic strategies.

4.2.2 Spatial Greedy Algorithm

As we explained in Chapter 2, the most widely used algorithm for the selection of cuboids for materialization is the HRU, a greedy algorithm presented in [38]. Although this algorithm performs well in the creation of a conventional (nonspatial) data cube, it cannot be applied for handling spatial measures. Therefore, we propose a new algorithm that materializes only selected cells in the cuboids chosen by the HRU algorithm. Note that each cell contains a group of pointers to spatial objects. We also refer to this as a group of spatial objects. The following heuristics are used for selection of groups of connected regions to be premerged.

- *access frequency*
  Based on the access history or the estimation of the access frequency of a set of nodes, one can calculate the benefit of the merge. If a group of connected regions is more frequently accessed than other groups, it is more beneficial to premerge (and save) this group of connected regions.

- *cardinality of a set of connected regions*
  If a candidate group has more connected regions than other groups, it is more beneficial to select this candidate in the premerge (having fewer disk accesses during on-line processing). Notice that if a merge is performed on a set of

connected regions at a descendant node, the subsequent cost analysis on its ancestor nodes should count the newly merged region as one region only.

- *sharing among the nodes in the cube lattice structure*
  If a candidate is shared among more nodes in the lattice structure, it is more beneficial to select this candidate for premerge. Notice that in this case, the access frequency of a group is the sum of access frequencies of all the nodes in which the group appears.

Based on these heuristics, a benefit formula is worked out to compute the total benefit of merging a group of connected regions. The *total benefit* is the sum of *direct benefit* and *indirect benefit*. The former is the benefit generated by the merged group itself due to the reduction of both the accessing and merging cost (since no merge needs to be computed at the query processing time); whereas the latter is the benefit of the other groups in the ancestors of the node containing the premerged group due to their use of premerged group, which reduces accessing and on-line computation costs. Ascending the lattice of cuboids leads to more general descriptions of data in the database. Subsequently, if some objects have same nonspatial descriptions in one cuboid, they will have same descriptions in all ancestors of that cuboid. We now introduce a term *non-occluded ancestor*.

Definition 4.2.1 Let F and G be groups containing pointers to spatial objects such that $G \subset F$. Then, group F is a *non-occluded ancestor* of G, $G \prec F$, if the following conditions are satisfied:

- group F has not been materialized
- there is no materialized group J such that $G \subset J \subset F$
- there is no materialized group J, $J \subset F$ such that $G \cap J \neq \emptyset$ and *cardinality(J)* $>$ *cardinality(G)* □

The following formulae compute the total benefit of premerging a group G:

$$direct\_benefit(G) \quad = \quad access\_frequency(G) \times cardinality(G) \qquad (2.1)$$

$$indirect\_benefit(G) = \sum_{G \prec A} access\_frequency(A) \times (cardinality(G) - 1) \quad (2.2)$$

$$total\_benefit(G) = direct\_benefit(G) + indirect\_benefit(G) \quad (2.3)$$

The Formula (2.1) indicates that the direct benefit of a group G is the product of its access frequency and its cardinality, i.e., the number of regions to be merged. This is derived based on the following observation. For a group containing k regions to be merged, if it is merged into one region, the cost of each access is to fetch the merged region once. However, if the group were not merged into one region, it would take about k unit accesses to fetch these k regions, perform on-line merge, store the result into a temporary file, and then take one unit access to fetch the merged temporary file. Thus, the merge saves about k unit disk fetches for each access.

While the access frequency and the cardinality of the connected group of spatial objects contribute to the direct benefit, sharing among nodes in the cube lattice structure contributes to the indirect benefit. In order to determine the indirect benefit for a group G, we have to consider all groups that contain group G. By premerging group G, we effectively decrease the cardinality of all its ancestor groups. If a group contains k connected regions, premerging its subgroup that contains n regions ($n < k$), decreases cardinality of the group by $n-1$. Formula 2.2 shows that only non-occluded ancestors of a group G contribute to its indirect benefit. The following example illustrates computation of the indirect benefit.

Example 7 Let $A = \{1, 2, 3, 4, 5\}$, $B = \{1, 2, 3, 5\}$, $C = \{1, 5\}$, $D = \{2, 3, 4\}$, and $E = \{2, 3, 4, 5\}$, and $F = \{4, 5\}$ be six mergeable groups. We explain the following five cases that can occur when calculating the indirect benefit of group D.

1. there are no materialized groups
   Since group D is contained within groups A and E, both these groups contribute to the indirect benefit of D.

2. only group E has already been materialized
   There are no groups that contribute to the indirect benefit of group D. Both A and E are occluded ancestors of D (E is materialized and $D \subset E \subset A$).

3. only group C has already been materialized
   The indirect benefit of D is the same as in case 1, because groups C and D do not intersect.

4. only group F has already been materialized
   The indirect benefit of D is the same as in case 1, because even though groups F and D intersect, the cardinality of D is larger than the cardinality of F.

5. only group B has already been materialized
   Only group E contributes to the indirect benefit of group D. Group A is occluded by group B, since groups B and D intersect and the cardinality of B is larger than the cardinality of D.

□

After the mergeable candidate groups are detected, the greedy algorithm proceeds as follows. In the first round, the algorithm computes the total benefits for all candidate groups, compares their benefits, and selects the one with the highest benefit. In subsequent rounds, the benefit estimation may change for some groups since these groups may contain the subgroups of the merged groups in the previous round(s). The benefit for these groups will be updated and such updates will propagate up along the lattice. The adjusted benefits are compared among the remaining candidates and the one with the highest current benefit is selected for the premerge. This process continues until it completes the maximum number of allowable merges where the maximum number of merges can be determined based on the allocated disk space, or other factors.

Based on the above outline, the algorithm is presented as follows.

Algorithm 4.2.1 (Spatial Greedy Algorithm) A greedy algorithm which selects candidate (connected) region groups for premerging in the construction of a spatial data cube.

Input:
- A cube lattice which consists of a set of selected cuboids (presented as nodes) obtained by running a cuboid-selection algorithm such as HRU [38].

- An access frequency table which shows the access frequency of each node in the lattice.
- A group of spatial pointers in each cell of cuboids in the lattice.
- A region map which delineates the neighborhood of the regions. The information is collected in an *obj_neighbor* table in the format of *(object_pointer, a_list_of_neighbors)*.
- *max_num_group* as the maximum number of groups which are expected to be selected for premerge.

Output: A set of candidate groups, stored in *merged_obj_table*, selected for spatial premerge, and a spatial data cube selectively populated with spatial measures.

Method:

- The main program is outlined as follows.

(1) $find\_connected\_groups(candidate\_table)$;
(2) $merged\_obj\_table = \emptyset$;
(3) $remaining\_set = candidate\_table$;
(4) REPEAT
(5) $\quad select\_candidate(candidate\_table, merged\_obj\_table)$;
(6) UNTIL $cardinality(merged\_obj\_table) \geq max\_num\_group$
(7) $populate\_cube(merged\_obj\_table)$;

- The procedure $find\_connected\_groups(candidate\_table)$ is outlined as follows. For each cuboid in the cube lattice, determine mergeable groups(s) in its spatial pointer groups, by using *obj_neighbor* table. Then, calculate the access frequency of every detected group by summing up frequencies of the cuboids in which the group appears.

- The procedure *select_candidate(candidate_table, merged_obj_table)* is outlined as follows.

```
(1)  PROCEDURE select_candidate(candidate_table, merged_obj_table) {
(2)      max_benefit = 0;
(3)      FOR EACH group IN remaining_set DO {
(4)          total_benefit = direct_benefit(group) + indirect_benefit(group);
(5)          IF (max_benefit < total_benefit)
(6)          THEN {
(7)              max_benefit = total_benefit;
(8)              best_group = group;
(9)          }
(10)     }
(11)     merged_obj_table += best_group;
(12)     remaining_set -= best_group;
(13) }
```

  The functions *direct_benefit(group)* and *indirect_benefit(group)* calculate direct and indirect benefit for the *group* according to Formula 2.1 and Formula 2.2. Notice that if more than one group have the same largest total benefit, the group containing fewer vertices is chosen in order to generate smaller merged region and save the total space.

- The procedure *populate_cube(merged_obj_table)* is outlined as follows. Objects stored in the *merged_obj_table* are linked with the cells of the cuboids. This can be implemented by storing pointers to the premerged objects. Note that more than one cell can point to a single spatial object, i.e., a spatial object may be the measure in more than one cuboid. □

Rationale of the algorithm. The *spatial greedy* algorithm can be reasoned as follows. The algorithm works on the cuboids selected by the cuboid-selection algorithm HRU [38]. Line (1) finds all mergeable groups within selected cuboids. Lines (2) and

(3) initialize *merged_obj_table* to an empty set, and *remaining_set* to all connected groups (*candidate_table*). Line (5) presents one iteration of the greedy algorithm. At each iteration, the algorithm selects the best candidate based on the benefit calculation. The algorithm is a greedy one because it commits to a local maximum benefit at each iteration, however, not every locally maximum choice can guarantee the global maximality. As shown in the analysis of the HRU algorithm in [38], the global optimality is an NP-hard problem. Therefore, based on the similar reasoning to that in [38], the algorithm derives a suboptimal solution for the selection of candidate groups. Note that instead of the actual number of groups (*max_num_group*), the percentage of groups to be selected for premerge can be specified.

Both our algorithm and the HRU algorithm are greedy algorithms for selective materialization in the construction of data cubes. Beside the difference in application domains: ours is on spatial data cube construction, whereas theirs is on nonspatial ones, there are several other major differences. First, the HRU algorithm is to selectively materialize *cuboids* (*views*), whereas ours is to selectively materialize particular *cells* of the chosen cuboids. This additional level of examination is essential since the nonspatial aggregation results in simple measures, whereas the spatial one needs both nontrivial spatial computation and substantial storage space. Thus, the two algorithms are dealing with the problems at different levels. Second, the HRU algorithm does not take node access frequency into consideration, whereas ours considers it seriously. We believe that access frequency is an important measure since it may not be beneficial to precompute and store the rarely accessed spatial elements.

In comparison with the HRU algorithm, the selection handled in our algorithm is at a deeper level and examines every precomputed element in a node to be materialized. This refined computation is more costly than examining only at a (lattice) node level. However, the complexity introduced here involves mainly simple benefit formula computation which costs usually less than a spatial operation. Moreover, this computation is done at the cube construction time but it will speed on-line spatial computation or save substantial storage space, and is thus worth doing. □

An example of the execution of our algorithm is presented below.

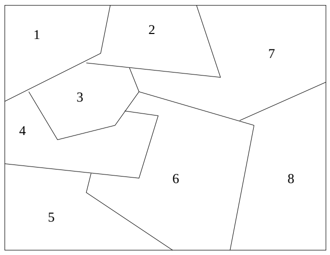

Figure 4.4: An example map 2

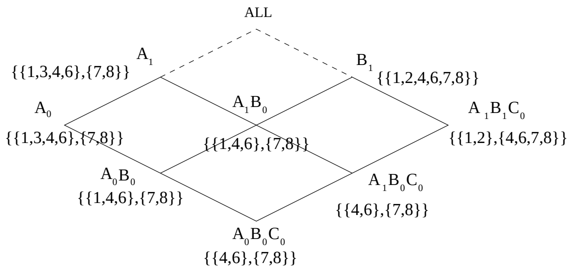

Figure 4.5: A lattice for the selected cuboids

Example 8 A map with its spatial region partition is given in Figure 4.4 and a (cube) lattice which represents the relationships among a set of selected nodes after execution of the HRU algorithm is presented in Figure 4.5. Suppose that the mergeable groups for each cuboid were extracted. We show them too in Figure 4.5. Some groups that belong to a single cuboids may are connected with each other, but they still exist as the separate groups (e.g., {4, 6} and {7, 8} in cuboid $A_0B_0C_0$). This is explained as follows. Only groups that belong to same tuples may be merged, because they describe the object with same values for nonspatial dimensions. Suppose the access frequency of each cuboid is shown in Table 4.4.

The access frequency of every candidate group (i.e., a set of connected regions) is equal to the access frequency of the corresponding node where it resides. If it resides in more than one node, its access frequency is the sum of the access frequency of all the nodes where it resides. For example, group {4, 6} appears in two nodes, $A_0B_0C_0$

Node (cuboid)	Access frequency
$A_0B_0C_0$	100
A_0B_0	350
$A_1B_0C_0$	80
A_1B_0	220
$A_1B_1C_0$	70
B_1	60
A_0	70
A_1	50

Table 4.4: Access frequency of the cuboids

Group	Freq.	dir	indir	total	dir	indir	total	dir	indir	total
"1,2"	70	140	60	200	140	0	140	140	0	140
"1,2,4,6,7,8"	60	360	0	360	240	0	240	180	0	180
"1,3,4,6"	120	480	0	480	240	0	240	240	0	240
"1,4,6"	570	1710	360	**2070**						
"4,6"	180	360	820	1180	360	70	430	360	70	**430**
"4,6,7,8"	70	280	180	460	280	180	460	210	120	330
"7,8"	870	1740	130	1870	1740	130	**1870**			

Table 4.5: First three iterations of spatial greedy algorithm

and $A_1B_0C_0$. Thus, the accumulated access frequency of the group {4, 6} is 100 + 80 = 180.

Using the benefit calculation formulae, Table 4.5 is generated which describes the process of benefit computation and premerged region selection, as shown below. At the first iteration, starting with group $G_{12} = \{1, 2\}$, we get $direct\_benefit(G_{12}) = frequency(\{1, 2\}) \times cardinality(\{1, 2\}) = 70 \times 2 = 140$, and $indirect\_benefit(G_{12}) = frequency(\{1, 2, 4, 6, 7, 8\}) \times (cardinality(\{1, 2\}) - 1) = 60 \times 1 = 60$. Thus the total benefit = direct benefit + indirect benefit = 200. Attention should be paid to the calculation of the indirect benefit of some groups. Let us examine another group, $G_{146} = \{1, 4, 6\}$. Its indirect benefit, $indirect\_benefit(G_{146}) = (frequency(\{1, 2, 4, 6, 7, 8\}) + frequency(\{1, 3, 4, 6\})) \times (cardinality(\{1, 4, 6\}) - 1) = (60 + 120) \times 2 = 360$. The largest total benefit (with a value of 2070) is for the group $\{1, 4, 6\}$, and that group is the first selected premerging group.

In the second iteration, for some groups, such as {4, 6, 7, 8}, the total benefit will not be changed. However, for some groups direct benefit, indirect benefit, or both may change. Take group $\{1, 2, 4, 6, 7, 8\}$ as an example. With the merge of $\{1, 4, 6\}$ in the first round, the cardinality of $\{1, 2, 4, 6, 7, 8\}$ reduces from 6 to 4. Therefore, its direct benefit = $frequency(\{1, 2, 4, 6, 7, 8\}) \times cardinality(\{\{1, 4, 6\}, 2, 7, 8\}) = 60 \times 4 = 240$. The indirect benefit is 0 since there is no any other larger group that can benefit from merging this one.

The computation of the indirect benefit becomes more complex like in the case of $\{4, 6\}$. In the first iteration, $\{1, 4, 6\}$, $\{1, 3, 4, 6\}$, $\{4, 6, 7, 8\}$, and $\{1, 2, 4, 6, 7, 8\}$ would benefit from merging $\{4, 6\}$. However, after group $\{1, 4, 6\}$ has been merged, it occludes groups $\{1, 3, 4, 6\}$ and $\{1, 2, 4, 6, 7, 8\}$. In other words, it is more beneficial to use $\{1, 4, 6\}$ than $\{4, 6\}$ to merge these two groups. Being the only non-occluded ancestor of $\{4, 6\}$, group $\{4, 6, 7, 8\}$ solely contributes to the indirect benefit of $\{4, 6\}$. Consequently, the indirect benefit drops from 820 to 70 whereas the direct benefit is not changed.

After execution of the third round, the three selected groups are {{1, 4, 6}, {7, 8}, {4, 6}}. The process repeats until the number of groups selected reaches a specified maximum number. □

4.2.3 Pointer Intersection Algorithm

The *spatial greedy* algorithm proposed in the previous subsection, although based on heuristics, selects good candidates for premerge. Unfortunately, the algorithm may not scale well enough in a presence of many map objects, and there are two major reasons for that:

- All groups of spatial pointers have to be divided into disjoint groups i.e., only regions that are connected may be merged.
- All disjoint groups are analyzed. The presence of a large number of mergeable groups is likely to result in very expensive computation of the indirect benefit i.e., some groups may have a number of non-occluded ancestors.

Thus, we propose *pointer intersection* algorithm that favors spatial pointer groups that appear in many cuboids and with high access frequency. In our subsequent analysis, we assume that a set of cuboids have been selected for materialization using an extended cuboid-selection algorithm similar to the HRU algorithm. We now examine heuristics to determine which sets of mergeable spatial objects should be precomputed. The general idea of the algorithm is as follows. Given a set of selected cuboids each associated with an (estimated) access frequency, and *min_freq* (the minimum access frequency threshold), a set of mergeable objects should be precomputed if and only if its access frequency is no smaller than *min_freq*. Notice that a merged object also counts as to be accessed if it is used to construct a larger object. Only after the intersections among the sets of object pointers are computed and those with low access frequency filtered out, does the algorithm examine their corresponding spatial object connections (neighborhood information).

The *pointer intersection* algorithm is outlined as follows.

Algorithm 4.2.2 (Pointer Intersection Algorithm) A pointer intersection method for the selection of a group of candidate connected regions for precomputation and storage of the merge results during the construction of a spatial data cube.

Input:

- A cube lattice which consists of a set of selected cuboids obtained by running an extended cuboid-selection algorithm similar to the HRU algorithm [38]. The selected cuboids are mapped to a sequence of numbers from the top level down.
- An *access frequency table* which registers the access frequency of each cuboid in the lattice.
- A group of spatial pointers (sorted in increasing order) in each cell of cuboids in the lattice.
- A region map which delineates the neighborhood of the regions. The information is collected in an *obj_neighbor* table in the format of *(object_pointer, a_list_of_neighbors)*.
- *min_freq*: A threshold which represents the minimal access frequency of a group of connected regions to be premerged.

Output: A set of candidate groups, stored in *merged_obj_table*, selected for spatial premerge, and a spatial data cube selectively populated with spatial measures.

Method:

- The main program is outlined as follows, where *max_cuboid* is the maximum number of cuboids selected for materialization.

(1) FOR $cuboid\_i = 1$ TO max_cuboid DO
(2) FOR $cuboid\_j = cuboid\_i$ TO max_cuboid DO
(3) FOR EACH $cell\_i$ IN $cuboid\_i$ DO
(4) $get\_max\_intersection(cell\_i, cuboid\_j, candidate\_table)$;
(5) $frequency\_computing\_\&\_filtering(candidate\_table)$;
(6) $spatial\_connectivity\_testing(candidate\_table, connected\_obj\_table)$;
(7) $shared\_spatial\_merging(connected\_obj\_table, merged\_obj\_table)$;
(8) $populate\_cube(merged\_obj\_table)$;

- The procedure *get_max_intersection (cell_i, cuboid_j, candidate_table)* finds the maximal intersections between the cell *cell_i* and all cells within the cuboid *cuboid_j*. It is outlined as follows.

```
PROCEDURE get_max_intersection(cell_i, cuboid_j, candidate_table) {
        cell_j = get_first_cell(cuboid_j);
        remaining_cell_i = cell_i;
        WHILE (|remaining_cell_i| > 1 AND cell_j ≠ ∅) DO {
                intersected_portion = get_max_intersect
                        (remaining_cell_i, cell_j);
                IF |intersected_portion| > 1
                THEN insert_candidate(intersected_portion, candidate_table);
                remaining_cell_i -= intersected_portion;
                cell_j = get_next_cell(cuboid_j);
        }
}
```

The function *get_max_intersect(cell_i, cell_j)* can be implemented as follows. Set two cursors, p_i and p_j, pointing to the starting positions in *cell_i* and *cell_j* respectively (where object pointers are sorted in increasing order). If both cursors point to the identical object, output the pointer to the resulting buffer and forward both cursors. Otherwise, forward the cursor which points to the smaller pointer. This process repeats until one of the cursors reaches the end of the set. The output is the resulting buffer.

Note that *insert_candidate (intersected_portion, candidate_table)* inserts the *intersected_portion* into the *candidate_table* if such a portion is nonexistent in the table together with cuboid ids. If the intersected portion is already in the candidate table, only the new cuboid id(s) is inserted.

- The procedure *frequency_computing_&_filtering (candidate_table)* is outlined as follows.

```
(1)  PROCEDURE frequency_computing_&_filtering(candidate_table) {
(2)      FOR EACH entry p IN candidate_table DO
(3)          IF max_frequency(p) < min_freq
(4)          THEN remove p from candidate_table;
(5)  }
```

The function *max_frequency(p)* returns the sum of the frequencies of the cuboids in which p or its supersets appear. This can be implemented by registering the cuboid ids with each candidate in the candidate table and adding the frequencies of those cuboids which contain p or contain a candidate q being a superset of p.

- The procedure *spatial_connectivity_testing (candidate_table, connected_obj_table)* is outlined as follows. For each entry e in the *candidate_table*, using the *obj_neigh bor* table, repeatedly find the set of connected objects g with maximal cardinality. Then, put g into *connected_obj_table*, and remove it from e until no more grouped objects can be found. Finally, sort the grouped objects in the *connected_obj_table* in increasing order according to its cardinality (i.e., the number of objects in the group).

- The procedure *shared_spatial_merging (connected_obj_table, merged_obj_table)* is outlined as follows. For each group in the *candidate_obj_table*, check if any of its subsets has already been premerged (*candidate_obj_table* is sorted). If premerged subsets of the groups are found, use them to minimize the number of merged regions; otherwise merge the original map regions.

- The procedure *populate_cube(merged_obj_table)* does not differ from the *populate_cube(merged_obj_table)* procedure in *spatial greedy* algorithm. □

Rationale of the algorithm. The algorithm is to find those spatial object groups in the data cube which are frequently accessed and mergeable, and to perform spatial merge for them off-line (during the spatial data cube creation time). The algorithm works on the cuboids that are selected based on an extension to HRU, a well-known cuboid-selection algorithm [38]. Lines (1) to (4) ensure that every pair of cuboids

Cuboid_1	Cuboid_2	Cuboid_3
30	15	20

Table 4.6: Access frequencies of selected cuboids

is examined for each candidate cube cell which derives the maximal intersections of spatial object pointers and stores them into candidate table. Note that a self-intersection for cuboids is performed as well. Line (5) removes from the candidate table those candidates whose total access frequency is smaller than *min_freq*. Line (6) finds the spatially connected subsets within each candidate (a group of pointers to spatial objects) and line (7) materializes them and puts into the *merged_obj_table*. Optimization has been explored in each procedure/function. For example, procedure *get_max_intersection*, checks one cell *cell_i* against *cuboid_j*, by first finding the first candidate cell in *cuboid_j* and then extracting the intersected portion. Afterwards the intersected portion is removed from *cell_i* since there is no more such portion in the remaining cells of the cuboid.
Note. We now clarify the reason for applying the self-intersection in this algorithm (Lines (1) and (2)), since it might not be obvious to a reader. There can be a number of groups that appear in a single, yet frequent cuboid and it is important that such groups be identified. Were a self-intersection not applied, these groups would be skipped. The performance analysis, conducted in Chapter 5, will show that the self-intersection has a significant positive impact on the effectiveness of the algorithm (see Figure 5.6). □

Example 9 In order to illustrate the execution of this algorithm, we revive the example used for the *rough measures* algorithm. The map (Figure 4.3), the lattice showing selected cuboids (Figure 4.2), and the sets of pointers for selected cuboids (Table 4.1) are identical like in Example 6. Let the the access frequency of each selected cuboid be as shown in Table 4.6, and let *min_freq* threshold be 40.

After applying steps depicted on Lines (1) to (4) of the algorithm, we get a set of candidates in Table 4.7. Raw access frequency of a candidate is a sum of the

Intersected_portion	Raw access frequency	Accumulated access frequency	Total access frequency
{1, 4, 7, 20}	45	0	45 ←
{2, 3, 6, 9}	45	0	45 ←
{10, 14, 19 }	45	0	45 ←
{10, 19}	65	0	65 ←
{1, 4}	50	15	65 ←
{7, 20}	50	15	65 ←
{2, 3}	65	0	65 ←
{6, 9}	50	15	65 ←
{1, 4, 13}	35	0	35
{7, 17, 20}	35	0	35
{6, 9, 16}	35	0	35
{5, 18}	35	0	35
{8, 12, 15}	35	0	35

Table 4.7: Candidate_table for selected cuboids

frequencies of all cuboids in which the candidate is found as a maximal intersection. For example, candidate {1, 4} is detected as a maximal intersection between cuboids 1 and 3, so that its raw frequency is 50. Note that {1, 4} appears in cuboid 2 as well, but not as a maximal intersection with any other cuboid ({1, 4, 7, 20} is a maximal intersection between cuboids 1 and 2, while {1, 4, 13} is a maximal intersection between cuboids 2 and 3). On the other hand, the candidates that fully contain a certain group contribute to the accumulated access frequency of that group. The accumulated access frequency of {1, 4} is 15 (access frequency of cuboid 2). Notice that a single cuboid can be counted only once in the calculation of the frequencies. Finally, total access frequency is the sum of raw and accumulated access frequencies. The candidates whose total access frequency is not below the threshold are shown with an arrow (←). To make Table 4.7 smaller, we do not show self-intersections in this example. Since *min_freq* threshold is larger than threshold for any of the cuboids, none of the candidates resulted only from a self-intersection can be merged.

After applying *spatial_connectivity_testing* procedure, we detect that the following regions: {1, 4, 7}, {1, 4}, {2, 3}, and {6, 9} (Table 4.8) should be merged. At the first

{1, 4, 7}
{1, 4}
{2, 3}
{6, 9}

Table 4.8: Regions to be premerged

glance, it seems that it is wasteful to store both {1, 4} and {1, 4, 7} since the former is a subset of the latter. However, if both have high access frequency, it is beneficial to store both for the fast response (avoiding on-the-fly spatial merge). Moreover, other on-the-fly spatial merges may also benefit from storing both groups. For example, {1, 2, 4} may use the precomputed {1, 4}, whereas {1, 2, 4, 7} may use {1, 4, 7}. □

4.2.4 Object Connection Algorithm

In this subsection we present *object connection* algorithm, that is only slightly different from *pointer intersection* algorithm. While the *pointer intersection* algorithm first computes the intersections among the sets of object pointers, and then performs threshold filtering and examines their spatial object connections, the *object connection* algorithm examines the corresponding spatial object connections before threshold filtering.

The *object connection* algorithm is examined in the following.

Algorithm 4.2.3 (Object Connection Algorithm) The object connection method for the selection of a group of candidate connected regions for precomputation and storage of the merge results during the construction of a spatial data cube.

Input: The same as Algorithm 4.2.2.

Output: The same as Algorithm 4.2.2.

Method:

- The main program is different from that of Algorithm 4.2.2 at Line (4) where connection is checked immediately, before proceeding further. Thus, the old Line (6) is removed since it has been done in Line (4).

```
(1)  FOR cuboid_i = 1 TO max_cuboid DO
(2)      FOR cuboid_j = cuboid_i TO max_cuboid DO
(3)          FOR EACH cell_i IN cuboid_i DO
(4)              get_max_connected_intersection
                     (cell_i, cuboid_j, candidate_connected_obj_table);
(5)  frequency_computing_&_filtering(candidate_connected_obj_table);
(6)  shared_spatial_merging(candidate_connected_obj_table,
         merged_obj_table);
(7)  populate_cube(merged_obj_table);
```

- Since only the procedure *get_max_connected_intersection (cell_i, cuboid_j, candidate_connected_obj_table)* is different from the procedure *get_max_intersection* of Algorithm 4.2.2, it is outlined as below. Other procedures are essentially the same and thus are not presented here.

```
(1)  PROCEDURE get_max_connected_intersection
         (cell_i, cuboid_j, candidate_connected_obj_table) {
(2)      cell_j = get_first_cell(cuboid_j);
(3)      remaining_cell_i = cell_i;
(4)      WHILE (|remaining_cell_i > 1| AND cell_j ≠ ∅) DO {
(5)          intersected_portion = get_max_intersect
                 (remaining_cell_i, cell_j);
(6)          IF |intersected_portion| > 1
(7)          THEN insert_connected_candidate
                 (intersected_portion, candidate_table);
(8)          remaining_cell_i -= intersected_portion;
(9)          cell_j = get_next_cell(cuboid_j);
```

(10) }

(11) }

Note that the only difference from Algorithm 4.2.2 is at Line (7) which calls *insert_connected_candidate* rather than *insert_candidate*. The procedure *insert_connected_candidate (intersected_portion, candidate_connected_obj_table)* breaks the *intersected_portion* into a set of connected portions, by checking the *obj_neighbor* table. Each connected portion with the length greater than 1 is inserted into the *candidate_connected_obj_table* if such a portion is nonexistent in the table. In this case, its combined access frequency should be the sum of *cell_i*'s and *cell_j*'s access frequencies. If the connected portion is already in the table, the access frequency should be accumulated. □

Rationale of the algorithm. The major difference of this algorithm from the former one, Algorithm 4.2.2, is at handling of connected objects. The former delays the connectivity checking after *min_freq* threshold filtering, whereas this algorithm does it at the insertion into the candidate table. By looking at the algorithms, one may think that they produce identical results in terms of selected regions for premerge. In the subsequent discussion we will show that it may not always be the case.

Suppose that A and B are two groups detected by *get_max_intersect* procedure. Moreover, let us assume that both groups contain a common connected subgroup C. In the case of *pointer intersection* algorithm these two groups will be checked for spatial connectivity only if their access frequencies are no smaller than *min_freq* threshold. On the other hand, if we apply *object connection* algorithm, groups A and B will be divided into mergeable (connected) subgroups. Thus, group C will be inserted into *candidate_connected_obj_table*. Its total access frequency can be higher than that of groups A and B, since it can appear in more cuboids (union of cuboids for A and B). Thus, it may occur that neither A nor B pass frequency threshold (in *pointer intersection* algorithm) and that C does pass frequency threshold (in *object connection* algorithm). Note that the group C does not get detected in *pointer intersection* algorithm, unless A or B passes the frequency threshold. Therefore, *pointer* *intersection* algorithm generates a subset of groups generated by *object connection* algorithm. We now provide a more formal explanation.

Theorem 4.2.1 Let PIG be a set of groups selected by the pointer intersection algorithm, and OCG a set of groups selected by the object connection algorithm. We claim that $PIG \subseteq OCG$.

Proof. We first provide a proof that each group in PIG must be also in OCG.

According to the algorithm 4.2.2 each group S, $S \in PIG$, must consist of connected regions and its access frequency must pass *min_freq* threshold. Group S will be also detected by *insert_connected_candidate* procedure of algorithm 4.2.3 and later due to its sufficient access frequency will be selected for the premerge. Thus, PIG is a subset of OCG.

Now, we prove that relation $PIG = OCG$ does not always hold.

Let $G = G_1, G_2, \ldots G_n$ be groups of spatial pointers found in more than one cuboid (i.e., intersections) with the following properties:

- For all groups in G, there is a common subgroup S that consists of connected regions.
- There is no group outside G that has S as a subgroup.
- The total access frequency of every group in G is below *min_freq* threshold.

Assume that group S does not exist among intersected groups i.e., it is not detected in procedure *get_max_intersection* of *pointer intersection* algorithm.

Due to insufficient access frequency, none of the groups in G will be put in PIG group. In addition, group S will not be detected by algorithm 4.2.2. On contrary, group S will be detected by *insert_connected_candidate* procedure of algorithm 4.2.3. The set of cuboids *cuboids(S)* in which it resides is

$$cuboids(S) = \bigcup_{i=1}^{n} cuboids(G_i)$$

Since the access frequency of S is higher than that of any group in G, group S may pass the frequency threshold and become a member of OCG.

Connected intersections	Raw access frequency	Accumulated access frequency	Total access frequency
{1, 4, 7}	45	0	45←
{2, 3}	65	0	65←
{6, 9}	65	0	65←
{1, 4}	50	15	65←
{1, 4, 13}	35	0	35
{12,15}	35	0	35

Table 4.9: Candidate_connected_obj table for selected cuboids

Therefore, there may be some groups in OCG that do not appear in PIG. This concludes the proof. □.

In addition to its effectiveness, *object connection* algorithm has different efficiency than *pointer intersection* algorithm. This will be studied in more detail in the following chapter. □

For the same example as for *pointer intersection* algorithm, the execution of Algorithm 4.2.3 is presented as follows.

Example 10 The execution of Lines (1) to (4) of Algorithm 4.2.3 will lead to a set of candidate connected object pointer groups as shown in Table 4.9. Note that the algorithm finds same maximal intersection groups like Algorithm 4.2.2 (see column "Intersected_portion" in Table 4.7). After *insert_connected_candidate* procedure is executed for each of the intersections, we get candidates shown in "Connected intersections" column of Table 4.9. Similarly to Example 9, the access frequencies are computed and frequency filtering applied. Candidates that pass the frequency threshold are marked with an arrow (←). The marked groups are the regions that are selected for premerge.

Since Examples 9 and 10 contain only a small number of objects, the two algorithms produce identical results. Our performance analysis will show that the case described in Theorem 4.2.1 occurs very seldom, and that it has a small impact on the benefits of selective materialization.

□

4.3 Utilization of Spatial Measures in On-Line Processing

Clearly, there are three main tasks in the life cycle of a spatial data warehouse: creation, usage, and maintenance. The focus of this study is the process of creation of a spatial data warehouse. However, we devote this section to the usage of a spatial data warehouse, or more precisely, the usage of materialized spatial measures. Note that the issue of maintenance is not studied in this thesis.

Earlier in this thesis, we elaborated three methods for collecting spatial measures, namely *collection of spatial pointers*, *approximate computation of spatial measures*, and *selective materialization of spatial measures*. While, the measures collected by the first method can be utilized similarly to nonspatial (numerical) measures, the latter two methods need more discussion.

Despite having different accuracy and thus different objectives, both *approximate computation of spatial measures*, and *selective materialization of spatial measures*, populate only portions of the cuboids in the spatial data cube. While the first method eliminates merged MBRs with low *area_weight*, the second one filters out merged regions whose materialization is considered to provide little benefit (while wasting storage space).

4.3.1 Utilization of Estimated Spatial Measures

Estimated, or roughly calculated spatial measures provide a user of a spatial data warehouse with an insight into nonspatial and spatial properties of objects on the map. Although they can seldom be used for final decision making, they can help the user focus on the particular segments of the map. To make a decision support process more fruitful, we suggest the implementation that allows users to dynamically change the *area_weight* threshold. Moreover, we believe that approximate computation of spatial measures should serve only as a coarse grain tool.

4.3.2 Utilization of Precomputed Spatial Measures

With a large number of frequently used spatial objects precomputed and stored in the *merged_obj_table*, it is interesting to see how to find the best candidate set stored in the table for efficient on-line processing. For example, suppose the *merged_obj_table* stores the following set of merged regions: {1, 4}, {1, 4, 7}, {2, 3}, and {3, 8, 9}. A measure cell derived by an OLAP operation may have the following set of spatial pointers: {1, 2, 3, 4, 7, 8, 9}. A smart search algorithm may return two precomputed (merged) regions: {1, 4, 7}, and {3, 8, 9}, with one additional spatial merge, {2} + {3, 8, 9}, done on-the-fly. However, an unintelligent algorithm may return two precomputed regions: {1, 4}, and {2, 3}, with three additional spatial merges, {7} + {1, 4}, {8} + {2, 3} + {9}, done on-the-fly.

At the first glance, one may suggest to first match the precomputed regions containing the largest number of merged objects, then proceed to those with smaller number of merged objects. However, this may not always work well. For example, suppose the *merged_obj_table* contains {1, ..., 10}, {1, ..., 11}, {11, ..., 20}, and {12, ..., 14}, and the targeted measure cell is {1, ..., 20}. If we first select the region containing the most elements, the result will be {1, ..., 11}, and {12, ..., 14}. However, the selection of {1, ..., 10} and {11, ..., 20} is a better choice.

We have the following technique for the selection of precomputed spatial objects for a target T, which consists of a set of spatial object pointers, representing a spatial measure cell resulted from a spatial OLAP operation. Let M be the *merged_obj_table*.

1. Create table S such that, $S = \{s \mid s \in M \wedge s \subseteq T\}$. Notice that an element of *merged_obj_table* M is also the element of table S, if and only if, it is fully contained within the target T. If T is a large set, an index can be constructed on M and/or S for efficient search.

2. Search S to find all the maximal groups G such that

$$G = \{g \mid g \in S \wedge g \subseteq T\}$$

$$(\forall g_1, g_2 \in G)(g_1 \neq g_2 \Rightarrow g_1 \cap g_2 = \emptyset)$$

No	Set of premerged objects	Coverage	Num of premerged objects	Num of on-line merges
1	{1, 2, 3}, {4, 7, 8}, {5, 6}, {9, 10}	10	4	4
2	{1, 2, 3}, {5, 6}, {7, 8, 9, 10}	9	3	4
3	{1, 2, 3, 4}, {5, 6}, {7, 8, 9, 10}	10	3	3
4	{1, 2, 3, 4}, {5, 6}, {9, 10}	8	3	5
5	{3, 4, 5, 6, 7, 8}, {9, 10}	8	2	4

Table 4.10: Candidates for selection of premerged spatial objects

This is implemented by finding the first match $g \subseteq T$, extracting t, and then recursively repeating for $T - g$.

All maximal groups G are stored in the pool of candidates Γ.

3. For each group $G \in \Gamma$ calculate the following parameters:
   - *coverage*: number of objects in G
   - *num_of_precomputed_objects*: number of premerged groups in G
   - *num_of_on_line_merges*: number of objects that have to be merged on-the-fly in order to compute the target T.

   Clearly, the following equation holds:

   $num\_of\_on\_line\_merges = cardinality(T) - coverage + num\_of\_precomputed\_objects$

4. Among all groups in Γ select the group with the smallest value for *num_of_on_line_merges*.

Example 11 Suppose a spatial measure cell resulted from an OLAP operation contains the following set of spatial pointers: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Assume that the search of the *merged_obj_table*, M (Step 1) finds the following set of precomputed objects: {1, 2, 3}, {1, 2, 3, 4}, {3, 4, 5, 6, 7, 8}, {4, 7, 8}, {5, 6}, {7, 8, 9, 10}, {9, 10}.

Step 2 derives the sets of candidates, shown on Table 4.10. In Step 4 group No. 3 is selected as the answer since only three objects have to be merged on-the-fly.

□

Chapter 5

Implementation and Experiments

This chapter consists of two parts, the presentation of GeoMiner: a system prototype for spatial data and mining, and the experimental results of our research. Since the GeoMiner system prototype has resulted from a joint work of GeoMiner Research Group members, the author of this thesis will focus on his part, OLAP engine, and its importance for data mining modules. Since only a portion of the presented research has been implemented in the GeoMiner system, a simulation-based performance study was conducted.

5.1 Design and Implementation of the GeoMiner system

The GeoMiner system [36] is an extension and evolution from a relational data mining system DBMiner [35] researched and developed in Intelligent Database Systems Research Laboratory at Simon Fraser University. At the present time the DBMiner system contains the following five data mining functional modules: *characterizer*, *comparator*, *associator*, *classifier*, and *predictor*. Several additional data mining modules, including mining from time-related data, are at research and development stage. We advise the reader to look at numerous publications that explain DBMiner structure and its data mining techniques [32, 33, 34, 35, 43, 44].

We are aware that data mining is unsupervised learning, and that a user has to direct the discovery process. Therefore, an important issue in designing and developing a data mining system is providing a user with an easy and a straightforward way to formulate his/her mining requests. Due to our belief that the success of relational databases should be credited in part to the creation of the standardized query language - SQL, we suggest that the underlining structure of a proposed data mining language be SQL. In addition to expressing mining requests, in the form of rules, the proposed language has to handle spatial predicates, such as *close to, contains, intersects*, etc. We suggest a GMQL - Geo-Mining Query Language [37] for formulating mining requests.

An adequate presentation of discovered knowledge is yet another significant issue for a data mining system and dealing with spatial data adds even more importance to it. Only if spatially-related knowledge is visualized, can it be interesting to a knowledge worker. Consequently, we designed and implemented various visualization tools for all types of discovered knowledge.

5.1.1 System Architecture

The GeoMiner system is constructed on top of the DBMiner system. Mining of nonspatial data is directed to the DBMiner system; whereas mining of spatial data and the relationships between spatial and nonspatial data are performed by the dedicated GeoMiner functions. The general architecture of the GeoMiner system, presented on Figure 5.1, features five units:

1. A set of discovery modules that includes: *geo-characterizer*, *geo-comparator*, *geo-associator*, *geo-cluster analyzer*, and *geo-classifier*. We plan to develop two additional modules: *geo-predictor* and *geo-pattern analyzer*.

2. A data cube mining engine based on DBMiner discovery kernel for multidimensional manipulation of data.

3. A spatial database server, included within MapInfo Professional 4.1 Geographic Information System.

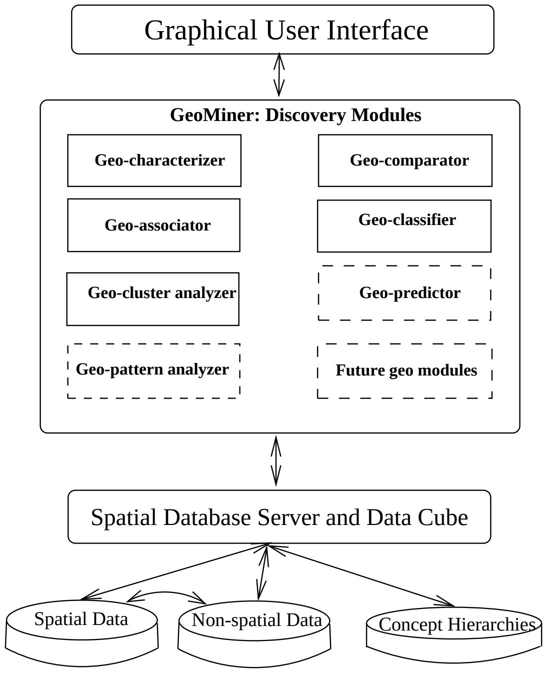

Figure 5.1: General architecture of GeoMiner

4. A graphical user interface for interactive data mining and for display of mining results in the form of tables, charts, maps, etc.

5. The data-, and knowledge-base, storing nonspatial and spatial data and their concept hierarchies.

The functionalities of the five existing discovery modules are outlined as follows.

- Geo-characterizer finds a set of characteristic rules at multiple levels of abstraction from a relevant set of data in a spatial database. It provides users with a multiple level and a multiple angle view of the data in a spatial database. By using this module the following question, for example, can be answered: *"Given spatial hierarchies of the United States, what are general income patterns according to region partitions?"*.

- Geo-comparator discovers a set of comparison rules that contrast the general features of different classes of the relevant sets of data in a spatial database. It compares one set of data, known as the *target class*, to the other set(s) of data, known as the *contrasting class(es)*. For example, this module may show *the differences in migration patterns in the United States* , or find *the clusters or features related to the locations which differentiate the profiting stores from the non-profiting ones.*

- Geo-associator extracts a set of strong spatially-related association rules from a relevant set of data in a spatial database. An association rule shows the frequently occurring patterns (or relationships) a database [49]. A typical spatial association rule is in the form of "$X \rightarrow Y$ ", where X and Y are sets of spatial and/or nonspatial predicates. For example, an association rule can reveal *the relationships between golf courses and other nearby objects like parks, roads, lakes, etc.*

- Geo-cluster analyzer uses an efficient algorithm CLARANS [54] to perform spatial clustering. After it detects clusters, *geo-cluster analyzer* finds nonspatial descriptions of the clusters by using the attribute-oriented induction method [32]. For example, one can find *how the stores are clustered and then, find descriptions for each cluster to determine appropriate marketing strategies.*

- Geo-classifier adopts a generalization-based decision tree induction method to build a classification tree that classifies a set of relevant data according to one of the nonspatial attributes [44]. The classification tree is displayed and by clicking on any of the nodes of the tree a user highlights corresponding region(s) on the map. For example, one may *classify states in the United States according to the median family income in a state.*

The data mining modules described above use spatial database server to extract data relevant to the data mining process. The spatial database used for the implementation, MapInfo Professional, provides a connection to many external databases using

ODBC functionalities, import of data from other spatial data formats, and querying of spatial data using native version of Spatial SQL.

The data retrieved from the databases is analyzed by data mining modules. The data cube underlying architecture enables fast manipulation (*roll-up, drill-down, slicing, dicing*) and analysis of large amount of data. The multidimensional data analysis utilizes concept hierarchies, which are stored in a database. With the ascension of a concept hierarchy, information becomes more general, but still remains consistent with the lower concept levels. Take *occupation* concept hierarchy as an example. Both *VLSI design engineer* and *systems engineer* can be generalized to concept *computer engineer* which in turn can be generalized to concept *engineer*, which includes *mechanical engineer* as well. A similar hierarchy may exist for spatial data. For example, in a generalization process, regions representing counties can be merged to states and states can be merged to larger regions. Concept hierarchies can be built based on the expert knowledge or, in the case of numerical concept hierarchies, created automatically. The relational concept hierarchies are stored as tables in a relational database. The spatial concept hierarchies contain precomputed spatial aggregations to accelerate data analysis process.

5.1.2 Implementation of OLAP in the GeoMiner System

One of the essential features of GeoMiner is its ability to perform multilevel spatial data mining and spatial data analysis. Data in spatial databases usually contains detailed information at the primitive level of abstraction, also known as *raw data.* It is desirable to summarize a large quantity of data, and present it at a high abstraction level. For example, given climate data for a region, one may want to summarize this detailed data, and present the general characteristics of the region's climate. Such a process would generalize raw data into concepts like *cold, mild, hot* (for temperature), *dry, wet* (for precipitation), etc. Portions of the map (regions) that are described by the same high level concepts can be merged together. However, with a blind generalization, data may be summarized to too high a level, and provide a common sense knowledge. Thus, we stress the interactive and multilevel paradigm of the

GeoMiner system.

The system provides not only the power of generalization (*roll-up*), but also the power of specialization (*drill-down*). While roll-up or reduction of a dimension is a relatively simple concept, drill-down requires more explanation. One may wonder how a drill-down operation can be performed and whether it introduces an additional level of complexity. Obviously, it cannot be accomplished by working directly on a high level cuboid. A value, once generalized, cannot be restored without being saved beforehand. Therefore, we maintain a *base cuboid*, also known as the *least generalized cuboid*. When drilling-down to a specific cuboid, we effectively roll-up from the base cuboid. In addition to typical OLAP operations on nonspatial data, the GeoMiner system allows for OLAP on spatial data, also known as spatial OLAP. The OLAP component of the GeoMiner system consists of two modules: *characterizer*, and *comparator*, that produce *characteristic* and *comparison* rules respectively. We now examine these types of rules in more detail by going through two examples.

Example 12 Suppose that a user wants to *characterize (summarize) demographic patterns in the United States* by presenting the relationships among the regions with respect to population size, median family income, and percentage of people holding bachelor degree. This data mining request can be formulated with the following GMQL query.

```
MINE CHARACTERISTICS
AS "USA_states"
ANALYZE sum(pop)
WITH RESPECT TO geo, statename,
        pop, med_fam_income, with_bachelor_degp
FROM states_census
WHERE time = 1996
```

First, the system retrieves the data related to *name of the state* (statename), *population* (pop), *median family income* (med_fam_income), *percentage of population holding bachelor degree* (with_bachelor_degp), and with *time* being the year 1996. *Geo* is a spatial attribute that corresponds to a map object.

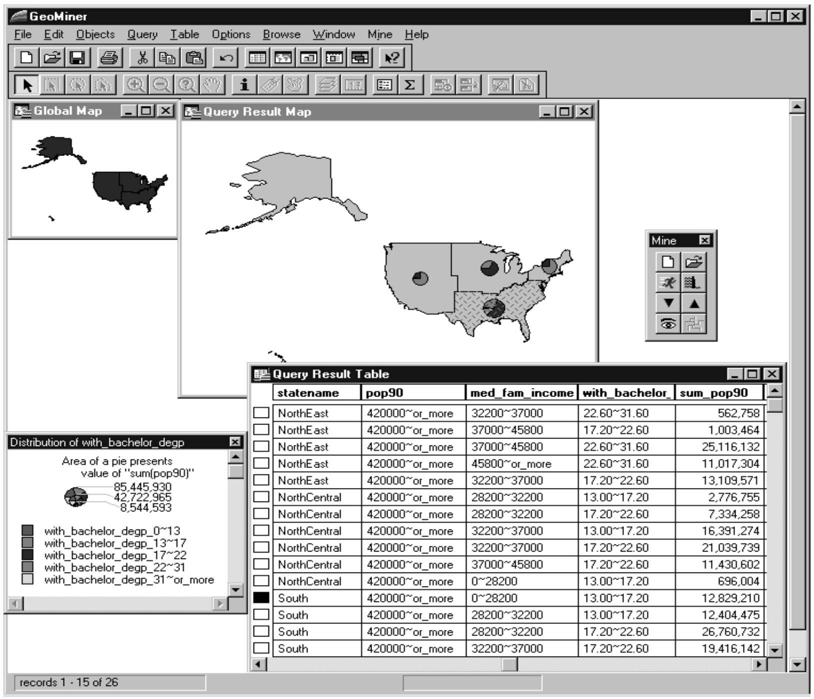

Figure 5.2: Display of spatial characteristic rules

Based on the given query and the spatial concept hierarchy for geo attribute and nonspatial concept hierarchies for pop, med_fam_income, and with_bachelor_degp attributes a spatial-dominated generalization is performed as follows [53].

Spatial-dominated generalization triggers the merge of the connected regions, and creates a set of larger merged regions. Effectively, spatial descriptions of states, *geo*, are generalized into larger regions like *New England*, *Middle Atlantic*, *North Central*, etc. Ascending the spatial concept hierarchy leads to fewer objects. Generalization of the spatial objects continues until the *spatial generalization threshold* is reached. The spatial generalization threshold is defined as the maximum number of regions in a *generalized relation*. Generalized relation is a table that uses generalized values for attributes (a table at the right-hand side in Figure 5.2).

After the spatial generalization process is performed, the nonspatial data is retrieved and analyzed for each of the generalized spatial objects using the attribute-oriented induction method [32]. Three main steps of the attribute-oriented induction technique are:

1. climbing the concept hierarchy when attribute values in a tuple are changed to the generalized values
2. removing attributes when further generalization is impossible and/or there are too many distinct values for an attribute
3. merging identical tuples

The induction continues until a value for every attribute is generalized to the desired level, specified by the *generalization threshold* for that attribute. During the process of merging identical tuples, the number of merged tuples is stored in additional attribute *count* to enable quantitative presentation of the acquired knowledge. In our example, sum of population for states in merged regions is also stored to enable presentation of the measure *"sum(pop)"*. Notice that a single spatial object can be described by multiple tuples in a generalized relation. This is the result of having different nonspatial descriptions for objects that generalize to the larger object.

The above process creates a generalized map of the United States (Figure 5.2). However, we believe that at this moment, fruitful and interactive OLAP is yet to start. Drill-down or roll-up can be performed interactively on such generalized data to zoom-in or zoom-out the generalized spatial regions or to examine the details of their associated nonspatial properties. Invoking roll-up and drill-down OLAP operations fetches the cuboids from the lattice of cuboids. The fact that the cuboids contain materialized spatial measures greatly enhances mining at multiple levels of abstraction. Figure 5.3 shows the results after drilling-down along spatial dimension to present details describing the southern part of the United States. The results can be presented in the form of a two-dimensional chart, pie chart on the map, and a generalized relation.

□

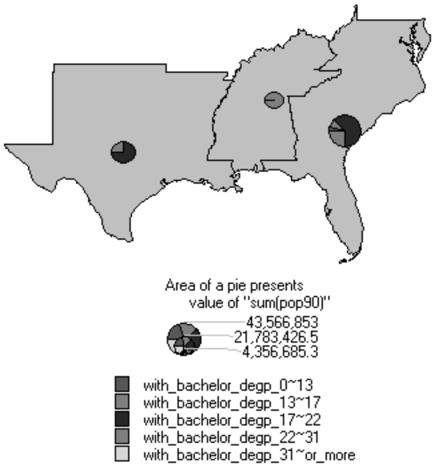

Figure 5.3: Drilling-down along spatial dimension

In the presented example, mining is performed by generalizing first along the spatial hierarchy. Then, this process triggers the attribute-oriented induction process on nonspatial attributes to describe nonspatial properties of generalized spatial regions. Therefore, such generalization is called *spatial data-dominated generalization* [53].

Alternatively to *spatial data-dominated generalization* presented in the above example, *nonspatial data-dominated generalization* can be performed if a spatial concept hierarchy is not given. This method includes the following steps:

1. Collecting all data relevant to the query.

2. Applying the attribute-oriented induction on nonspatial attributes, i.e., generalizing to higher concepts. For instance, *median family income* values in range (10K : 20K] can be generalized to *poor*.

3. Collecting pointers to spatial objects that are described by the same high level nonspatial concepts. These pointers are put in generalized tuples. Again, the

generalization threshold is used to determine when to stop the generalization process.

4. Potential merging of neighboring objects that belong to same generalized tuples.

The results of this process can be visualized in the form of thematic map which present regions according to their generalized nonspatial descriptions. We now describe the last step in a little bit more detail. In some cases, merging of objects that are compact enough (e.g., states) is not necessary and would only add to the overall complexity. Thus, the current version of the GeoMiner system does not perform such a merge. If the merge operation is to be performed, it could be significantly accelerated by precomputing and storing the most frequently used spatial aggregations (merged regions).

Example 13 Suppose that a user wants to find the *migration patterns in the United States*, and to compare regions with large increase in the population from 1980 to 1992 and regions where the population decreased during the same period. This data mining request can be formulated with the following GMQL query.

```
MINE COMPARISON
AS "Migration Patterns"
ANALYZE sum(pop)
WITH RESPECT TO geo, statename,
        pop, crimes100000R, med_fam_income
FROM states_census
FOR "Significant_population_increase"
        WHERE pop80_92P > 20
VERSUS "Population_decrease"
        WHERE pop80_92P < 0
```

The query compares the two classes of objects with respect to *spatial location* (geo), *name of the state* (statename), *population* (pop), *ratio of crimes per 100,000 people* (crimes100000R), and *median family income* (med_fam_income).

To process this query, the system first retrieves the relevant set of data from the relation *states_census*. The collected data is partitioned into three contrasting classes: "*Significant_population_increase*","*Population_decrease*", and "*Others*". The "*Significant_population_increase*" class contains states where population between 1980 and 1992 increased by more than 20% (where pop80_92P > 20). The "*Population_decrease*" class contains states where population decreased in the same period of time. Finally, the "*Others*" class consists of objects that do not belong to either one of the first two classes. The objects belonging to the contrasting classes are generalized to the same levels of concept hierarchies. Then, *roll-up*, *drill-down* and *slice* operations can be applied synchronously on all classes. Generalization and specialization are performed based on the hierarchies associated with the following dimensions: spatial dimension (*geo*) and four non-spatial dimensions: *statename, pop, crimes100000R,* and *med_fam_income*. The measures contain two values, *count* as the default measure and *sum(pop)* to present sum of population in particular regions. An example of the result of the comparison query execution is presented in Figure 5.4.

□

5.1.3 Role of OLAP in Spatial Data Mining

Presentation of summarized data is only one of the reasons for performing spatial OLAP operations. More importantly, we see OLAP as a tool that enhances spatial data mining. Previously, we briefly described *geo-associator*, *geo-cluster analyzer*, and *geo-classifier* modules. The attractiveness of these mining modules is at their ability to discover multilevel knowledge from a spatial database. When performing either generalization or progressive deepening it is essential that the response time be small. Underlined data cube technology provides the GeoMiner system with a reasonably fast response time.

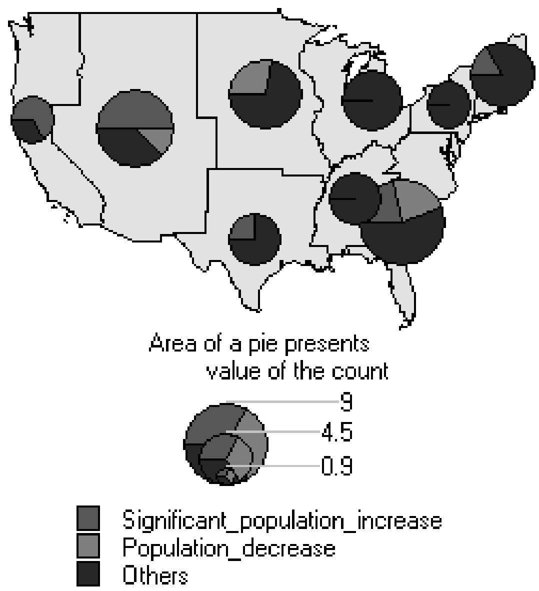

Figure 5.4: Results of a comparison query

5.2 Performance Analysis of Proposed Algorithms

In Section 4.2, we presented three algorithms for selective materialization of spatial measures: *spatial greedy* algorithm, *pointer intersection* algorithm, and *object connection* algorithm. In order to evaluate and compare their effectiveness and efficiency, we implemented the algorithms and conducted a simulation study.

The simulation is controlled by using the following parameters:

- *number_of_objects* on the map. Since the only spatial data type that we discussed so far is a region, the map is comprised of regions only.
- *max_number_of_neighbors* for an object.
- *number_of_cuboids* selected by the HRU cuboid-selection algorithm [38].
- *min_number_of_tuples* as the number of tuples in the most generalized cuboid.

- *max_number_of_tuples* as the number of tuples in the least generalized cuboid.
- *max_frequency* as the maximal access frequency of a cuboid.
- *perc_of_groups* to materialize (for *spatial greedy* algorithm).
- *min_freq_ratio* as the ratio (for *pointer intersection/ object connection* algorithm) between minimal frequency threshold and average access frequency of all cuboids. Minimal frequency threshold is used to filter out groups of spatial objects that have low access frequency. Specifying the frequency threshold is done in an indirect way. We enter a *ratio* between the threshold and the average cuboid frequency, because it gives more control over the simulation.

We now explain the major steps of our simulation.

1. Generating a map.
   A map is created by specifying values for *number_of_objects* and *max_number_of_neighbors*. We do not take into account size of the map objects (regions) and the number of vertices.

2. Setting up a spatial data warehouse.
   We create a spatial data warehouse by specifying three parameters: *number_of_cuboids*, *min_number_of_tuples* in a cuboid, and *max_number_of_tuples* in a cuboid. Being primarily concerned with spatial measures, we do not compose values for dimensions, and numerical measures. Instead, we generate a set of pointers to spatial objects for each tuple in each cuboid. In addition each cuboid is assigned a random frequency in range (0:*max_frequency*].

3. Selective materialization of spatial measures.
   We choose and execute an algorithm. For *spatial greedy* algorithm we specify a value for *perc_of_groups* to be materialized, whereas for the other two algorithms we specify a value for *min_freq_ratio* threshold. The chosen algorithm selects groups of spatial pointers to be merged and stored in a spatial data cube.

4. Simulating typical OLAP operation on the spatial data warehouse.
   We simulate posing of queries to the spatial data warehouse. Here, we use cuboid frequencies generated beforehand. We also pose queries that are infrequent (do not appear in the set of selected cuboids).

The map, content of tuples, and cuboids' access frequencies are generated randomly following uniform distribution. We believe that other distributions should not significantly affect the relative performance and thus would lead to similar conclusions. The simulation was performed on a Pentium 200MHz machine running Windows NT 4.0 operating system. The simulation code was written entirely in Microsoft Visual C++ 4.2.

In our presentation of all three algorithms, we assumed the existence of *obj_neighbor* table that accelerates checking connectivity of objects on the map. However, creation of such a table is likely to take some substantial processing effort and thus there is an option to perform neighborhood test directly on the map objects.

We divide our performance analysis study into two parts. First, we analyze *effectiveness* of the algorithms. Since the goal of selective materialization is to enhance on-line processing, we study the usability of materialized groups. We are interested in knowing (1) how much on-line processing time is decreased by performing off-line precomputation, and (2) what is the storage overhead that such precomputation yields. Second, we compare *efficiency* of the algorithms in terms of precomputation running time. Although precomputation running time is not even distantly as crucial as on-line running time, we are still concerned with precomputation efficiency. The main reason for this concern is maintenance of a spatial data warehouse. Even though spatial objects may not very frequently change, their nonspatial attributes can change. Since we consider merge of objects with same nonspatial descriptions, updates of measures (both spatial and nonspatial) may be quite frequent.

5.2.1 Effectiveness of the Algorithms

In this subsection we examine effectiveness of the proposed algorithms. We are mainly interested in determining the benefits that precomputation generates and the resulted storage overhead. We now define a few terms used throughout the subsequent analysis.

- *saving in the number of disk accesses*
  The goal of materialization of spatial measures is getting short response time for OLAP operations. If n spatial objects are to be merged during on-line processing it would take $n + 1$ disk accesses to read n objects and store the resulting - merged object. If these n objects are premerged only one disk access (read) is needed. Moreover, since we focus on computation of spatial measures we count only disk accesses to objects that are to be merged (n in the above example). Note that these objects may be original spatial objects or already premerged objects. Thus, we define *saving in the number of disk accesses* as the percentage of disk accesses that are avoided (not necessary) due to off-line premerge.

- *storage overhead*
  Materialization of spatial measures may yield large storage overhead. We present *storage overhead* as the ratio between total storage space needed for spatial measures and the space for the original map objects.

We first analyze *spatial greedy* algorithm in isolation, because of its different stopping criterion from that of the other two algorithms. Figure 5.5 shows the effectiveness of *spatial greedy* algorithm. The figure illustrates the benefits of selective materialization, expressed as saving in the number of disk accesses during on-line processing.

Note that in this experiment we considered only OLAP queries whose results are (partially) materialized (chosen by the HRU algorithm). Later in this subsection we will examine OLAP queries that are not chosen by the HRU algorithm.

The slope for the benefit curve in Figure 5.5 decreases when the number of materialized groups increases. Our explanation to this is that the initial premerge obtains a large boost due to that many of the merged regions are shared by different nodes and that the premerge leads to a relatively big reduction ratio on the sizes of the merged nodes. Such initial materialization really benefits many subsequent OLAP operations. When a certain proportion of all mergeable groups is materialized, the candidates with such nice features may have been used up and the benefit becomes

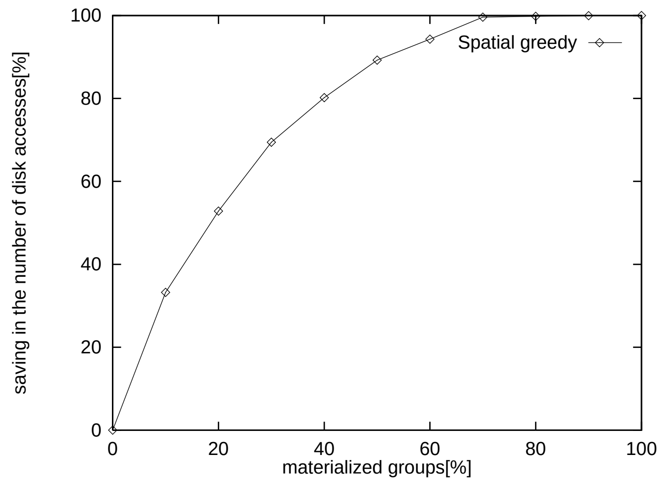

Figure 5.5: Spatial greedy algorithm: benefits of materialization

marginal. However, we feel that the simulation study only discloses such a potential trend. The concrete savings and when the saturation point is reached can be only told by experiments with a large number of spatial objects in the real world situation.

Based on available storage space, the spatial data warehouse designer should determine the number of premerged groups. According to the experiment described above, the materialization of a small portion of groups leads to reasonably good performance in terms of the trade-off between response time and the storage space.

The remaining two algorithms, *pointer intersection* and *object connection*, are compared with respect to their effectiveness in Figure 5.6. Here, we analyze benefit as a function of *min_freq_ratio*, introduced earlier in this section. The figure reveals the following:

- The benefits of both algorithms decrease with the increase of frequency threshold.
- There is only a slight difference between effectiveness of the two algorithms.
- If self-intersection is not applied, the benefits do not converge to 100%.

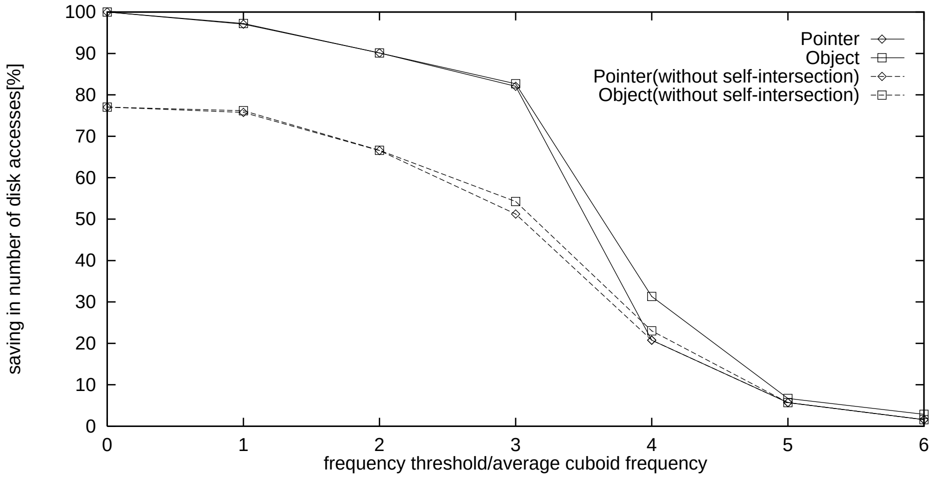

Figure 5.6: Pointer intersection and object connection algorithms: benefits of materialization

It was expected that a higher frequency threshold leads to the smaller number of premerged objects and thus to the smaller benefit. Like in the case of *spatial greedy* algorithm, benefits for these two algorithms converge to 100% (reach 100% percent when no frequency threshold is applied). The slight difference between effectiveness of *pointer intersection* and *object connection* algorithms follows from Theorem 4.2.1. The difference is more or less marginal in our simulation, however only the real world application with a large number of spatial objects can confirm a potential trend. As we explained in our presentation of *pointer intersection* and *object connection* algorithms, performing the self-intersection on cuboids vastly improves the benefits of materialization. If the self-intersection were not applied, at most 77% of disk accesses would be avoided during on-line processing. When the frequency threshold increases the differences become marginal.

We used the results gathered in the previous two experiments to compare the effectiveness of all three algorithms. The objective of this performance study was to determine which algorithm selects best candidates for premerge and under which conditions. In order to compare all three algorithms, we had to find a common denominator for the algorithms, and the natural choice was *storage overhead.* Note

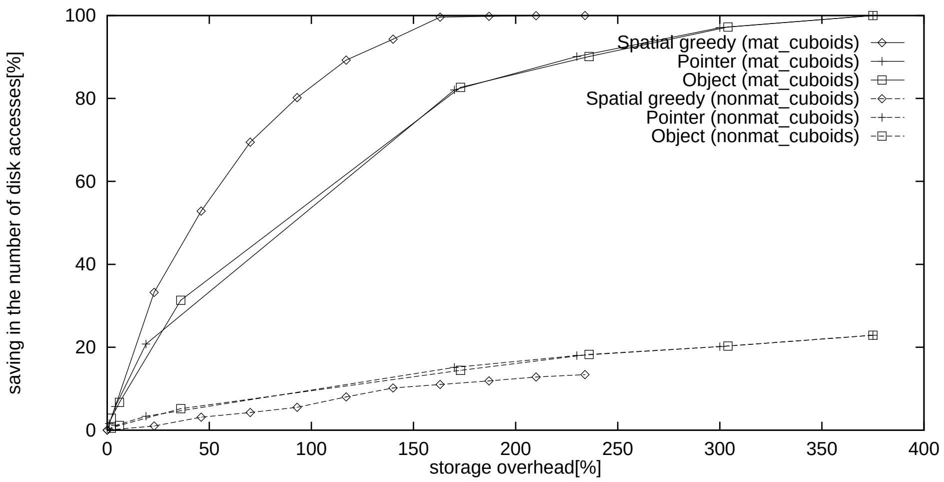

Figure 5.7: Comparison of algorithms: benefit of materialization

that storage overhead is not an input parameters for any of the algorithms.

Until now, we have looked at benefits of precomputation for partially or fully materialized cuboids, i.e., chosen by the HRU algorithm. We are, in addition, interested in improving the response time for all other OLAP queries (those that are not even partially materialized). Thus, in the following experiment we generated a number of queries that simulate on-line processing, and checked how much the premerged objects can improve their response time. Being mainly interested in spatial measures we created only groups of spatial pointers as an answer to a query. The number of such queries was as high as 100,000.

Figure 5.7 shows that:

- For materialized cuboids, *spatial greedy* algorithm reaches the saturation point faster than the other two algorithms do.

- *Pointer intersection* and *object connection* algorithms are better at handling non-materialized cuboids.

This is analyzed as follows. First and foremost, it is important to realize that *pointer intersection* and *object connection* algorithms select candidate groups from a larger pool than *spatial greedy* algorithm does. Let us illustrate this with a simple example. Suppose the group (A,B) appears in two or more tuples (in different cuboids), but is always accompanied by some other connected objects. Thus, this group will not be detected by the greedy algorithm (groups like $(A,B,\ldots)$ will be detected instead), but it may be an intersection between at least two cuboids. On the other hand every mergeable group extracted by the greedy algorithm, is also a self-intersection of a cuboid it belongs to.

Consequently, *pointer intersection/object connection* algorithm selects a number of small cardinality groups that are common to a number of cuboids. Note that a premerged object may be used for answering an OLAP query only if it is fully contained in one of the resulting tuples (it cannot be contained within more than one tuple for a single query). Since premerged objects are computed on the basis of materialized cuboids (chosen by the HRU cuboid-selection algorithm), all these objects are legitimate candidates for answering at least one query. In general, premerged objects consolidated from a large number of original map objects are better candidates than the smaller ones. Thus, by not selecting low cardinality groups *spatial greedy* algorithm utilizes storage better than the other two algorithms do. For the above reasons, *spatial greedy* algorithm outperforms *pointer intersection* and *object connection* algorithms in answering the OLAP queries whose results are materialized.

Exactly the opposite happens when the queries whose results are not materialized are posed. The likelihood of fitting large (in terms of inner cardinality) premerged objects into resulting ones is small. Thus, it is more beneficial to use small objects generated by *pointer intersection* and *object connection* algorithms. This explains the bottom part of Figure 5.7. However, there is an issue of accessing premerged objects that may not be overlooked when considering non-materialized queries. The access to spatial measures of a materialized cuboid is fast due to a highly indexing structure of a spatial data cube. Answering to non-materialized queries requires search for best candidates among premerged objects. Note that cuboids for such queries are not created off-line. Intuitively, premerged objects can be organized in a hierarchical structure such as R^*-tree structure. Nonetheless, further research in this direction is necessary.

Another observation from Figure 5.7 is that curves for *pointer intersection* and *object connection* algorithms intersect. This simply shows that additional objects (see Theorem 4.2.1) premerged by the latter algorithm are not always very useful for on-line processing.

The above analysis leads to the following conclusion: If only queries with materialized results are to be run against the spatial data warehouse, *spatial greedy* algorithm should be used. On contrary, if there is no specific pattern in the usage of the data warehouse and there are few queries whose results are materialized, *pointer intersection* or *object connection* algorithm should be used. We believe that latter conditions are more realistic in a real world application.

5.2.2 Efficiency of the Algorithms

The fact that *spatial greedy* algorithm has different stopping criterion than the other two algorithms do, makes us unable to strictly compare efficiency of all three algorithms. Thus, we first discuss the efficiency of *spatial greedy* algorithm and then compare the efficiencies of *pointer intersection* and *object connection* algorithms. Despite this limitation, we will suggest the favorable conditions for each of the algorithms. We elected to vary two parameters: *number_of_objects* and *number_of_cuboids*. Not only that these two parameters are significant in our simulation study, but also and more importantly, they are vital for a user of a spatial data warehouse. In all experiments, we fix the following parameters: *number_of_neighbors = 10*, *min_number_of_tuples = 5*, *max_number_of_tuples = 100*, *max_frequency = 1000*.

Figure 5.8 shows the execution time as a function of number of objects in the database. In this experiment we fix, the number of cuboids to 10. We can see that *spatial greedy* algorithm is very sensitive to the number of objects in the database. Since we assume that each cuboid covers the whole map, the simulator generates (*number_of_objects / max_number_of_tuples*, *number_of_objects / min_number_of_tuples*) objects for every tuple. Thus, the first step of the algorithm, detection of mergeable groups is very expensive and it creates a large number of groups. Having a large

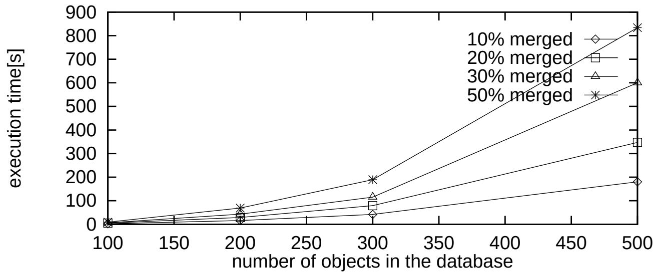

Figure 5.8: Scalability of spatial greedy algorithm as a function of number of map objects

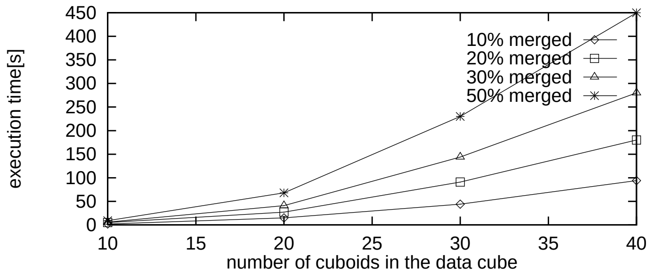

Figure 5.9: Scalability of spatial greedy algorithm as a function of number of cuboids

number of groups does not only increase the number of iterations in the greedy algorithm, but also prolongs the execution of each iteration. Note that benefit for every unmerged group has to be recalculated in every iteration of the greedy algorithm. Not surprisingly, execution time has linear growth with the increase of percentage of groups (*perc_of_groups*) to be materialized.

Figure 5.9 shows the scalability of the algorithm when number of cuboids increases (the number of objects was set to 100). The performance of the algorithm is like in the previous experiment. Thus, we conclude that *spatial greedy* algorithm is equally sensitive to the number of objects and the number of cuboids (views) in the data cube. Later, we will see that this property makes the algorithm very useful under certain conditions.

We now focus on the remaining two algorithms. Figure 5.10 shows the performance

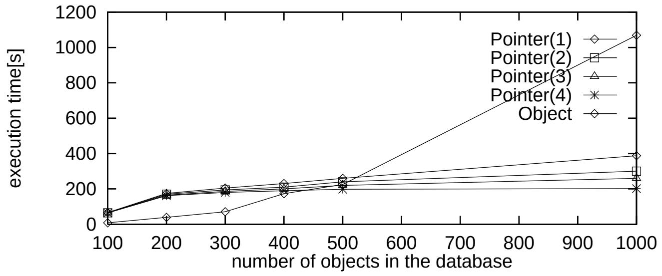

Figure 5.10: Scalability of pointer intersection and object connection algorithms as a function of number of objects

comparison of the algorithms when they are applied on maps with different number of objects. The number of cuboids was set to 10 for this experiment. When the number of objects is small, *object connection* algorithm has an edge over *pointer intersection* algorithm. However, by increasing the number of objects, the performance of *object connection* algorithm significantly deteriorates, while *pointer intersection* algorithm shows little sensitivity to the number of objects.

This is analyzed as follows. The first step of both algorithms is finding the intersecting groups among the tuples in the cuboids. After the intersecting groups are detected, *pointer intersection* algorithm filters groups with low access frequency. In order to perform such a filtering, the algorithm has to detect the total access frequency of every group (see *max_frequency* function in Algorithms 4.2.2 and 4.2.3). The total frequency is a sum of the raw frequency and the accumulated frequency. Computation of the accumulated frequency of a group is a very expensive operation, since the algorithm has to find all groups that contain the group. Since *pointer intersection* algorithm performs the above step for all intersections, there is a large time overhead. In the case of *object connection* algorithm the filtering step is postponed after connectivity test (see *spatial_connectivity_testing* procedure). A large number of groups are eliminated in the connectivity test early in *object connection* algorithm. For the above reasons, *object connection* algorithm outperforms *pointer intersection* algorithm when the map contains small number of objects.

With the small number of objects on the map, tuples in the cuboids contain relatively few pointers to spatial objects. Consequently, cardinality of intersecting groups is small too. Increasing the number of objects leads to the tuples with more pointers, and thus to larger cardinality of intersected groups. However, the number of intersecting groups does not change much, and it converges to

$$M = \sum_{i=1}^{number\_of\_cuboids} (tuples(i) \times \sum_{j=i}^{number\_of\_cuboids} tuples(j))$$

where, M is maximal number of intersections, and $tuples(k)$ is number of tuples in cuboid k.

Thus, the running time for *pointer intersection* algorithm only slightly increases with the increase of the number of objects.

On the other hand, having high cardinality groups introduces a huge processing ballast for connectivity test applied to all intersecting groups early in *object connection* algorithm. Being large, many groups are split into a number of mergeable groups so that, the frequency filtering step applied in *object connection* algorithm, that contains *max_frequency* function becomes more expensive than the very same step in *pointer intersection* algorithm (dealing with more groups).

To conclude, *object connection* algorithm is very sensitive to the increase of the number of objects on a map. Notice that all experiments were conducted with *obj_neighbor* table created off-line.

Another observation from the curves in Figure 5.10 is that the frequency threshold is irrelevant for the execution time of *object connection* algorithm. This was expected since frequency filtering is the last step in the algorithm. On contrary, *pointer intersection* algorithm shows slightly better performance when the frequency threshold increases. Notice that changing the frequency threshold does not influence execution time of costly *max_frequency* function.

We now compare the two algorithms with respect to scalability to the number of cuboids in the spatial data cube. Figures 5.11 (a) and (b) show the performance of the algorithms when applied to maps having 100 and 1000 objects respectively. The curves in the figures show that *object connection* algorithm is superior to *pointer intersection* algorithm when the number of objects is small. The former algorithm is

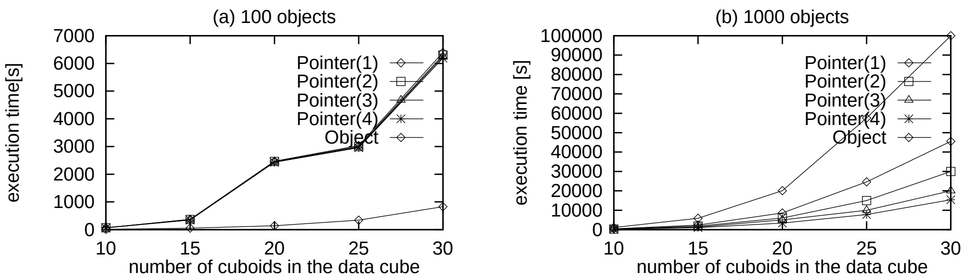

Figure 5.11: Scalability of pointer intersection and object connection algorithms as a function of number of cuboids

very insensitive to the number of cuboids (with small number of objects). However, as soon as the number of objects increases, we get results similar to the ones in Figure 5.10.

We now analyze the observations from Figure 5.11 (a). Having a large number of cuboids directly leads to large number of intersections. While *object connection* algorithm filters a majority of the intersection groups in its connectivity test, *pointer intersection* algorithm considers all the groups in its *max_frequency* function. If the experiment were done without creating *obj_neighbor* table off-line, performances of *object connection* algorithm would be not even closely as good. It is very clear that creation of such a table off-line significantly boosts up the algorithms, especially the *object connection* one. Such a "create-once and use-many-times" model is realistic since it is likely that nonspatial properties of map objects will change more frequently than the map objects themselves.

Figure 5.11 (b) simply confirms that *pointer intersection* algorithm is better at handling large number of objects, and it can be analyzed similarly to Figure 5.10.

We now identify conditions that favor one algorithm over the others.

- We anticipate that future spatial data warehouses will be built on the following two premises: large number of objects and relatively small number of frequent queries. According to the conducted experiments, we believe that *pointer intersection* algorithm fits best into this data warehouse environment. Typical applications that confirm to above assumptions are regional weather pattern analysis, demographic analysis, real estate business, etc.

- However, if the number of frequent queries is large, or there are no queries with prevalent frequency, *spatial greedy* algorithm can be used.

- Lastly, if a spatial data warehouse is to contain few objects a data warehouse designer could choose applying *object connection* algorithm for selective materialization of spatial measures.

Finally, the above analysis, as well the analysis conducted in the previous subsection, shows that *pointer intersection* algorithm is likely to have the best performance among all three algorithms.

Chapter 6

Conclusion

We summarize our research work in this chapter. Discussion on future directions in spatial data warehousing and spatial OLAP follows the summary of the research.

6.1 Summary

In this thesis, we have studied the construction of a spatial data warehouse based on a spatial data cube model, which consists of both spatial and nonspatial dimensions and measures. Accordingly, we have made necessary modifications to the star schema model, (widely used for organizing data warehouses of relational data) that facilitate OLAP operations on the spatial data.

The focus of our study has been on spatial measures and their materialization. We have shown that it is not wise to compute spatial measures on-the-fly, however that materializing them all would lead to dramatic storage needs. Thus, we have proposed three heuristic algorithms for cell-based selective materialization, namely *spatial greedy* algorithm, *pointer intersection* algorithm, and *object connection* algorithm. The performances of the algorithms have been studied and compared with the conditions listed for choosing one over the others. In addition, a method for selection of the best materialized sets of objects from a table storing all the materialized regions has been outlined. We have also suggested an approximation method (*rough measures* algorithm) for computation and storing of spatial measures.

Currently, the methods studied in this thesis are being implemented in our spatial data mining system, GeoMiner[36], which takes spatial OLAP as the essential function module. Moreover, spatial OLAP operations have been integrated with spatial data mining modules in the GeoMiner system.

6.2 Discussion and Future Research Issues

The proposed algorithms and their performance study present convincing arguments for selective materialization of mergeable regions for efficient construction of spatial data cubes and for performing spatial OLAP operations. Nonetheless, we foresee further improvements and extensions of the algorithms. These and some issues that should be considered in the future are outlined below.

- noise handling
  Mergeable regions, discussed in this thesis, are specified as the regions that share some common boundaries. However, it is sometimes desirable to ignore small separations and merge the regions which are located very close. For example, two wheat fields separated by a highway can be considered as one region. This case can be handled by minor modifications of the algorithms, that treat those regions that are separated by very minor (relative) distance as connected (and thus mergeable) regions.

- utilization of materialized spatial measures
  Access to spatial measures of a materialized cuboid is fast due to a highly indexing structure of a spatial data cube. However, answering non-materialized cuboids requires searching for best candidates among precomputed objects. Accordingly, efficient algorithms and data structures should be designed and implemented.

- access frequency information
  Our algorithms for selective materialization of spatial measures assume the existence of information about the cuboid access frequencies. What if there does not exist such information initially? There are several methods to handle this.

One method is to assign initial access frequency only to every level in the lattice of cuboids based on some data semantics or assuming that medium levels are accessed most frequently, and low-levels are accessed less frequently than the higher ones. The frequency counts can be adjusted based on later accessing records. Alternatively, we may choose to materialize every mergeable group accessed initially, record group access frequencies, and choose to throw away the later rarely used groups when disk space runs low.

- size of the regions

  The algorithms consider the access frequency and the cardinality of the mergeable groups but not the concrete size of the mergeable regions. However, some regions could be substantially larger or more complicated and thus take substantially larger space and more computation than the others. A possible solution could be to add the compression ratio benefit which is the ratio in size between the premerged and not premerged regions.

- other spatial measure operations

  The proposed algorithms address only the region merge operations in the spatial measure computation. We believe that the principles discussed here are generally applicable to other spatial measure operations, such as thematic map overlay, spatial join [30, 51], etc. Take map overlay as an example. If a measure in a spatial data cuboid represents an overlay of multiple thematic maps, such as *altitude* and *temperature* maps, it will consist of a nontrivial overlay map both in size and effort of computation. Selective materialization of such overlay for frequently used groups of spatial objects seems to be essential, and thus the studied principles should be applicable. However, the concrete algorithms have yet to be explored.

- automatic generation and dynamic adjustment of concept hierarchies

  One of the important features of our research is the existence of concept hierarchies and we assumed that they are created by users or domain experts.

However, it is preferable to generate them automatically, based on the data distribution. While automatic generation and dynamic adjustment of nonspatial hierarchies have been studied [33], no work has been reported on solving these problems in the domain of spatial concept hierarchies. Further advances in this direction could greatly enhance spatial OLAP, and make it more robust.

- data integration
  Spatial data is usually stored in different industrial firms and government agencies using different data formats. Data formats are both structure-specific and vendor-specific. There have been a lot of work on data integration and data exchange, but with little success. These issues have become crucial with the emergence of data warehouses.

- various application domains
  We believe that the principles of cell-based selective materialization for computation of spatial data cubes are not confined to spatial data only. Other databases which handle complex objects, such as multimedia databases, engineering design databases, will encounter similar problems and it is essential to perform object-based selective materialization as a space/time trade-off for efficient OLAP operations.

Several very important issues, not addressed in this thesis, are the efficient storage, indexing, and incremental update of spatial data cubes, as well as caching of spatial measures. Some of these problems have been extensively studied by the (relational) OLAP community. Some research results, particularly for indexing and incremental update, can be partially applied to spatial data cubes. However, the selectively populated cuboids in a spatial data cube, as opposed to fully populated cuboids in a nonspatial data cube, call for new refined methods.

With the recent success and great promise of OLAP technology, spatial OLAP holds a high promise for fast and efficient analysis of large amount of spatial data. Thus, spatial OLAP is expected to be a promising direction for both research and development in the years to come.

Bibliography

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 506–521, Bombay, India, Sept. 1996.

[2] D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek. Efficient view maintenance in data warehouses. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 417–427, Tucson, Arizona, May 1997.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data*, pages 322–331, Atlantic City, NJ, June 1990.

[4] S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An efficient and robust access method for points and rectangles. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 28–39, Bombay, India, Sept. 1996.

[5] M. H. Brackett. *The Data Warehousing Challenge: Taming Data Chaos*. John Wiley & Sons, 1996.

[6] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. In *Proc. 1993 ACM-SIGMOD Conf. Management of Data*, pages 237–246, Washington, DC, May 1993.

[7] D. Chatziantoniou and K. Ross. Querying multiple features in relational databases. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 295–306, Bombay, India, Sept. 1996.

[8] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26:65–74, 1997.

[9] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proc. 1995 Int. Conf. Data Engineering*, pages 190–200, Taipei, Taiwan, March 1995.

[10] E. F. Codd. A relational model for large shared data banks. *Communications of ACM*, 13:377–387, June 1970.

[11] E. F Codd, S. B. Codd, and C. T. Salley. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. In *E. F. Codd & Associates available at http://www.arborsoft.com/OLAP.html*, 1993.

[12] G. Colliat. OLAP, relational and multidimensional database systems. *SIGMOD Record*, 25:64–69, 1996.

[13] R. J. Earle. Methods and apparatus for storing and retrieving multi-dimensional data in computer memory. In *Arbor software corporation, U.S. Patent 5359724*, October 1994.

[14] H. Edelstein. Faster data warehouses. In *TechWeb*, December 4 1995.

[15] M. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6:86–95, 1994.

[16] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 2nd ed.* Bemjamin/Cummings, 1994.

[17] M. Ester, H.-P. Kriegel, and J. Sander. Spatial data mining: A database approach. In *Proc. Int. Symp. on Large Spatial Databases (SSD'97)*, pages 47–66, Berlin, Germany, July 1997.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, Portland, Oregon, August 1996.

[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. Density-connected sets and their application for trend detection in spatial databases. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pages 10–15, Newport Beach, California, August 1997.

[20] M. Ester, H.-P. Kriegel, and X. Xu. Knowledge discovery in large spatial databases: Focusing techniques for efficient class identification. In *Proc. 4th Int. Symp. on Large Spatial Databases (SSD'95)*, pages 67–82, Portland, Maine, August 1995.

[21] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data*, pages 163–174, San Jose, CA, May 1995.

[22] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[23] E. L. Glaser, P. DesJardins, D. Caldwell, and E. D. Glasser. Bit string compressor with boolean operation processing capability, U.S. Patent 5036457. October 1994.

[24] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, cross-tab and sub-totals. In *Proc. 1996 Int'l Conf. on Data Engineering*, pages 152–159, New Orleans, Louisiana, Feb. 1996.

[25] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–54, 1997.

[26] O. Günther. Efficient computation of spatial joins. In *Proc. 9th Int. Conf. Data Engineering*, pages 50–60, Vienna, Austria, April 1993.

[27] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environment. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 358–369, Zurich, Switzerland, Sept. 1995.

[28] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 157–166, Washington, D.C., May 1993.

[29] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Technical Note 1996 available at http://db.stanford.edu/ ullman/ullman-papers.html#dc*, Stanford University, Computer Science, 1996.

[30] R. H. Güting. An introduction to spatial database systems. *The VLDB Journal*, 3:357–400, 1994.

[31] D. Hackatorn. Reinventing enterprise systems via data warehousing. In *Proc. The Data Warehousing Institute Annual Conference*, Washington, D.C., 1995.

[32] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.

[33] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 157–168, Seattle, WA, July 1994.

[34] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 420–431, Zurich, Switzerland, Sept. 1995.

[35] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanović, B. Xia, and O. R. Zaiane. DBMiner: A system for mining knowledge in large relational databases. In *Proc. 1996 Int'l Conf. on Data Mining and Knowledge Discovery (KDD'96)*, pages 250–255, Portland, Oregon, August 1996.

[36] J. Han, K. Koperski, and N. Stefanović. GeoMiner: A system prototype for spatial data mining. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 553–556, Tucson, Arizona, May 1997.

[37] J. Han, K. Koperski, N. Stefanović, and Q. Chen. Design and implementation of spatial data mining engine: GeoMiner. *Submitted for publication*, June 1997.

[38] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 205–216, Montreal, Canada, June 1996.

[39] W. H. Inmon. *Building the Data Warehouse*. QED Technical Publishing Group, Wellesley, Massachusetts, 1992.

[40] W. H. Inmon. The data warehouse and data mining. *Communications of ACM*, 39:49–50, 1996.

[41] W. H. Inmon, J. A. Zachman, and J. G. Geiger. *Data Stores, Data Warehousing, and the Zachman Framework: Managing Enterprise Knowledge*. McGraw-Hill, 1997.

[42] T. Johnson and D. Shasha. Some approaches to index design for cube forests. *Bulletin of the Technical Committee on Data Engineering*, 20:27–35, 1997.

[43] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, pages 207–210, Newport Beach, California, August 1997.

[44] M. Kamber, L. Winstone, W. Gong, S. Cheng, and J. Han. Generalization and decision tree induction: Efficient classification in data mining. In *Proc. of 1997 Int. Workshop on Research Issues on Data Engineering (RIDE'97)*, pages 111–120, Birmingham, England, April 1997.

[45] D. A. Keim, H.-P. Kriegel, and T. Seidl. Supporting data mining of large databases by visual feedback queries. In *Proc. 10th of Int. Conf. on Data Engineering*, pages 302–313, Houston, TX, Feb. 1994.

[46] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, New York, 1996.

[47] R. Kimball and K. Strehio. Why decision support fails and how to fix it. *SIGMOD Record*, 24:92–97, 1997.

[48] E. Knorr and R. Ng. Finding aggregate proximity relationships and commonalities in spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 8:884–897, 1996.

[49] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proc. 4th Int'l Symp. on Large Spatial Databases (SSD'95)*, pages 47–66, Portland, Maine, Aug. 1995.

[50] K. Koperski, J. Han, and J. Adhikary. Mining knowledge in geographic data. In *Comm. ACM (to appear)*, 1997.

[51] R. Laurini and D. Thompson. *Fundamentals of Spatial Information Systems.* Academic, 1992.

[52] A. Levy, A. Mendelzon, and Y. Sagiv. Answering queries using views. In *Proc. 7th ACM Symp. Principles of Database Systems*, pages 95–104, San Jose, California, March 1995.

[53] W. Lu, J. Han, and B. C. Ooi. Knowledge discovery in large spatial databases. In *Proc. Far East Workshop on Geographic Information Systems*, pages 275–289, Singapore, June 1993.

[54] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 144–155, Santiago, Chile, September 1994.

[55] P. O'Neil and G. Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Record*, 24:8–11, September 1995.

[56] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[57] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Proc. 1996 Int. Conf. Parallel and Distributed Information Systems*, pages 158–169, Miami Beach, Florida, December 1996.

[58] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[59] S. Sarawagi. Indexing OLAP data. *Bulletin of the Technical Committee on Data Engineering*, 20:36–43, 1997.

[60] S. Sarawagi. Techniques for indexing OLAP data. In *Technical Report, IBM Almaden Research Center*, 1997.

[61] A. Shoshani. OLAP and statistical databases: Similarities and differences. In *Proc. 16th ACM Symp. Principles of Database Systems*, pages 185–196, Tucson, Arizona, May 1997.

[62] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database research: Achievements and opportunities into the 21st century. *SIGMOD Record*, 25:52–63, March 1996.

[63] D. Sristava, S. Dar, H. V. Jagadish, and A. V. Levy. Answering queries with aggregation using views. In *Proc. 1996 Int. Conf. Very Large Data Bases*, pages 318–329, Bombay, India, September 1996.

[64] J. Ullman and J. Widom. *A First Course in Database Systems.* Prentice-Hall, 1997.

[65] J. Widom. Research problems in data warehousing. In *Proc. 4th Int. Conf. on Information and Knowledge Management*, pages 25–30, Baltimore, Maryland, Nov. 1995.

[66] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, pages 103–114, Montreal, Canada, June 1996.

[67] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data*, pages 159–170, Tucson, Arizona, May 1997.