Object-Based Selective Materialization for Efficient Implementation of Spatial Data Cubes

Nebojsa Stefanovic, *Member*, *IEEE Computer Society*,
Jiawei Han, *Member*, *IEEE Computer Society*, and
Krzysztof Koperski, *Member*, *IEEE Computer Society*

Abstract—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 OLAP of spatial data. In this paper, we study methods for spatial OLAP, by integration of nonspatial 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 selective materialization of the spatial objects resulted from spatial OLAP operations. The focus of our study is on a method for spatial cube construction, called *object-based selective materialization*, which is different from *cuboid-based selective materialization* proposed in previous studies of nonspatial data cube construction. Rather than using a cuboid as an atomic structure during the selective materialization, we explore granularity on a much finer level, that of a single cell of a cuboid. Several algorithms are proposed for object-based selective materialization of spatial data cubes and the performance study has demonstrated the effectiveness of these techniques.

Index Terms—Data warehouse, data mining, online analytical processing (OLAP), spatial databases, spatial data analysis, spatial OLAP.

1 Introduction

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 [22]. Following the trend of the development of data warehousing and data mining techniques [2], [7], [13], [15], we propose to construct *spatial data warehouses* to facilitate online spatial data analysis and spatial data mining [4], [5], [6], [11], [14], [16], [17], [18], [19], [20]. Similar to nonspatial data warehouses [2], [13], we consider that a spatial data warehouse is a *subject-oriented, integrated, time-variant*, and *nonvolatile* collection of both spatial and nonspatial data in support of management's decision-making process. In this paper, we study how to construct such a spatial data warehouse and how to implement efficient *online 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 (BC), 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 even may 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 BC close to 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 be about

- *N. Stefanovic is with Seagate Software, 840 Cambie St., Vancouver, BC, Canada V6B 4J2. E-mail: nebojsa.stefanovic@seagatesoftware.com.*
- *J. Han is with the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6. E-mail: han@cs.sfu.ca.*
- *K. Koperski is with MathSoft, Inc., Suite 500, 1700 N. Westlake Ave., Seattle, WA 98109. E-mail: krisk@statsci.com.*

Manuscript received 13 Apr. 1998; revised 8 July 1999; accepted 21 Sept. 1999.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 106716.

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 any dimension 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 not only show some interesting applications of spatial data warehouses, but also indicate that there are some 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 industry 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 has been a lot of work on data integration and data exchange. In this paper, 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 a spatial data warehouse to make it consistent and up-to-date will not be addressed in this paper either. The second challenge is the realization of fast and flexible online 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 [3], [9], [10]. Unfortunately, these methods alone cannot provide sufficient support for OLAP of spatial data because spatial OLAP operations usually 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. Like in the case of OLAP of relational data, in order to achieve adequate performance, it is necessary to precompute and store some aggregates. Spatial indexing and accessing methods are not designed for such tasks. On the other hand, data warehouse model and OLAP techniques used in the realm of relational data are not fully adequate for handling spatial data. Thus, new models and techniques should be developed for online analysis of voluminous spatial data.

In this paper, we propose the construction of 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 some 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. Previous studies on selective materialization of data cubes select appropriate *cuboids* for materialization [2], [12]. Our approach explores selective materialization at finer granularity than at the cuboid level; we consider the selective materialization at the cell level, that is, only some cells of a cuboid will be materialized. A cell of a cuboid is associated with a spatial object resulted from merging of spatial objects which have the same nonspatial properties. Note that merging spatial (vector or raster) objects is a computationally expensive operation [10]. This object-based approach reflects a trade-off between space and time for efficient implementation of spatial OLAP operations. On the one hand, it is important to precompute some spatial OLAP results, such as merge of spatially connected regions. This is essential not only for fast response in spatial OLAP, but also, and often more importantly, for sophisticated, multi-dimensional spatial data analysis and spatial data mining, such as spatial association, classification, etc. [11]. On the other hand, spatial OLAP often generates a large number of new spatial objects which take huge amounts of storage space if they are all materialized, even just considering one cuboid. Thus, we have to consider to first materialize the frequently used and shared spatial objects.

The remainder of the paper is organized as follows: In Section 2, we introduce a model of spatial data warehouse and a spatial data cube structure. In Section 3, we analyze the methods for computing spatial measures and propose three algorithms for object-based selective precomputation of spatial measures. The performance study of the proposed algorithms is presented in Section 4. In Section 5, we discuss the necessity of selective materialization at the cuboid cell level in other applications, including the case not confined to spatial databases. Other considerations in the construction of spatial data cubes and implementation of spatial OLAP operations are also discussed. The study is summarized in Section 6, with some future research issues proposed as well.

2 A Model of Spatial Data Warehouses

Unlike relational or entity-relationship models which are used for designing databases for ad hoc querying and online transaction processing, data warehouse is designed for online analytical processing and business decision making, and it usually adopts a star schema model [2], [15], where the data warehouse contains a large central table (*fact table*) and a set of smaller attendant tables (*dimensional tables*) joined to the central table. The fact table stores the keys of multiple dimensions and the numerical measures and the dimensional tables store the textual description of the dimensions. A variant of a star schema model is called a snowflake (schema) model [2], [15], where some dimension tables are normalized, further split into more tables, forming the shape similar to a snowflake. With such a star/snowflake schema model, multidimensional databases or data cubes [1], [12] can be constructed to facilitate typical

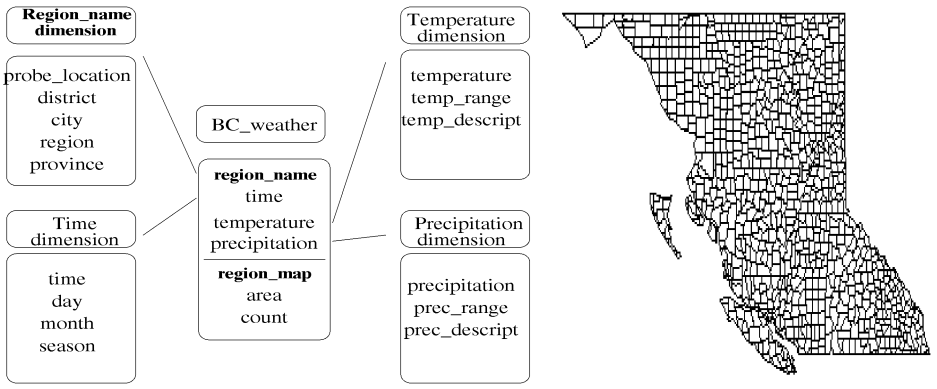

Fig. 1. Star model of a spatial data warehouse: BC_weather and corresponding BC weather porbes map.

OLAP operations, such as *drill-down, roll-up, dicing, slicing, pivoting*, etc.

To model spatial data warehouses, the star/snowflake schema model is still considered to be a good choice because it provides a concise and organized warehouse structure and facilitates OLAP operations. In a spatial warehouse, both dimensions and measures may contain spatial components. A spatial data cube can be constructed according to the dimensions and measures modeled in the data warehouse.

There are three types of *dimensions* in a spatial data cube:

1. Nonspatial dimension is a dimension containing only nonspatial data. For example, two dimensions, *temperature* and *precipitation*, can be constructed for the 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 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 US map is spatial data. However, each state can be generalized to some 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 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.

Notice that the last two 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*.

Moreover, a computed measure can be used as a dimension in a 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 [5], [20], spatial classification [4], [18], or spatial association analysis [17].

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

1. 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 get the total revenue by year, by county, etc. Numerical measures can be further classified into *distributive*, *algebraic*, and *holistic* [8]. 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*; otherwise, it is *holistic*, 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 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* will be grouped into the same cell, and the measure so formed contains a collection of pointers to those regions.

A nonspatial data cube contains only nonspatial dimensions and numerical measures. If a spatial data cube contained spatial dimensions but no spatial measures, its OLAP operations, such as drilling or pivoting, could be implemented in a way similar to nonspatial data cubes. However, the introduction of spatial measures to spatial data cubes raises some challenging issues on efficient implementation, which will be the focus of this study.

The star model of Example 1 and its corresponding dimensions and measures are illustrated as follows:

Example 4. A star model can be constructed, as shown in Fig. 1, for the *BC_weather* warehouse of Example 1. The data warehouse consists of four dimensions: *temperature*, *precipitation*, *time*, and *region_name*, and three measures: *region_map*, *area*, and *count*. A concept hierarchy for each

Region_name:	Time:
probe location $\subset$ district $\subset$ city $\subset$ region $\subset$ province	hour $\subset$ day $\subset$ month $\subset$ season
Temperature:	Precipitation:
any $\supset$ (cold, mild, hot)	any $\supset$ (dry, fair, wet)
cold $\supset$ (below -20, -20 to -10, -10 to 0)	dry $\supset$ (0 to 0.05, 0.05 to 0.2)
mild $\supset$ (0 to 10, 10 to 15, 15 to 20)	fair $\supset$ (0.2 to 0.5, 0.5 to 1.0, 1.0 to 1.5)
hot $\supset$ (20 to 25, 25 to 30, 30 to 35, above 35)	wet $\supset$ (1.5 to 2.0, 2.0 to 3.0, 3.0 to 5.0, above 5.0)

Fig. 2. Hierarchy for each dimension in *BC_weather*.

dimension can be created by users or experts or generated automatically by data clustering or data analysis. A dimension in a spatial data cube is usually organized using tree hierarchy. However, our approach can be applied to lattice hierarchy too. The way how nonspatial data is generalized is transparent to the methods we use. Fig. 2 presents the hierarchies for dimensions in *BC_weather* warehouse.

Of the three measures, *region_map* is a *spatial* measure which represents a collection of spatial pointers to the corresponding regions, *area* is a *numerical* measure which represents the sum of the total areas of the corresponding spatial objects, and *count* is a *numerical* measure which represents the total number of base regions (probes) accumulated in the corresponding cell.

Table 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 and measures, OLAP operations can be performed by stepping up and down along any dimension presented in Fig. 1.

Let us examine some popular OLAP operations and analyze how they are performed in 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. This can be realized by transforming the selection criteria into a query against the spatial data warehouse and be processed by query processing methods [10].
2. *Pivoting*, which presents the measures in different cross-tabular layouts. This can be implemented in a similar way as in nonspatial data cubes.
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 nonspatial measures, aggregation is implemented in the same way as in nonspatial data cubes [1], [8], [24]. 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, presenting it, or performing appropriate generalization from it when necessary.

From this analysis, one can see that a major performance challenge for implementing spatial OLAP is the efficient construction of spatial data cubes and implementation of roll-up/drill-down operations. This can be illustrated by analysis of how OLAP is performed in the data cube of Example 1.

Example 5. The roll-up of the data cube of BC weather probe of Example 1 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 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.

TABLE 1
Table for Weather Probes

Region_name	Time	Temperature	Precipitation
AA00	01/01/98	-4	1.5
AA01	01/01/98	-7	1.0
...	...	...	...
AA00	01/02/98	-6	2.5
AA01	01/02/98	-8	1.0
...	...	...	...

TABLE 2
Result of a Roll-Up Operation

Time	Temperature	Precipitation	Region-map	Area	Count
March	cold	0.1 to 0.3	{AL04,AM03, ... XN87}	150,000	12
March	cold	0.3 to 1.0	{AM10, AN05,... YP90}	180,000	14
...	...	...	...	...	...

TABLE 3
Result of Another Roll-Up Operation

Time	Temperature	Precipitation	Region-map	Area	Count
January	below -20	dry	{AK04,AK07, ... VS67}	200,000	21
January	below -20	fair	{AG10, AG05,... TP90}	230,000	20
...	...	...	...	...	...

By doing so, the generalized data cube contains three dimensions, *Time (in months)*, *Temperature (in monthly average)*, and *Precipitation (in monthly average)*, two numerical measures (*area* and *count*) and one spatial measure *region-map* which is a collection of spatial object_ids, as shown in Table 2. Moreover, roll-up and drill-down can be performed dynamically, which may produce another table, as shown in Table 3.

Two different roll-ups from the BC weather map data (Fig. 1) produce two different generalized region maps, as shown in Fig. 3, each being the result of merging a large number of small (probe) regions shown in Fig. 1. Computing such spatial merges of a large number of regions flexibly and dynamically poses a major challenge to the implementation of spatial OLAP operations. Only if appropriate precomputation is performed, can the response time be satisfactory to users.

Similar to the structure of a nonspatial data cube [2], [24], a spatial data cube consists of a lattice of cuboids, where the lowest one (*base cuboid*) references all the dimensions at the primitive abstraction level (i.e., group-by all the dimensions), and the highest one (*apex cuboid*, which contains only one cell) summarizes all the dimensions at the top-most abstraction level (i.e., no group-by's in aggregation).

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. The aggregation (such as sum, average, etc.) of numeric values results in a new numeric value. However, the clustering of spatial object pointers may not lead to a single, new spatial object. If all the objects pointed to by the spatial pointers are connected, they can be merged into one large region; otherwise, they will be represented by a set of isolated regions. A numeric value usually takes only two to eight bytes of storage space and a small amount of time in computation. However, a spatial object 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 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 in the database as *materialized views* (*computed cuboids*) to facilitate fast 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 (*non-spatial*) data warehouse industry: some materialize every cuboid, some none, and some only a part of the cube (i.e., some of the cuboids). There are interesting studies on the efficient computation of data cubes [1], [12], [24]. A previous study [12] shows that materializing every view (cuboid) requires a huge amount of disk space, whereas not materializing any view requires a great deal of on-the-fly, and often redundant, computation. A greedy algorithm, which we refer to as *HRU Greedy algorithm*, has been proposed in [12] to determine which cuboids should be precomputed for partial materialization of data cubes. In the implementation of spatial data cubes, we face a dilemma for balancing the cost of online computation and the storage overhead of storing computed spatial measures: The substantial computation cost for on-the-fly computation of spatial aggregations calls for precomputation but substantial overhead for storing aggregated spatial values discourages it. Obviously, one should not materialize every cuboid with limited storage space but one cannot afford to compute all the spatial aggregates on-the-fly.

This 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

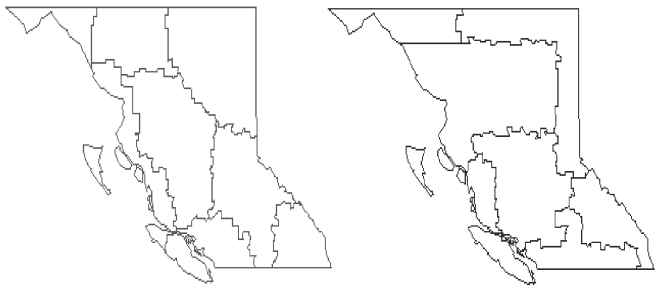

Fig. 3. Roll-up operation along different dimensions.

completely materialize a cuboid or not at all. However, for materialization of 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 preaggregated. We examine the technique in detail in the next section.

3 Methods for Computing Spatial Measures in Spatial Data Cube Construction

In this discussion, we assume that the computation of spatial measures involves *spatial region merge* operation only. The principles discussed in this paper, however, are also applicable to other kinds of spatial operations, such as *spatial map overlay*, *spatial join* [9], [10], and *intersection* between lines and regions.

3.1 Choices for Computation of Spatial Measures

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

1. Collect and store the corresponding spatial object pointers but do not perform precomputation of spatial measures in a spatial data cube. This can be implemented by storing, in the corresponding cube cell, a pointer to a collection of spatial object pointers. This choice indicates that the (region) merge of a group of spatial objects, when necessary, may have to be performed on-the-fly. It is still a good choice if only spatial display is required (i.e., no real spatial merge has to be performed), or if there are not so many regions to be merged in any pointer collection (thus, online merging is not very costly). If the OLAP results is just for viewing, display-only mode could be useful. However, OLAP results can be used for further spatial analysis and spatial data mining, such as association, classification, etc. [11]. It is thus important to merge a number of spatially connected regions for such analysis.
2. Precompute and store some rough approximation/estimation of the spatial measures in a spatial data cube. 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 result. For example, the minimum bounding rectangle (MBR) of the spatial merge result (representable by two points) can be taken as a rough estimate of a merged region. Such a precomputed result is as small as a nonspatial measure and can be presented quickly to users. If higher precision is needed for specific cells, the application can either fetch precomputed high quality results, if available, or compute them on-the-fly.
3. Selectively precompute some spatial measures in a spatial data cube. This seems to be a smart choice. However, the question is how to select a set of spatial measures for precomputation. The selection can be performed at the cuboid level, i.e., either precompute and store *each* set of mergeable spatial regions for *each* cell of a selected cuboid, or precompute none if the cuboid is not selected. Since a cuboid usually consists of a large number of spatial objects, it may involve precomputation and storage of a large number of mergeable spatial objects but 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.

Based on the above analysis, one can see that there are two promising directions to solve the problem: 1) explore efficient polygon amalgamation methods for region merge, which has been studied recently in [25], or 2) explore selective precomputation of the mergeable spatial objects in spatial data cubes. Our subsequent discussion is focused on how to select a group of mergeable spatial objects for precomputation from a set of targeted spatial data cuboids.

3.2 Methods for Selection of Mergeable Spatial Objects for Materialization

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 the shortest time to evaluate results of spatial OLAP queries. The groups can be organized in a partial order, e.g., 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 $G_i \subseteq G_j$. Similar to [12], 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 identifiers for map regions form a set M.
- A lattice L where each node of the lattice contains a set S such that,

$$S = \{s \mid s \subset M\}$$
$$(\forall s_i, s_j \in S)(i \neq j \Rightarrow s_i \cap s_j = \emptyset).$$

  The lattice L corresponds to the lattice of cuboids chosen by HRU Greedy 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, $T = \{G \mid G \subset M\}$, of selected mergeable groups, described as follows. Groups in T provide minimum online computation time to merge an average node of the lattice.

Benefit. 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:

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

The problem of finding the set of mergeable groups which takes the minimum computation time to compute the merges of all the groups is intractable because it can be reduced to a set-covering problem. Thus, we propose several heuristic strategies in the remaining part of this paper.

We now 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 ith 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 five levels, the total number of cuboids that can be generated will be $5^{10} - 1 \approx 9.8 \times 10^6$. 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 [12].

A greedy cuboid-selection algorithm has been presented in [12], 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 online computation of a particular spatial cuboid, should be considered in the cost estimation since the spatial computation, such as region merging, map overlaying, and spatial join, could be expensive when involving a large number of spatial objects. Even when we decide to materialize a cuboid, it is still unrealistic to compute and store every spatial measure for each 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 any detail or may only be examined a small number of times. In our subsequent analysis, we assume that a set of cuboids have been selected for materialization using an extended cuboid-selection algorithm similar to HRU Greedy [12] and examine how to determine which sets of mergeable spatial objects should be precomputed.

In the following sections, we introduce the *spatial greedy* algorithm, *pointer intersection* algorithm, and *object connection* algorithm, that selectively materialize spatial measures. For more details on the algorithms, we refer to [23]. The algorithms select cells that will be materialized, i.e., merged during spatial data cube construction time. Therefore, in the remainder of this paper we refer to this process as a *premerge*.

3.2.1 Spatial Greedy Algorithm

As we have already explained, although the HRU Greedy algorithm [12] performs well in the creation of a conventional (nonspatial) data cube, it cannot be applied for handling spatial measures. We propose a new algorithm that materializes only selected cells in the cuboids chosen by HRU Greedy algorithm. It is important to observe that each cuboid contains a number of groups of spatial objects. Furthermore, each group is comprised of *connected* objects that have same nonspatial descriptions. As explained earlier, our goal is to premerge groups that provide the largest benefit for online processing. We identify two sources of benefit in premerging a group. First, premerge of a group eliminates merge on-the-fly; second, it reduces merging cost for all groups that include that group. We refer to these two benefits, as *direct* and *indirect* benefit, respectively. In each iteration of our greedy algorithm, the benefits for all groups are compared and the one with the highest benefit is chosen. The following heuristics are used for selection of groups of connected regions to be premerged.

- *Access frequency*. Based on the access history or estimated access frequency of a set of cuboids, one can calculate the benefit of the merge. If a group of connected regions is more frequently accessed than other groups, it is beneficial to premerge (and save) this group of connected regions.
- *Cardinality of a group of connected regions, i.e., the number of regions in a group*. If a candidate group has more connected regions than other groups, it is beneficial to select this candidate in the premerge (having fewer disk accesses during online processing). Notice that if a merge is performed on a group of connected regions at a descendant node (cuboid), the subsequent cost analysis on its ancestor nodes should count the newly merged region as one region only.
- *Sharing among the cuboids in the cube lattice structure*. If a candidate is shared among more cuboids in the lattice structure, it is 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 cuboids 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 online computation costs. Ascending the lattice of cuboids leads to more general descriptions of data in the database. Subsequently, if some objects have the same nonspatial descriptions in one cuboid, they must have the same descriptions in all ancestors of that cuboid. We now introduce a term *nonoccluded ancestor* used for computation of indirect benefit.

Definition 3.1. *Let F and G be groups containing pointers to spatial objects such that $G \subset F$. Then, group F is a nonoccluded 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 $cardinality(J) > cardinality(G)$.*

Example 6 illustrates all three conditions stated in the definition.

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

$$direct\_benefit(G) = access\_frequency(G) \times cardinality(G) \tag{3.1}$$

$$indirect\_benefit(G) = \sum_{G \prec A} access\_frequency(A) \times (cardinality(G) - 1) \tag{3.2}$$

$$total\_benefit(G) = direct\_benefit(G) + indirect\_benefit(G). \tag{3.3}$$

Equation (3.1) 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 online 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 cuboids 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 an ancestor group contains k connected regions, premerging its subgroup G that contains n regions ($n < k$) decreases the cardinality of the ancestor group to $k - (n - 1)$. Equation 3.2 shows that only nonoccluded ancestors of a group G contribute to its indirect benefit. The following example illustrates computation of the indirect benefit.

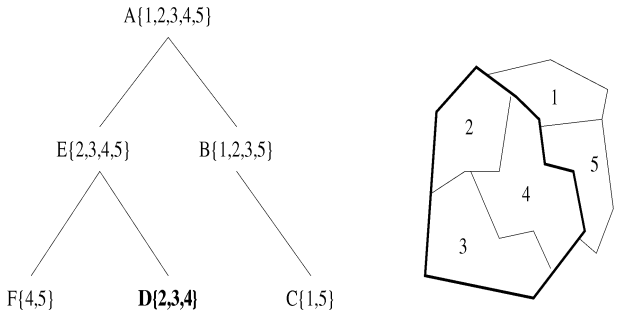

Fig. 4. The partial order of groups in Example 6.

Example 6. 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. The partial order of the groups and the corresponding map are shown in Fig. 4. 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 groups (A and E) contribute to indirect benefit of D.
2. Only group E has already been materialized. Both A and E are occluded ancestors of D (E is materialized and $D \subset E \subset A$). Thus, there is no group that contributes to indirect benefit of D.
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. 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. Thus, only group E contributes to the indirect benefit of D.

After the mergeable candidate groups are detected, the greedy algorithm proceeds as follows. In the first iteration, the algorithm computes the total benefits for all candidate groups, compares their benefits, and selects the one with the highest benefit. In subsequent iterations, the benefit estimation may change for some groups since these groups may contain the subgroups of the merged groups in the previous iteration(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

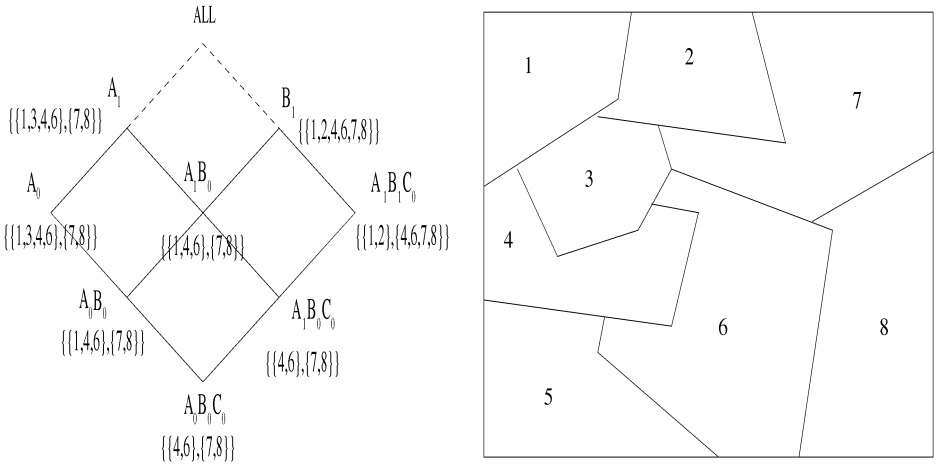

Fig. 5. Lattice for selected cuboids and the corresponding map.

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 3.1 (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 Greedy [12].
- An access frequency table which shows the access frequency of each cuboid 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. $select\_candidate\ (remaining\_set, merged\_obj\_table)$;
6. UNTIL $cardinality(merged\_obj\_table) \geq max\_num\_group$

Rationale of the algorithm. The algorithm works on the cuboids selected by the cuboid-selection algorithm (HRU Greedy) [12]. 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. In 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 HRU Greedy algorithm in [12], the global optimality is an NP-hard problem. Therefore, based on the similar reasoning to that in [12], 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. In comparison with HRU Greedy algorithm, the selection handled in our algorithm is at a deeper level and examines every precomputed element in a cuboid to be materialized. This refined computation is more costly than examining only at a (lattice) cuboid 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 online spatial computation or save substantial storage space, and is thus worth doing. In addition to this, HRU Greedy algorithm does not take cuboid access frequency into consideration, whereas ours considers it seriously.

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

Example 7. Fig. 5 shows a (cube) lattice, which represents a relationship among a set of cuboids selected by HRU Greedy algorithm. Region map is presented as well. The access frequency of each cuboid is shown in Table 4. The data cube consists of three dimensions: A, B, and C. We use indices to represent different abstraction levels for a dimension, e.g., $A_0B_0C_0$ represents the base cuboid. Notice that some groups (e.g., $\{4, 6\}$ and $\{7, 8\}$) may be disjoint in low-level cuboids, but merged into larger groups in high-level cuboids of the lattice. This is explained as follows:

TABLE 4
Access Frequency of the Cuboids

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

Only groups that belong to same tuples may be merged because they describe the object with the same values for nonspatial dimensions. Rolling up along dimensions in the data cube increases the likelihood of having identical generalized values.

The access frequency of every candidate group (i.e., a group of connected regions) is equal to the access frequency of the corresponding cuboid where it resides. If it resides in more than one cuboid, its access frequency is the sum of the access frequency of all the cuboids where it resides. For example, group $\{4,6\}$ appears in two cuboids, $A_0B_0C_0$ and $A_1B_0C_0$. Thus, the accumulated access frequency of the group $\{4,6\}$ is $100+80=180$.

The process of benefit computation and premerge region selection is depicted in Table 5. In 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

$$\text{total benefit} = \text{direct benefit} + \text{indirect benefit} = 200.$$

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 2,070) 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 iteration, the cardinality of $\{1,2,4,6,7,8\}$ reduces from 6 to 4. Therefore,

$$\text{direct benefit} = frequency(\{1,2,4,6,7,8\}) \times cardinality(\{\{1,4,6\},2,7,8\}) = 60 \times 4 = 240.$$

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 nonoccluded 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 unchanged.

After execution of the third iteration, 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.

3.2.2 Pointer Intersection Algorithm

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

TABLE 5
First Three Iterations of Spatial Greedy Algorithm

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			

- 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 nonoccluded ancestors.

Thus, we propose the *pointer intersection* algorithm that favors spatial pointer groups that appear in many cuboids and with high access frequency. In our subsequent analysis, we again assume that a set of cuboids have been selected for materialization using an extended cuboid-selection algorithm similar to HRU Greedy 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 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).

Algorithm 3.2 (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 HRU Greedy [12]. 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 a cuboid in the lattice.
- A region map which illustrates 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 minimum accumulated access frequency of a group of connected regions. The candidate group which passes the threshold will be considered for precomputation.
- *n_cuboids*. The number of cuboids selected for materialization.

Output. A set of candidate groups selected for spatial precomputation.

Method.

- The main program is outlined as follows:

1. FOR $cuboid\_i := 1$ TO $n\_cuboids$ DO
2. FOR $cuboid\_j := cuboid\_i$ TO $n\_cuboids$ 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. $shared\_spatial\_merging$ $(connected\_obj\_table, merged\_obj\_table)$;

Rationale of the algorithm. The algorithm is to find those spatial object groups in the data cube which are frequently accessed and mergeable, and then perform spatial merge for them in precomputation. The algorithm works on the candidate cuboids that are selected based on an extension to a well-known HRU Greedy cuboid selection algorithm [12]. Lines 1 to 4 ensure that every pair of cuboids is examined for each candidate cube cell which derives the maximal intersections of spatial object pointers and stores them into candidate table. Line 5 removes from the candidate table those candidates whose accumulated access frequency is less than min_freq. Line 6 finds the spatially connected subsets from each candidates object pointer set and line 7 materializes them and puts them into the $merged\_obj\_table$. Optimization has been explored in each procedure/function.

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 the self-intersection not applied, these groups would be skipped. The performance analysis, conducted in Section 4, will show that the self-intersection has a significant positive impact on the effectiveness of the algorithm (see Fig. 9).

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

Example 8. Assume that a set of cuboids have been selected from lattice, as shown on Fig. 6, using an extended cuboid selection algorithm like HRU Greedy [12]. 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. To focus our discussion on the selective materialization of spatial measures, only the groups of spatial pointers are shown in Table 6 without presenting the corresponding dimension values. Let access frequencies for *cuboid_1*, *cuboid_2*, and *cuboid_3* be 30, 15, and 20, respectively and min_freq be 40. The map is shown in Fig. 7.

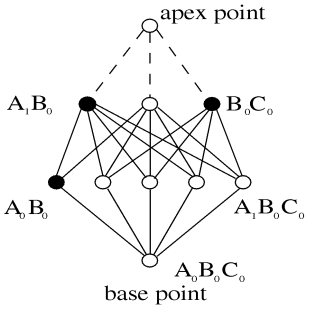

Fig. 6. A lattice showing selected cuboids.

After applying steps depicted on Lines 1 to 4 of the algorithm, we get a set of candidates in Table 7. Raw access frequency of a candidate is a sum of the 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 group contribute to the accumulated access frequency of that group. The accumulated access frequency of $\{1,4\}$ is 15 (access frequency of cuboid 2). 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 ($\leftarrow$). Since the *min_freq* threshold is larger than the threshold for any of the cuboids, none of the candidates resulted only from a self-intersection can be merged. Thus, we do not show self-intersections in this example.

After applying *spatial_connectivity_testing* procedure, we detect that the regions: $\{1,4,7\}$, $\{1,4\}$, $\{2,3\}$, and $\{6,9\}$ should be merged. At first 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 groups have high access frequency, it is beneficial to store them 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 $\{1,4\}$, whereas $\{1,2,4,7\}$ may use $\{1,4,7\}$.

3.2.3 Object Connection Algorithm

In this section, we present the *object connection* algorithm, which differs slightly from the *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.

Algorithm 3.3 (Object Connection Algorithm). An 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 3.2.

Output. The same as Algorithm 3.2.

Method.

- The main program is outlined as follows:

1. FOR $cuboid\_i := 1$ TO $n\_cuboids$ 1 DO
2. FOR $cuboid\_j := cuboid\_i$ TO $n\_cuboids$ 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. $shared\_spatial\_merging$ $(candidate\_connected\_obj\_table, merged\_obj\_table)$;

Rationale of the algorithm. The major difference of this algorithm from the former one, Algorithm 3.2, is at the handling of connected objects. The former delays the connectivity checking after min_freq threshold filtering (Line 6), whereas the new one does it at the insertion into the candidate table (Line 4). 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

TABLE 6
Sets of Pointers for 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}

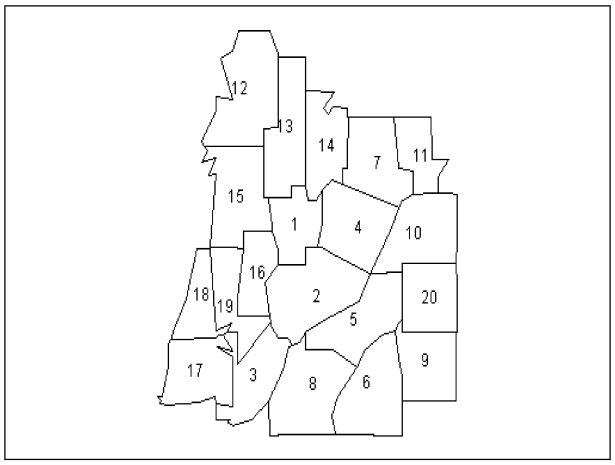

Fig. 7. An example map.

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 the *min_freq* threshold. On the other hand, if we apply the *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. A formal explanation in the form of a theorem can be found in [23].

For the same example as for the *pointer intersection* algorithm, the execution *object connection* algorithm is presented as follows:

Example 9. The execution of Lines 1 to 4 of the algorithm leads to a set of candidate connected object pointer groups, as shown in Table 8. Similarly to Example 8, the access frequencies are computed and frequency filtering applied. Candidates that pass the frequency threshold are marked with an arrow ($\leftarrow$). The marked groups are the regions that are selected for premerge.

Since Examples 8 and 9 contain only a small number of objects, the two algorithms produce identical results. Our performance analysis, presented later in the paper, shows that the case when the object connection algorithm selects more groups then the pointer intersection algorithm occurs very seldom and that it has a very small impact on the benefits of selective materialization.

4 Performance Analysis of Proposed Algorithms

In the previous section, 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 Greedy cuboid-selection algorithm [12].

TABLE 7
Candidate_Table for Selected Cuboids

Intersected_portion	Raw access frequency	Accumulated access frequency	Total access frequency
{1, 4, 7, 20}	45	0	45 $\leftarrow$
{2, 3, 6, 9}	45	0	45 $\leftarrow$
{10, 14, 19 }	45	0	45 $\leftarrow$
{10, 19}	65	0	65 $\leftarrow$
{1, 4}	50	15	65 $\leftarrow$
{7, 20}	50	15	65 $\leftarrow$
{2, 3}	65	0	65 $\leftarrow$
{6, 9}	50	15	65 $\leftarrow$
{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 8
Candidate_Connected_Obj Table for Selected Cuboids

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

- *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.
- *pct_of_groups* to materialize (for *spatial greedy* algorithm).
- *min_freq_ratio* as a ratio between minimal frequency threshold and average access frequency of all cuboids (for *pointer intersection/object connection* algorithm).

We now list the major steps of our simulation.

1. Generating a map.
2. Setting up a spatial data warehouse.
3. Selective materialization of spatial measures.
4. Simulating typical OLAP operation on the spatial data warehouse.

The performance analysis study is divided into two parts. First, we analyze *effectiveness* of the algorithms. Since the goal of selective materialization is to enhance online processing, we study the usability of materialized groups. We are interested in knowing 1) how much online processing time is decreased by performing offline 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 online 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.

4.1 Effectiveness of the Algorithms

In this section, 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 two terms used throughout the subsequent analysis.

- *Saving in the number of disk accesses*. The goal of materialization of spatial measures is getting a short response time for OLAP operations. If n spatial objects are to be merged during online 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 due to offline 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 it has a stopping criterion different from that of the other two algorithms. Fig. 8 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 online processing. The slope for the benefit curve in Fig. 8 decreases when the number of materialized groups increases. Our explanation to this is that the initial premerge provides a large boost due to that many of the merged regions are shared among different cuboids. When a certain proportion of all mergeable groups is materialized, the candidates with such nice features have been almost used up and the benefit becomes marginal.

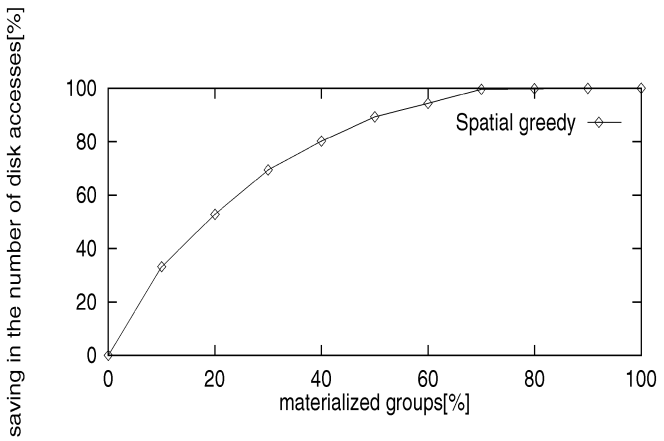

Fig. 8. Spatial greedy algorithm: Benefits of materialization.

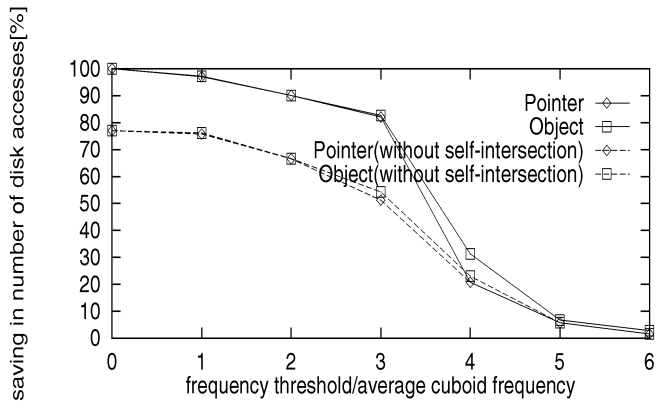

Fig. 9. Pointer intersection and object connection algorithms: Benefits of materialization.

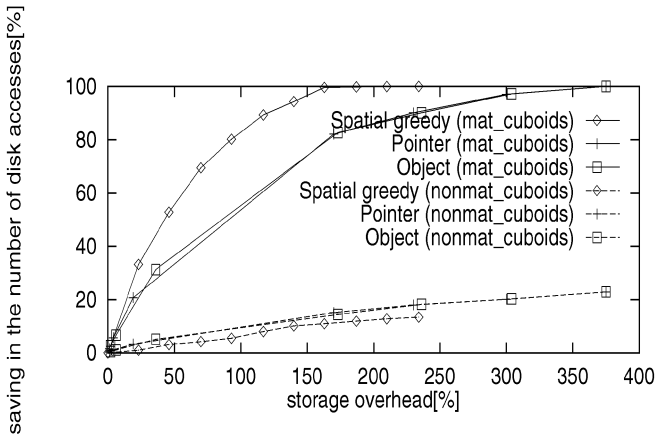

Fig. 10. Comparisons of algorithms: Benefits of materialization.

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 a 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 Fig. 9. 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 percent.

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 percent (reach 100 percent when no frequency threshold is applied). The slight difference between effectiveness of *pointer intersection* and *object connection* algorithms follows from the reasoning from the previous section (we refer to [23] for detailed explanation). 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 percent of disk accesses would be avoided during online 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 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 Greedy 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 online processing, and checked how much the premerged objects can improve their response time. The number of such queries was as high as 100,000.

Fig. 10 reveals 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 nonmaterialized 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 Greedy 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

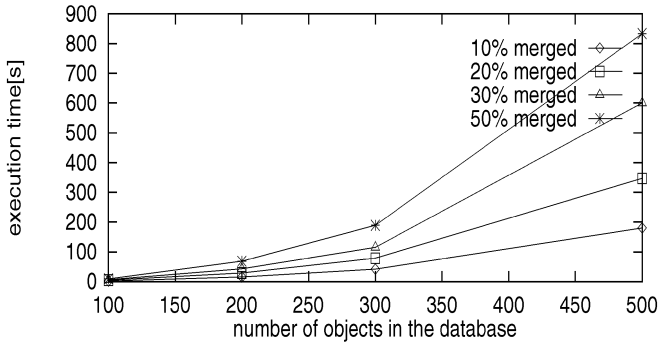

Fig. 11. Scalability of spatial greedy algorithm as a function of the number of map objects.

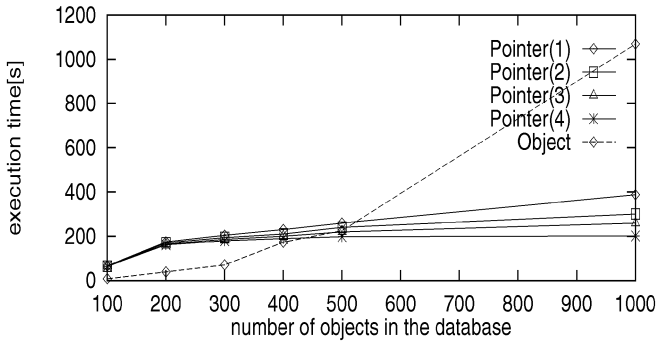

Fig. 12. Scalability of pointer intersection and object connection algorithms as a function of the number of objects.

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 Fig. 10. However, there is an issue of accessing premerged objects that may not be overlooked when considering nonmaterialized 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 nonmaterialized queries requires a search for the best candidates among premerged objects. Note that cuboids for such queries are not created offline. Another observation from Fig. 10 is that curves for *pointer intersection* and *object connection* algorithms intersect. This simply shows that additional objects premerged by the latter algorithm are not always beneficial for online 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 the 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.

4.2 Efficiency of the Algorithms

The fact that *spatial greedy* algorithm has a 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.

Fig. 11 shows the execution time as a function of the number of objects in the database. In this experiment, we fix the number of cuboids to 10. One 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 between *number_of_objects/max_number_of_tuples* and *number_of_objects/min_number_of_tuples* objects for every tuple, where the latter value can be quite large when the most generalized cuboid is close to the apex node in the cube lattice. Thus, in the first step of the algorithm, detection of mergeable groups is very expensive and it creates a large number of groups. Having a large 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 each iteration of the greedy algorithm. Not surprisingly, execution time has linear growth with the increase of percentage of groups (*pct_of_groups*) to be materialized.

We now focus on the remaining two algorithms. Fig. 12 shows the performance comparison of the algorithms when they are applied on maps with a different number of objects. We vary *min_freq_ratio* from 1 to 4, while the number of cuboids is 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 3.2 and 3.3). 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 the *pointer intersection* algorithm performs the above step for all intersections, there is a large time overhead. In the case of the *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 the *object connection* algorithm. For the above reasons, the *object*

connection algorithm outperforms the *pointer intersection* algorithm when the map contains a 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 the maximal number of intersections, and $tuples(k)$ is the number of tuples in cuboid k.

Thus, the running time for the *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 the 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 the *object connection* algorithm, that contains *max_frequency* function, becomes more expensive than the very same step in the *pointer intersection* algorithm (dealing with more groups).

To conclude, the *object connection* algorithm is very sensitive to the increase of the number of objects on a map.

Another observation from the curves in Fig. 12 is that the frequency threshold is irrelevant to the execution time of the *object connection* algorithm. This was expected since frequency filtering is the last step in the algorithm. On the contrary, the *pointer intersection* algorithm shows slightly better performance when the frequency threshold increases.

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: a large number of objects and a relatively small number of frequent queries. According to the conducted experiments, we believe that the *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, the *spatial greedy* algorithm can be used.
- Last, if a spatial data warehouse is to contain few objects, a data warehouse designer could choose to apply the *object connection* algorithm for selective materialization of spatial measures.

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

5 Discussion

We have proposed a *spatial data cube*-based data warehouse model to facilitate efficient OLAP operations on spatial data. A spatial data cube consists of both spatial and nonspatial dimensions and measures. Since it is challenging to compute spatial measures in such a data cube, our study has been focused on methods for selective materialization of spatial measures.

Previous studies [12], [21] show that materialization of all cuboids, even for a moderate number of dimensions, leads to enormous storage requirements. On the other hand, not performing materialization at all would cause severe performance penalties because all computation would have to be performed on-the-fly. Thus, selective materialization has been a good choice in data cube construction. Moreover, object-based selective materialization is essential for efficient computation of spatial data cubes. Materialization at a finer granularity than that of the cuboid-based selective materialization enables us to materialize only those portions of the cuboid(s) that will be used frequently.

In the last two sections, methods are proposed and studied for selective materialization based on the relative access frequency of the sets of *mergeable spatial regions*, that is, the sets of mergeable spatial regions should be precomputed if they are expected to be accessed/used frequently. However, the methodology of object-based selective materialization should not be confined to spatial region merge. Since many OLAP applications may encounter complex types of data, it is expected that this methodology is applicable to computing data cubes with complex types of measures. We present some examples in the next section.

5.1 General Applicability of Object-Based Selective Materialization

Based on our previous discussion, a spatial data cube can be viewed as an extension of a relational data cube, with the data types of dimensions and measures extended to include complex spatial data types, such as spatial regions. Similarly, one can expect that other extended data cube structures with measures containing complex data types may need object-based selective materialization as well. For example, a spatial data cube whose measures contain the overlays of multiple thematic maps, a multimedia data cube whose measures contain images or videos, and a time-series data cube whose measures contain complex time sequences, may all need the techniques developed here. We examine two such examples.

Example 10 (Overlay of multiple thematic maps). Many spatial databases contain multiple thematic maps. Spatial data analysis may involve the study of only one thematic map or the study of the overlay of multiple thematic maps. For example, a spatial database may contain several thematic maps: one about the highways and roads of a region, the second about the sewage network, and the third about the altitude of the region. To choose an area for a housing development, one needs to consider many factors, such as the road network

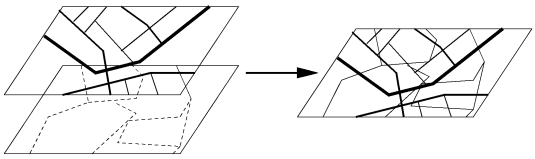

Fig. 13. Overlay of maps containing objects of different spatial data types.

connection, the sewage network connection, altitude, etc. This will require a system to support interactive drill-down and roll-up along multiple dimensions in a spatial data warehouse, which is essentially the study of multiple-dimensional overlays of multiple thematic maps of different spatial data types, such as regions, lines, and networks.

Let us walk through the process of interactive spatial OLAP based on the above scenario (see Fig. 13).

1. Determining dimensions in the spatial data cube. Usually, there is one nonspatial dimension associated with each thematic map, such as roads, altitude, etc. Creation of concept hierarchy for each dimension can be done in a more or less straightforward way, similar to that for nonspatial data cubes. Each level of abstraction for each dimension can be presented by one thematic map. Ascending the concept hierarchy leads to a more general thematic map.
2. Determining measures in the spatial data cube. A spatial measure can be a collection of spatial objects with the same nonspatial descriptions at a given combination of the levels of abstraction. On the other hand, numeric measures may include numeric aggregations, such as *income per capita* for people living in these areas.
3. Spatial OLAP.
   a. An initial OLAP request: A user specifies an OLAP request by choosing dimensions of interest as well as the level of details for each of the dimensions (e.g., "*Show me regions that are 5-10Km from a major highway and above 500 meters in altitude*").
   b. Spatial overlay: Multiple thematic maps, including maps containing different spatial data types, are overlaid in this step. It is well known that spatial overlay is a computationally expensive operation.
   c. Interactive drill-down/roll-up along any of the dimensions: A user is seldom satisfied with initial OLAP results. Thus, he/she often navigates through a lattice of cuboids by performing roll-up/drill-down operations. Suppose that the user's request changes to "*Show me the regions in further detail, which are close to a dense city street network and are flat in altitude.*" This may involve the drill-down from the major highway network map to more a detailed city-street network, and from a rough altitude network map (above 500 meters in altitude) to a more refined altitude map. Notice the answer to the new query (drilling) requires the computation of the overlay of the maps on the new scales.

   The last two steps may be repeated a number of times, i.e., until a user gets a satisfactory response.

The above process indicates that the performance bottleneck is the overlay of the thematic maps at multiple- dimensional spaces. One solution is to perform this computation on-the-fly upon every OLAP request, but this would substantially slow down the query response time. To achieve interactive online analytical processing, selective materialization should be explored to speed up the processing. Moreover, selective materialization on cell level rather than on cuboid level is more promising for the following reasons:

- Some portions of the map are accessed more frequently than the others. Land close to a city and highways could be more popular than the land far away from the city and highways. Thus, one may expect to materialize some portions of the cuboid for fast spatial OLAP response.
- Some ranges of nonspatial dimension values are accessed more frequently than the others. For example, it is more likely to look for housing construction land within the range between 200-1,000 meters in altitude than above 2,000 meters.

Therefore, object-based selective materialization is useful for multiple-dimensional analysis in this case. Similarly, the algorithms proposed in the previous sections can be modified accordingly to handle the case of spatial overlays.

The methodology of the object-based selective materialization is applicable to other kinds of applications involving complex types of data. Let us examine an example of time-series data analysis.

Example 11 (Online analytical processing of time-series data). Time-series data has been popularly used in market analysis, trend and deviation analysis, predictive modeling, etc. Take the stock analysis as an example. Many people may like to capture stock price fluctuations during the course of a day, a week, or a month. Suppose that we are given detailed stock curves where data is gathered hourly. Different customers may want to look at stock fluctuation curves at different levels of granularity in terms of time dimension, such as daily, weekly, or monthly.

It is interesting to construct a multidimensional, time-series data cube to facilitate online analytical processing and data mining on time-series data. However, it is expensive to store stock fluctuation curves as a measure since it may take kilo- to mega-bytes of storage space to store each stock fluctuation curve. When rolling up along time dimension (such as examining monthly or quarterly stock fluctuation), it is natural to ignore some details, such as hourly fluctuations, and the stock fluctuation

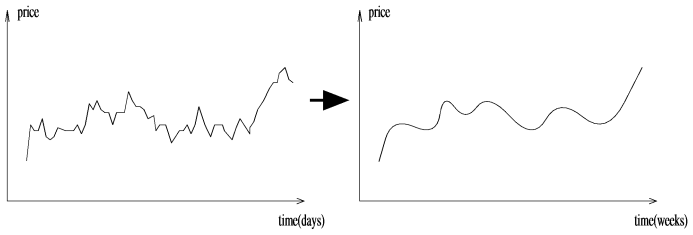

Fig. 14. OLAP of time-series data.

curves will need to be smoothed to ignore the details, as shown in Fig. 14. Computing a new curve involves FFT, wavelet analysis, or some other methods. These are reasonably expensive operations, and it is beneficial that some computation be done in the preprocessing stage, during the data warehouse creation time. Quite often, people are interested in the stocks of the major companies (e.g., IBM, Microsoft), including more detailed information. However, it could be satisfactory to most users to store only monthly or quarterly results for small, noninfluential stocks. Moreover, a certain time of year (e.g., quarter ends) could be more important for data analysts than others.

These popular user requests indicate that expensive precomputation should be performed selectively not at the cuboid level but at the object-level.

These examples, and other similar considerations for online analytical processing of other kinds of nonrelational data (such as multimedia, CAD, engineering data, etc.), lead to an important conclusion: object-based selective materialization is a necessary technique for construction of data cubes with measures of nontraditional data types.

5.2 Further Enhancements of the Method

In this section, we discuss some concerns related to the proposed method and some possible alternatives.

Our method assumes the existence of information about the access frequencies of a set of selected cuboids. What if there exists no such information initially? There are several methods to handle this. One solution is to assign an initial access frequency only to a *level* in the lattice (less work than assigning it to every *cuboid*), based on some data semantics or simply assuming that medium levels (such as the *county* level in a province map) are accessed most frequently, and lower levels (such as the *probe-station* level) are accessed less frequently. Such a frequency estimate can be adjusted based on later accessing history. Alternatively, one may choose to materialize every mergeable group accessed initially, record access frequencies, and toss away the rarely used groups when the storage requirements reach a certain threshold. In our experimental study, we assumed the uniform access frequency for all the cells of each cuboid. This may not be true in real cases. For example, *Southern_BC* is often accessed more frequently than *Northern_BC* due to that the population and business are clustered in the former. In these cases, the granularity of the access frequency assignment can be tuned to a subset of a cuboid with little change to the method itself.

Our method selects groups of regions for precomputation based on the access frequency but not on the concrete size of the mergeable regions. This is reasonable if all the base regions are of comparable size and complexity. However, in some databases, some regions could be substantially larger or more complicated and thus take substantially larger space and more computation than others. Taking this factor into consideration, the judgment criteria can be modified accordingly, for example, adding the cost of region merging and a compression ratio which is the difference in size between the precomputed vs. not precomputed regions.

In our analysis, we have adopted the philosophy and algorithm of selective materialization of data cubes as proposed in [12]. Is it possible to use our analysis to the approaches which attempt to materialize every cuboid in a data cube [1]? The answer is positive because even if it is reasonable to assume the materialization of every cuboid for nonspatial measures, it is rarely possible to materialize every cell of the cuboids for spatial measures since each cell contains groups of large spatial objects. Therefore, selective materialization of mergeable spatial objects for OLAP is essential for implementation of spatial OLAP operations.

In our algorithm, the mergeable regions 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 could be considered as one region.

Selective materialization is a trade-off between online OLAP computation time and the overall storage space. However, even if we always make a good selection, there are still groups whose spatial aggregates are not precomputed. Spatial merge, join, or overlay are expensive operations and their online computation may slow down the system substantially. A possible compromise is to adopt a multiresolution approach which distinguishes high and low resolution computations. For low resolution, it first quickly returns the precomputed rough estimation such as the minimum bounding rectangles (MBRs), whereas for high resolution, a selective materialization method can be employed for a balance of relatively fast response and reasonable precomputation and storage overhead. By doing so, a user will get a fast response on browsing-like OLAP operations and may selectively examine his/her interested regions with a finer resolution at a higher cost.

Important themes, which have not been addressed in this discussion, are the efficient storage, indexing, and

incremental update of precomputed spatial data cube elements under the policy of object-based selective materialization of OLAP data.

6 Conclusions

We have studied the construction of spatial data warehouses based on a spatial data cube model which consists of both spatial and nonspatial dimensions and measures. For OLAP on spatial data, we studied issues on efficient computation of spatial measures and proposed methods for object-based selective materialization of spatial OLAP computation results. We showed that object-based selective materialization is necessary for data warehouses that contain complex, nonrelational data.

Three algorithms, *spatial greedy*, *pointer intersection*, and *object connection*, have been developed for partial materialization of the spatial objects resulted from spatial OLAP operations. The performance of the algorithms is studied and compared with the conditions listed for choosing one over the others. Spatial OLAP operations have been integrated with spatial data mining in the GeoMiner system [11]. This integration itself shows the importance of object-based selective materialization because spatial OLAP results can be used for spatial data mining and other analyses.

With the recent success of OLAP technology, spatial OLAP holds a great promise for the fast and efficient analysis of large amounts of spatial data. Thus, spatial OLAP is expected to be a promising direction for both academia and industry in the years to come. The development of new spatial database technology for spatial OLAP, the object-based selective materialization of other spatial measures, including spatial overlay and spatial joins, the efficient storage and indexing of partially materialized spatial data cubes, and the smooth integration of spatial OLAP with spatial data mining are interesting issues for future research.

References

[1] S. Agarwal, R. Agrawal, P.M. Deshpande, A. Gupta, J.F. Naughton, R. Ramakrishnan, and S. Sarawagi, "On the Computation of Multidimensional Aggregates," *Proc. 1996 Int'l Conf. Very Large Data Bases,* pp. 506–521, Sept. 1996.

[2] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *ACM SIGMOD Record,* vol. 26, pp. 65–74, 1997.

[3] M. Egenhofer, "Spatial SQL: A Query and Presentation Language," *IEEE Trans. Knowledge and Data Eng.,* vol. 6, pp. 86–95, 1994.

[4] M. Ester, H.-P. Kriegel, and J. Sander, "Spatial Data Mining: A Database Approach," *Proc. Int'l Symp. Large Spatial Databases (SSD '97),* pp. 47–66, July 1997.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining (KDD '96),* pp. 226–231, Aug. 1996.

[6] C. Faloutsos and K.-I. Lin, "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets," *Proc. 1995 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 163–174, May 1995.

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

[8] 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,* vol. 1, pp. 29–54, 1997.

[9] O. Günther, "Efficient Computation of Spatial Joins," *Proc. Ninth Int'l Conf. Data Eng.,* pp. 50–60, Apr. 1993.

[10] R.H. Güting, "An Introduction to Spatial Database Systems," *The Very Large Data Base J.,* vol. 3, pp. 357–400, 1994.

[11] J. Han, K. Koperski, and N. Stefanovic, "GeoMiner: A System Prototype for Spatial Data Mining," *Proc. 1997 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 553–556, May 1997.

[12] V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Implementing Data Cubes Efficiently," *Proc. 1996 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 205–216, June 1996.

[13] W.H. Inmon, *Building the Data Warehouse.* John Wiley, 1996.

[14] D.A. Keim, H.-P. Kriegel, and T. Seidl, "Supporting Data Mining of Large Databases by Visual Feedback Queries," *Proc. 10th Int'l Conf. Data Eng.,* pp. 302–313, Feb. 1994.

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

[16] E. Knorr and R. Ng, "Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining," *IEEE Trans. Knowledge and Data Eng.,* vol. 8, pp. 884–897, Dec. 1996.

[17] K. Koperski and J. Han, "Discovery of Spatial Association Rules in Geographic Information Databases," *Proc. Fourth Int'l Symp. Large Spatial Databases (SSD '95),* pp. 47–66, Aug. 1995.

[18] K. Koperski, J. Han, and N. Stefanovic, "An Efficient Two-Step Method for Classification of Spatial Data," *Proc. Eighth Symp. Spatial Data Handling,* pp. 45–55, 1998.

[19] W. Lu, J. Han, and B.C. Ooi, "Knowledge Discovery in Large Spatial Databases," *Proc. Far East Workshop Geographic Information Systems,* pp. 275–289, June 1993.

[20] R. Ng and J. Han, "Efficient and Effective Clustering Method for Spatial Data Mining," *Proc. 1994 Int'l Conf. Very Large Data Bases,* pp. 144–155, Sept. 1994.

[21] K. Ross and D. Srivastava, "Fast Computation of Sparse Datacubes," *Proc. 1997 Int'l Conf. Very Large Data Bases,* pp. 116–125, Aug. 1997.

[22] A. Silberschatz, M. Stonebraker, and J. D. Ullman, "Database Research: Achievements and Opportunities into the 21st Century," *ACM SIGMOD Record,* vol. 25, pp. 52–63, Mar. 1996.

[23] N. Stefanovic, "Design and Implementation of On-Line Analytical Processing (OLAP) of Spatial Data," master's thesis, Simon Fraser Univ., Canada, Sept. 1997.

[24] Y. Zhao, P.M. Deshpande, and J.F. Naughton, "An Array-Based Algorithm for Simultaneous Multidimensional Aggregates," *Proc. 1997 ACM-SIGMOD Int'l Conf. Management of Data,* pp. 159–170, May 1997.

[25] X. Zhou, D. Truffet, and J. Han, "Efficient Polygon Amalgamation Methods for Spatial Olap and Spatial Data Mining," *Proc. Sixth Int'l Symp. Large Spatial Databases (SSD '99),* pp. 167–187, July 1999.

Nebojsa Stefanovic received the BSc degree in computer engineering from University of Belgrade, Yugoslavia in 1993, and the MSc degree in computer sciences from Simon Fraser University in 1997. His primary research interests are in spatial OLAP, spatial data mining, and data visualization. He is currently with Seagate Software in the Research and Development Department. He is a member of IEEE Computer Society.

Jiawei Han received the PhD degree in computer sciences from the University of Wisconsin at Madison in 1985. He is currently the director of Intelligent Database Systems Research Laboratory and professor in the School of Computing Science, Simon Fraser University, Canada. He has conducted research in the areas of database systems, data mining, data warehousing, geo-spatial data mining, Web mining, deductive and object-oriented database systems, and artificial intelligence, with more than 150 journal or conference publications. He is currently a project leader of the Canada Networks of Centres of Excellence (NCE)/IRIS-3 project "Building, querying, analyzing, and mining data warehouses on the Internet" (1998-2002), and the chief architect of the DBMiner system. He has served or is currently serving in the program committees for more than 50 international conferences and workshops. He has also been serving on the editorial boards of *IEEE Transactions on Knowledge and Data Engineering*, *Data Mining and Knowledge Discovery*, and *Journal of Intelligent Information Systems*. He is a member of the IEEE Computer Society, the ACM, ACM-SIGMOD, ACM-SIGKDD, and AAAI.

Krzysztof Koperski received the MSc degree in electrical engineering from Warsaw University of Technology, Poland in 1988, and the PhD degree in computer sciences from Simon Fraser University in 1999. He is currently a research scientist at the Data Analysis Products Division of MathSoft, Inc. His research interests include spatial data mining, image data mining, spatial databases, and data visualization. He is a member of the IEEE Computer Society and the ACM.