9-1-2003

Spatial Data Mining Using Branch Grafted R-tree.

Priyanka Dubey

Spatial Data Mining Using Branch Grafted R-tree

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

University of Nebraska at Omaha

by

Priyanka Dubey

Fall, 2003

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

THESIS ACCEPTANCE

Acceptance for the faculty of the Graduate College,
University of Nebraska, in partial fulfillment of the
requirements for the degree Master of Science in Computer Science,
University of Nebraska at Omaha

Committee

Zhengxi Chen

S. K. Willemsen

Peter Wolcott

Chairperson Zhengxi Chen

Date 12/2/2003

Spatial Data Mining Using Branch Grafted R-tree

Priyanka Dubey (MS, Computer Science)

University of Nebraska at Omaha

Advisor: Dr. Zhengxin Chen

Abstract

Spatial data mining is a process of extraction of implicit information, such as weather patterns around latitudes, spatial features in a region, etc., with a goal of knowledge discovery. The existing spatial data mining methods typically identify a specific data-mining task for knowledge discovery. An example of a mining task may involve finding weather patterns in the northwestern region of U.S.A. To find such weather patterns one could employ an existing data structure, such as a B+ tree followed by the analysis of the mined weather data for knowledge discovery. This is a typical top-down approach of identifying a task, selecting a data structure, followed by queries and analysis.

This thesis provides a method and a simulation for mining spatial rules for the purpose of knowledge discovery. The thesis takes a bottom up approach: it employs Branch Grafted R-tree for the storage and retrieval of spatial data, followed by identifying tasks, followed by spatial queries and analysis. The Branch Grafted R-tree is an efficient data structure more suitable for efficient retrieval of data. This type of bottom up approach is unique and takes the advantage of the previous work carried out using Branch Grafted R-tree.

The thesis extends [2] and [3] for spatial data storage and access. Their work can be used for conventional spatial querying and spatial OLAP/data mining. The simulation in the thesis forms a basis for the further development and implementation of a data mining method.

In this thesis several queries and functions such as 'close_to', 'adjacent_to', etc. are designed to extract desired information from the data stored in the Branch Grafted R-tree, by extending window search algorithms [3]. The algorithms are developed and simulated to analyze the data that results by executing queries, to mine spatial rules such as spatial association rules, aggregation rules, and discriminant rules. The Branch Grafted R-tree serves as a common ground for mining different spatial rules.

The data mining process in the thesis is simulation, carried out with relatively small data size. The Branch Grafted R-tree uses RAM for data storage and retrieval. In the real world, the implementation would have to address the issues of disk access due to large data size.

Table of Contents

List of figures

Chapter 1 - Introduction

Spatial data mining refers to the extraction of implicit knowledge, spatial relationships, or other patterns, not explicitly stored in spatial databases. *Spatial data* differs from data stored in relational databases. It carries topological information usually organized by multidimensional indexing structures, accessed by spatial data access methods. Spatial reasoning, geometrical computations, and spatial knowledge representational techniques are used for spatial data processing. Spatial knowledge discovery can be defined as a process of extracting valid, novel, potentially useful, and ultimately understandable patterns from data.

Spatial data databases are very large databases and require special handling techniques. This type of data collected over long period of time requires knowledge discovery (not defined at the time of data collection) for providing meaningful information. This requires efficient and effective handling of the data, developing optimized queries, followed by analysis for knowledge discovery. Spatial data mining aims at discovering various kinds of patterns (rules) of interesting and regular knowledge from large spatial database. One example of mining a spatial rule is to find big cities with nearby spatial features, like large water bodies.

The spatial data is represented in the form of x and y coordinates with some non-spatial attributes (e.g. name). The examples of spatial data include data about cities, rivers, roads, counties, states, and mountain ranges and non-spatial attributes are census, number of industries in a geographical region etc.

Spatial data mining thus requires understanding of the spatial data and the relationship between the non-spatial and spatial information. The spatial data involves organizing the spatial data, designing queries for information retrieval with the intent of knowledge discovery and for analyzing result-sets to mine rules for knowledge discovery. In this thesis, programming techniques using Branch Grafted R-tree data structure have been developed for data mining. The Branch Grafted R-tree data structure has been used for the storage (in memory) of spatial data. R-trees are specialized forms of B+ trees adapted for the efficient representation, access and management of spatial data. R-trees are similar to B+ trees in the way the nodes are organized and managed [4]. The comparison between R-trees and B+ trees has been further discussed in the section 3.1.2. This Branch Grafted R-tree is like a basic R-tree with an enhanced method of record reorganization at the leaf node level resulting in a better performance over R-tree [2].

The existing data mining methods use the data structure such as R-tree, MBR technique or their variations. These methods focus on the task of mining itself and have strength in problem solving at conceptual level because of their top down approach. In one example the high level goal is to mine spatial association rule. A spatial association rule describes

the implication of one or a set of features by another set of features in spatial databases [9].

First relevant data set from the relational database (containing spatial data) is queried to extract the task relevant objects. The task-relevant-objects are refined at a coarse resolution level using R-trees, MBR techniques, and plane-sweep algorithms to extract the objects that are close to each other. The coarse level data further refined and filtered to find predicates. Apriori algorithm is then used for mining spatial association rules.

This thesis takes a different approach than the above-described methods. Simulation of the Branch Grafted R-tree (the generic spatial data of interest from text files stored in Branch Grafted R-tree) is first carried out. The goal is to efficiently retrieve relevant data objects. Thus, the approach is a bottom up approach. The bottom-up approach exploits the performance advantages of the Branch Grafted R-tree. The specific spatial rules are then mined as a last step.

The thesis extends the previous work, a thesis [2] and a thesis-equivalent project [3] on spatial data access methods developed on a Branch Grafted R-tree. The thesis equivalent project [3] extends the thesis work by [2] to access spatial data for conventional spatial querying and spatial OLAP/ data mining. The approach of the thesis is to integrate the conventional spatial query methods and spatial OLAP/ data mining by extending the

previous works [2] and [3]. Different data mining tasks use Branch Grafted R-tree as a common data structure.

The Branch Grafted R-tree is a general-purpose spatial data management data structure. The Branch Grafted R-tree has been analyzed and compared with R-tree [2]. The comparison shows that Branch Grafted R-tree has a better and improved performance over R-tree. Another important factor is that a spatial query design for the retrieval of coarse information. The mining of rules totally depends on the above-mentioned data. This thesis focuses on developing a method of spatial query design and functions such as 'close_to', 'within', 'intersects', etc. This is followed by the analysis of coarse data (relevant information) retrieved as a result of spatial query. The algorithm for analysis has been specifically designed for the thesis work. The Future Work section discusses the limitations of the work.

1.1 Organization of the thesis

- Chapter 2 is a review of literature and related previous works, data mining techniques, applications, and mining rules.
- Chapter 3 contains issues and challenges in spatial data mining. Spatial queries, data structures, and search algorithms used in data mining have been discussed.
- Chapter 4 covers details of the design of queries, functions and data mining rules for the thesis work.

- Chapter 5 describes the structure and organization of related spatial data, description of the algorithm for query retrieval, description of search algorithm, and description of analysis process.
- Chapter 6 discusses the results obtained, conclusion, and future work.

Chapter 2 -- Literature Review

Spatial data mining is the extraction of implicit knowledge, general relationships between spatial and non-spatial data, or other interesting patterns and characteristics not explicitly stored in spatial databases. Such discovery of knowledge may play an important role in understanding spatial data, capturing intrinsic relationships between non-spatial and spatial data, presenting data regularity in a concise manner and reorganizing spatial databases to achieve high performance.

Initially, statistical analysis was used for the mining of data. The statistical cluster analysis technique was modified for the use in spatial data mining [16]. The limitation of statistical approach to handle interrelated spatial data gave rise to other spatial data mining methods.

2.1 Issues Related to Spatial Data Mining

Over the years different methods related to spatial data mining have been developed. The main issues related to data mining are

a. Suitable data structures
b. Efficient and scalable data mining queries/ algorithms
c. Accurate and user friendly presentation of data mining results (mined spatial rules/ patterns)

2.2 Review of Spatial Data Mining Methods

Spatial data mining methods can be applied to extract interesting patterns and regular knowledge from large spatial databases. These methods are used for understanding spatial data, for discovering relationships between spatial and non-spatial data, and for constructing spatial knowledge bases and query optimization, and for capturing the general characteristic in concise manner. The spatial data mining methods have wide application in geographic information systems, remote sensing, image databases, imaging and other areas where the data type is spatial. The review of spatial data mining methods provides an overall picture of the methods of spatial data mining, their strengths and weaknesses. This review also provides the information about the work carried out so far and potential future challenges.

2.2.1 Statistical Method

Statistics is a most common approach for analysis of the spatial data. There exist a large number of algorithms and various optimization techniques [16, 24] for statistical analysis of spatial data. These methods handle the numerical data well and are based on the assumption of statistical independence among the spatially distributed data.

The assumption of statistical independence can cause problems in case of interrelated data. The neighboring objects influence spatial objects in consideration. The dependent variables can make the modeling process complicated. Statistical analysis requires the domain knowledge and statistical expertise. Statistical methods do not work well with

incomplete and inconclusive data. These methods could be computationally intensive and expensive.

2.2.2 Generalization Based Knowledge Discovery

Generalization based knowledge discovery needs background knowledge in the form of concept hierarchies. The concept hierarchies [17] are given by experts or, in some cases, automatically generated by data analysis. The idea behind the generation of concept hierarchy is to group the data based upon their common attributes or characteristics. Figure 2.1 shows an example of concept hierarchy. As we ascend the concept tree, information becomes more and more general and remains consistent with the lower concept levels.

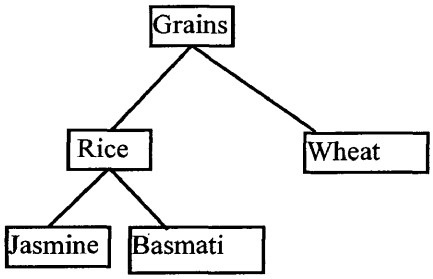
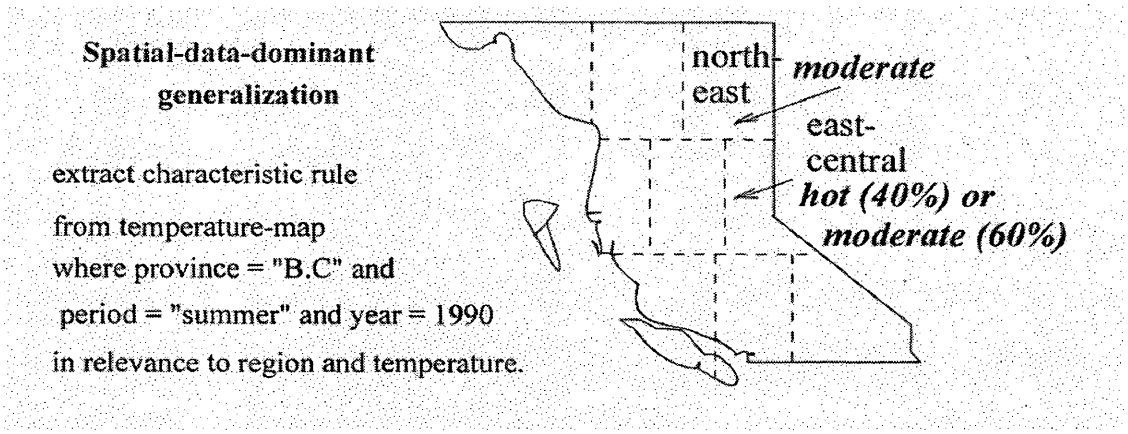


Figure 2.1 an illustration of Concept Hierarchy

Both *jasmine* and *basmati* can be generalized to the concept *rice,* which in turn can be generalized to the concept *grains*, which also includes *wheat*

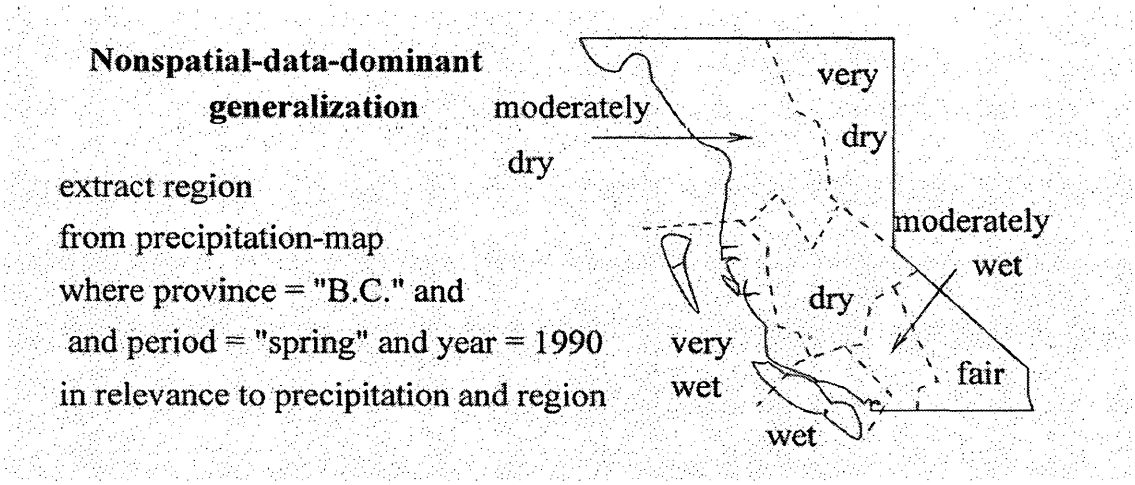
There are two types of concept hierarchies: Spatial and Non-Spatial. [18] The Spatial Data dominant generalization (figure 2.2) performed on spatial data by merging the spatial regions according to a description stored in a concept hierarchy, which may lead to a map, consisting of small number of areas. Then, a non-spatial description of each area is produced using the attributes. In the figure2.2 the non-spatial attributes are moderate and hot while the spatial regions are northeast and east central.

Figure 2.2 [7] Example of a query an the result of the execution of the spatial-data-dominant generalization method



The Non-Spatial Data dominant generalization (figure 2.3) is used for generalization of non-spatial data to a higher concept level. The algorithm creates maps, which consist of a small number of regions sharing the same high-level non-spatial description. The neighboring areas with the same generalized attribute values are merged together. In the figure 2.3 the generalized attribute values are 'very dry', 'moderately wet', 'fair', 'wet' etc. The user's requests in the form of queries initiate the general data characteristics and discovery process. The syntax of query is similar to SQL.

Figure2.3 [7] Example of a query and the result of execution of the non-spatial-data-dominant generalization method



The drawbacks of the above-described approach are:

In some cases such hierarchies are not present as priori. In case of spatial data the components are merged at a lower level of concept hierarchies. The quality of the mined characteristic rules depends considerably on concept hierarchies especially in case of spatial data. In many cases such hierarchies given by experts may not always be entirely appropriate.

2.2.3 Clustering

Data clustering is a branch of statistics. Data clustering discovers the distribution pattern of the data set in large spatial databases [13]. The advantage of this method is that the clusters can be found without much background knowledge or concept hierarchies. The techniques PAM (Partitioning Around Medoids) and CLARA (Clustering Large Applications) [19] have been used to discover the patterns. These algorithms attempt to

define clusters in a data set. PAM attempts to define cluster in the whole data set while CLARA operates on a sample of the data set and uses PAM algorithm on the samples to define clusters. Assuming that there are n objects, PAM finds k clusters by first finding a representative object for each cluster. A most centrally located point in the cluster is called as medoid. After selecting k medoids, the algorithm repeatedly tries to make a better choice of medoids analyzing all possible pairs of objects such that one object is medoid and other is not. CLARA draws multiple samples outputs the best clustering out of these samples. Both these algorithms are proven to be inefficient from the computational complexity point of view. A new algorithm CLARANS (Clustering Large Applications based upon Randomized Search) developed by Ng and Han [13] for cluster analysis helped improve the quality and efficiency of clustering. The main advantage of the technique is that cluster can be found directly from the data without the need of concept hierarchies.

The CLARANS algorithm assumes that the objects need to be clustered in the main memory. This may not be valid in case of large databases. The integration of CLARANS with R*-tree [23] to reduce the cost of computations alleviates the above described drawback. The medoids of the clusters are easily calculated with the help of central most object of a leaf node of the R*-tree containing neighboring points. The queries retrieve the central object of a leaf node.

2.2.4 Approximation and Aggregation

The aggregate proximity is the measure of closeness of set of points in the cluster to a certain feature. This method finds features of the clusters. Knorr and Ng [12] presented a method to find features close to clusters. The algorithm reports the features with the best aggregate proximity showing minimum and maximum distances of points in the cluster, average distance, and percentages of points, located in the distance less than specified thresholds.

The algorithm proposed by Knorr and Ng measures the aggregate proximity in order to find the reason of cluster formation. At the same time CLARANS (Clustering Large Applications based upon Randomized Search) algorithm is used to locate the clusters.

The use of k nearest neighbor searches using structures such as k-d trees, R-tree and its variants seem an obvious solution but such searches are unable to perform the search needed for their purpose.

To solve the problem Knorr and Ng [12] proposed CRH (C-encompassing circle, R-isothetic rectangle, H- Convex hull) algorithm that uses the computational geometry [20] concepts for distance computations, shape description, and overlap computations. The CRH algorithm is scalable, consistent, efficient, and computes relatively faster.

The approximation and aggregation methods consider features of the neighboring objects to find the characteristic of the clusters. The problem with this method is that a feature may have to be tested for overlap with a cluster many times. This may result in multiple computations.

2.2.5 Spatial Data Cube Construction and Spatial OLAP

On-line analytical processing (OLAP) [21] describes a class of technologies designed for live ad hoc data access and analysis. While transaction processing generally relies solely on relational databases, OLAP has become synonymous with multidimensional views of data. OLAP is generally used for the on-line mining of data. The relational data helps to integrate the spatial data to construct a data warehouse that facilitates spatial data mining. The fast and flexible on-line analytical processing for spatial data mining in spatial data warehouse has always been a challenge. For example, with a spatial data warehouse that supports spatial OLAP, a user can view a weather pattern on a map by month, by region, and by different combinations of temperature and precipitation. The user can dynamically drill down or roll up along any dimension to explore desired weather patterns such as "wet and hot regions in California in Summer 2000."

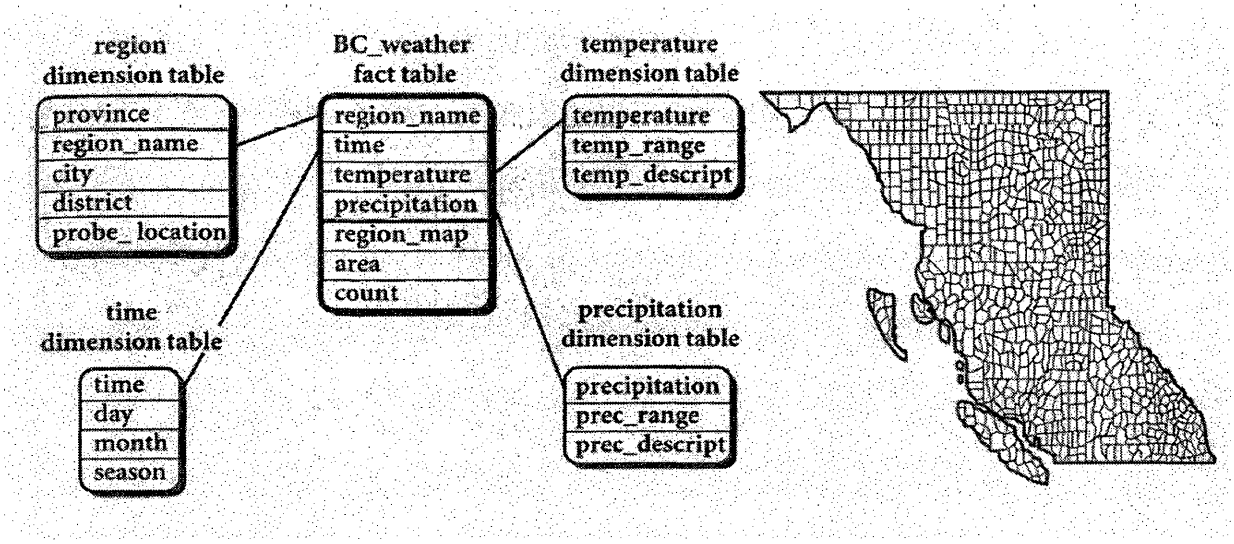
The integration of spatial data from heterogeneous sources and systems is a challenge regarding construction and utilization of spatial data warehouse that support OLAP. The star schema model in figure 2.4 is a good choice for modeling spatial data warehouse since it provides a concise and organized warehouse structure and facilitates OLAP

operations. In a spatial data warehouse both dimensions and measures may contain spatial components. A spatial data cube has three types of dimensions namely a non-spatial dimension, spatial to non-spatial dimension, and spatial to spatial dimension.

- A non-spatial dimension contains only non-spatial data.
- A spatial to non-spatial dimension is a dimension whose primitive-level data are spatial but whose generalization, starting at a certain high level, becomes non-spatial dimension.
- A spatial to spatial dimension is a dimension whose primitive level and all of its high-level generalized data are spatial.
- A numerical measure contains only numerical data. For example one measure in a spatial data warehouse could be the monthly revenue of a region, so that a roll-up may compute the total revenue by year, by country, etc.
- A spatial measure contains a collection of pointers to spatial objects.

In general OLAP provides users flexibility to carry out a given task dynamically. The efficient implementation of spatial data cube and OLAP is necessary otherwise on-line computation becomes a costly operation. Unlike non-spatial data, for spatial databases grouping and hierarchies can be numerous and unknown at design time therefore well known OLAP techniques may not be directly applicable. Many applications such as traffic supervision in a region generally need summarized data. This information can be obtained from transactional spatial databases but it may be computationally expensive. The spatial data warehouse and spatial cube construction are pre-requisites for OLAP.

Figure2.4 [7] A star schema of BC_wheather spatial data warehouse and corresponding British Columbia (Canada) weather probes map



2.3 A Brief Discussion about Data Mining Methods

The spatial data mining methods described in this chapter have their limitations with the size or the type of data. The statistical method described in section 2.2.1 may hide valuable information because these methods draw inferences based upon numbers that are result of intensive computational manipulations. Such results may not represent the population well. The generalization based knowledge discovery method (section 2.2.2) requires the existence of background knowledge in the form of concept hierarchies. The advantage of the clustering method (section 2.2.3) is that clusters can be found without much background knowledge or concept hierarchies. The medoids (a most centrally located point in the cluster) of the clusters are easily calculated with the help of central most object of a leaf node of the R*-tree containing neighboring points. R*-tree helps to reduce the cost of computations. The approximation and aggregation method (section 2.2.4) considers features of the neighboring objects to find the characteristic of the clusters, thus focusing on clusters only. The problem with this method is that a feature may have to be tested for overlap with a cluster many times. This may result in multiple computations. The on-line analytical processing (section 2.2.5) is a class of technologies designed for live ad hoc data access and analysis. OLAP provides users flexibility to carry out a given task dynamically. The efficient implementation of spatial data cube [22] and OLAP is necessary otherwise on-line computation becomes a costly operation.

The literature review continues through Chapter 3 with the review of various data structures including R-tree, Branch Grafted R-tree, and description of branch grafting algorithm. The review continues with the discussion of spatial queries and algorithms to access spatial data in a R-tree. The work in the thesis has modified windows search algorithm to search desired spatial object stored in the Branch Grafted R-tree. The literature review in Chapter 3 also contains the review of spatial data mining rules such as spatial association rule, discriminant rule, aggregation rule, and classification rule followed by justification of the thesis work.

Chapter 3 - Spatial Queries using Branch Grafted R-tree

3.1 Review of Data Structures for Spatial Data Mining

Spatial operations like spatial joins, map overlays, and spatial queries are used in the algorithms for spatial data mining. Spatial data access methods and data structures have an important role in spatial data mining. Spatial data structure consists of points, lines, rectangles, etc. Multidimensional trees such as Quad-trees, K-d trees, R-trees, R*-trees are used to build the indices of these trees. Spatial join is one of the expensive spatial operations. The section 3.1 describes some of the important data structures used for organization and storage of spatial data. The important data structures are R-tree, R^+-tree, R*-tree, and Branch Grafted R-tree. The thesis uses Branch Grafted R-tree due to improved performance over R*-tree. The insertion and search operations play a key role in the data retrieval used in the thesis.

3.1.1 B^+-tree

B+-tree [4] is most widely used dynamic data structure. It is balanced tree in which the internal nodes direct the search and the leaf nodes contain the data entries. The searching for records requires just a traversal from root to a leaf. The purpose of the intermediate nodes is to hold keys that partition and refine the node domain. Figure 3.1 shows and example of a B+ tree data structure.

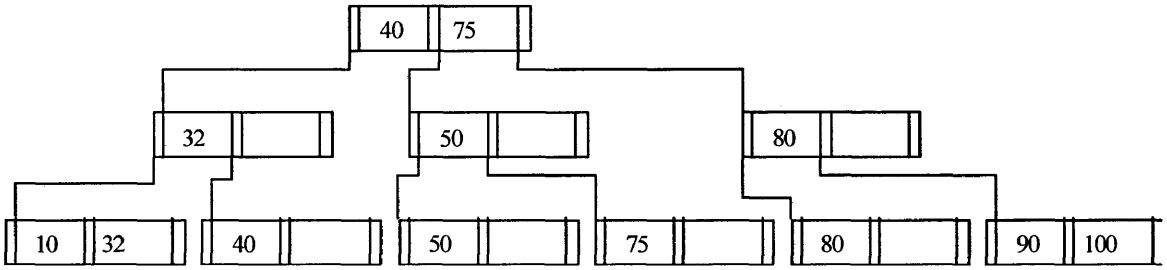


Figure 3.1 An Example of B+ tree

3.1.2 R-tree

R-tree [5] is a simple data structure used in spatial data mining. It is a height-balanced tree that contains all leaves at the same level. The root node has at least two children. The non-leaf nodes contain $\lceil m/2 \rceil$ entries, where m is the maximum number of entries in a node. The spatial data is represented as rectangles called minimum bounding rectangle. A non-leaf node contains information of the bounding rectangle, in the form of the coordinates, of its child node. The leaf node of an R-tree stores the record of spatial data stored, in the form of the coordinates of a rectangle, in relational database.

R-trees are specialized forms of B+ trees adapted for the efficient representation, access and management of spatial data. R-trees are similar to B+ trees in the way the nodes are organized and managed [4]. In R-tree and B+ trees, the actual data either resides in the leaf nodes or is directly pointed to by the leaf nodes. The intermediate nodes hold keys. These keys partition and refine the node domain as one travels from the root node to the leaf nodes. The spatial data represents n-dimensional objects and often quite large. If this data were stored in a normal B+ tree, the nodes would only be capable of holding a few records and the tree would become high. The height of the tree will result in poor performance of several operations. In R-tree by locating the data, or pointers directly to the data, in the leaf nodes allows one to store more intermediate node records in fewer nodes, making the resulting tree height considerably lower.

The two trees differ in their representation and management of keys. The keys in an n-dimensional R-tree may be coordinates of the minimum point (x,y,z,... coordinates) and the dimensions of the minimal n-dimensional box that bounds the n-dimensional boxes of all child nodes under the said node. These differ from traditional B+-tree keys, which are generally one-dimensional with an order implied among records within each node. The maximum number of records per node for the B+ tree in figure 3.1 is two and the minimum number of records per B+ tree node is one.

3.1.3 R^+ tree

R^+ tree is similar to R-tree [10] in structure. The R^+ tree stores bounding rectangles that do not overlap. In case of overlapping rectangles split algorithm ensures that the rectangle that is overlapping the two bounding rectangles splits into two disjoint rectangles. Such a node splitting algorithm is complex and produces duplicate data records. The search efficiency of R^+ tree is better than that of the R-tree.

3.1.4 R* tree

R*-tree [4] [23] uses the basic R-tree and utilizes a split algorithm to minimize cover, overlap, and margin of each bounding rectangle in the nodes. This tree supports both point and spatial data. It is more expensive to implement in comparison to R-tree. Data representation using an R*-tree is more accurate resulting in efficient query retrievals, insertions and deletions.

In order to reduce the need of splitting nodes, certain numbers of records are deleted from the node and reinserted in other nodes. This improves the quality of data organization in the R*-tree.

3.1.5 Branch Grafted R-tree

The Branch Grafted algorithm uses the R-tree with better reorganization of records at leaf-node level. The algorithm uses the grafting of a leaf or a branch node in the tree in order to reduce the number of nodes in the R-tree. The more accurate data structure improves the performance of query. The split in R-tree does not consider rest of the nodes in the tree while the Branch Grafted tree scans the other nodes in order to find an opening. This opening can accommodate a rectangle. This could avoid the need of a split and reduces the number of node in the tree. The improvement in search time is a result of lesser number of nodes in the tree.

The data structure used for storing spatial data in the thesis is Branch Grafted R-tree. Reference [2] carried out the implementation of the data structure.

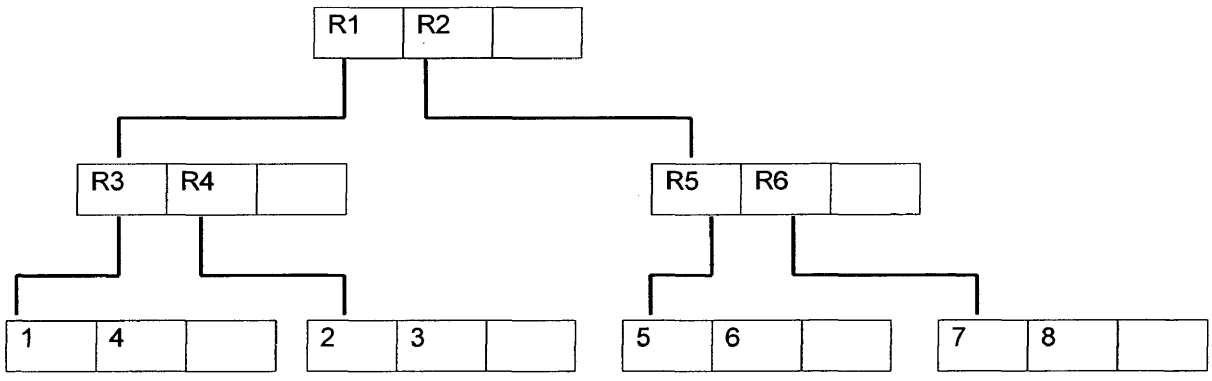


Figure3.2 An Example of Branch Grafted R-tree

3.1.5.1 Branch Grafting Algorithm Description

The Branch Grafting algorithm [1] and [2] improves the accuracy and speed of the node overflow operations. In case of a node overflow, the Branch Grafting algorithm first looks for records in the parent node that overlap the bounding directory of the full node. Individual records in the full node are then evaluated for placement under the overlapping parent nodes. Records are moved under a parent node if the resulting area of coverage for all nodes involved is smaller. If no records are moved to make room for the initially inserted record, then a split is performed using the original R-tree split algorithm.

The R-tree is a hierarchical data structure in which each node corresponds to the smallest d-dimensional rectangle that encloses its child nodes. The leaf nodes contain pointers to the actual data in the database. The parameters of the tree are chosen such that a small number of nodes are visited during the spatial query. This increases the response time of the data structure. The rectangles corresponding to different nodes may overlap, so that the spatial query may often require several nodes to be visited before ascertaining the presence or absence of a particular rectangle.

Figure3.3 Branch Grafting Insert Pseudo Code

This describes the record insertion process. The letter "N" represents a node in the tree. The letter "E" represents the new entry to be added and the letter "M" represents the maximum number of records per node.

S1 – invoke choose sub tree with the desired tree level as a parameter to find a node N into which the new entry E may be placed.

S2 -if N has less than M records
then insert E into N.
else invoke the overflow operation
end if (S2)

S3 – adjust all covering rectangles in the insertion path such that their minimum bounding rectangles enclose their child nodes. Grow the tree if necessary.

Figure3.4 Branch Grafting Overflow Pseudo Code

This describes the process to deal with attempted insertions into nodes that are full. The letter N represents the full node.

S1 – invoke the find opening operation
S2 – for each candidate target node
invoke the make move operation
if one or more nodes are moved out of N
then return success.
else invoke split node
end if
end for

3.2 Spatial Queries

Location based and feature based queries are the two main types of spatial queries. The first type of query is location-based query. In this case, we search for the nature of the feature associated with a particular location or in its proximity. The second is feature-based. In this case, we probe for the presence or absence of a feature, as well as seek its actual location.

The location-based queries are used to search the features associated with a location. These types of queries are easy to answer. The queries search tree in top down manner until the object is found. In order to find the feature in the proximity of a particular location the search is continued in the neighborhood of the node containing the object.

The feature-based queries are considered to be difficult due to absence of indexing-by-features that are related to spatial objects. The indexing is based on spatial occupancy only. The feature based queries process the data without examining every location in space.

The types of the location-based queries are regional-queries and point-queries. A regional query finds all data objects that intersect a given spatial rectangle formed around a spatial object that overlap or fall within the query window.

A point-query is a special case of the region query in which an aligned rectangle is a single point.

Following are some types of location-based queries [3]

1. Point-in-polygon query: Find all objects containing a point.
2. Window query: Find all objects within a query rectangle.
3. Overlap query: Find all objects that overlap a query rectangle.
4. Nearest neighbor query: Find the object closest to a query object.
5. Distance scanning: Enumerate the objects in increasing distance from a query object.
6. Spatial join: Find all pairs of objects that satisfy join condition, which may be intersection, containment, equality or others.
7. Buffer query: Determine the buffer around a query object, i.e., this query determines the amount of dead space in a node.
8. Path query: Determine a path between two points.

Spatial operation and spatial-join operations can be implemented by extending the window search algorithms described in the literature review (Chapter 2). The 'overlap' algorithm can be used to find objects that satisfy the join condition. The Contains algorithm can be used to find all objects that satisfy the given containment condition.

The objective of the path query is to find a path between two objects. The path query can be implemented by combining the window-search algorithm and the nearest-neighbor algorithm. This can be achieved by calculating the distance between coordinates of the

centers of the two points. A path must be found between these two points. These two centers can be used as the coordinates of the search window. A window search operation can be executed to find all data rectangles that lie in this window. The nearest-neighbor algorithm is used to find the closest data rectangles. This rectangle is the path between the two points.

The objective of spatial data access method is to minimize the number of index pages for a given query. The three main design objectives namely area, overlap, and perimeter should be considered to achieve the best possible performance in an index structure for multidimensional objects [4].

The area of the minimum-bounding rectangle that corresponds to different nodes must be small. The smaller area increases the probability of accessing a particular node. The area covered by a bounding rectangle should be reduced in order to minimize the dead space. This improves performance because the path to be traversed can be decided at a higher level.

The overlap between different minimum bounding rectangles is a critical factor. The overlap usually occurs in densely populated regions of the database. The overlap must be kept small as it reduces the number of paths to be traversed.

The minimum bounding rectangles are best for region queries. It is important that the perimeter of the minimum bounding rectangles also be as small as possible. The minimum margin of a bounding rectangle makes the rectangle more quadratic because, for a fixed area, the rectangular object with the smallest margin is a square. The quadratic rectangles can be packed easily and thus build a smaller rectangle.

The point-in-polygon query is a special case of window search algorithms (Contains). The nearest-neighbor query has been used in the nearest neighbor algorithm implemented in the thesis.

Branch Grafted R-tree has an advantage when a huge amount of spatial data needs to be stored. Grafting reduces the height of the tree, which helps in information retrieval from the tree. The information retrieval is faster as the search algorithm traverses fewer search paths due to lesser number of nodes reduced by grafting.

3.3 Algorithms to Access Spatial Data

Once the spatial data is organized using an efficient data structure such as Branch Grafted R-tree, a data retrieval method is needed for further processing. Section 2.3 contains a brief description spatial data access/ search algorithms. The algorithms have been implemented to test the performance of the Branch Grafted R-Tree [3].

3.3.1 Window Search Algorithm (Overlap)

The original overlap algorithm was described in [5]. This algorithm, (figure 3.5) finds all the objects that overlap a specified search window and can be used for location-based queries. It finds all the data records represented in the form of rectangles in the data structure that intersect the specified query object.

The algorithm is implemented by a complete tree traversal i.e. by visiting representative blocks in a top-down manner. The algorithm checks for overlap at each stage. In case of overlapping- non-leaf node the algorithm works recursively to reach at the lowest level of hierarchy. The objects that match with the query window are considered to be satisfying the query.

Figure 3.5 Overlap Algorithm Pseudo Code

```
if data record overlaps search window
        return true
else
        return false

[or]

if ((srch_win.x > (data_rec.x + data_rec.dx)) ||
         ((srch_win.x + srch_win.dx) < data_rec.x) ||
         (srch_win.y > (data_rec.y + data_rec.dy)) ||
         ((srch_win.y + srch_win.dy) < data_rec.y))
         return false
else
         return true
```

3.3.2 Window Search Algorithm (Contains)

This is a variation of the above described overlap algorithm. This algorithm finds all objects within a specified search window. This algorithm is implemented by a full tree traversal. The method involves visiting the blocks in a top-down manner. At each stage the block is checked for whether it lies completely within the given query window. In case of a non-leaf block that satisfies the given condition, the algorithm is applied recursively to reach at the lowest level of the hierarchy. The objects in the block that satisfy the given query window are reported. The method developed in the thesis uses similar approach for the retrieval of data stored in Branch Grafted R-tree. The implementation details have been discussed in the Chapter 4.

Figure 3.6 Window (Contains) Search Algorithm Pseudo Code

```
To find all index records that fall within the search window S.
Root Node – T
Search Window – S
Index Record – E

S1 –    Search subtree
        if T is not a leaf node
                then check each entry E to determine if rectangle
                part of index entry E is enclosed by S

                for all matching entries, call search on the tree
                whose root node is pointed to by the child pointer of E
                end for
        end if
S2 –    Search leaf node
        if T is a leaf node
                then check all entries E to determine if rectangle
                part of index entry E is enclosed by S

                if it is enclosed return E
                end if
        end if
```

3.3.3 Search by Distance Algorithm

The search-by-distance algorithm (figure 3.7) works similarly to the nearest-neighbor algorithm [8]. The algorithm has metrics such as search by the minimum distance, center-to-center distance, and maximum distance from the query window. The algorithm is implemented by a tree-traversal method, where the blocks are visited in a top-down manner. It checks at each stage the distance between the block and the query window. If the distance between the two is equal to the specified distance then the block is considered. If the block in the tree is a non-leaf node then the algorithm is applied recursively to reach at the lowest level of the hierarchy. If the objects at the leaf nodes are at the specified distance from the query window, then they are reported.

At each node, the algorithm sorts all of the entries using minimum distance MINDIST (P, R), maximum distance MAXDIST (P, R), and center distance CTRDIST (P, R) between the data records where P is the data record and R is the search window.

Figure 3.7 Search by Distance Algorithm Pseudo Code

To find all records that lie within a specified distance from the search window.

Root – T
Search Window – S
Distance – D

```
S1 –    Search subtree
        if node is not a leaf node
                then check each data record of the node to determine
                if the distance of the data record E from the
                search window is less than D

                for all records that match call the search routine
                on the tree whose node is pointed to by the child
                pointer of index record E
                end for
        end if

S2 –    Search leaf node
        if the node is a leaf node
                for all data records E of the node
                if the distance from the search window is less than
                D then return E
                end for
        end if
```

3.3.4 Nearest Neighbor Algorithm

This algorithm is an extension of the search-by-distance algorithm. The nearest-neighbor-search [8] finds points with minimum point-to-point distance from a given query point. The Nearest Neighbor ranks all objects in terms of their distances from a query in an incremental manner. The nearest neighbor algorithm uses a list/queue as a data structure to store the blocks of the underlying data structure and objects.

The algorithm works in a top-down manner in the tree. In this process the algorithm computes the distance between the centers of the bounding rectangle of the record and the query window. The record that lies within the specified distance is considered to be the candidate record. The nearest neighbor is retrieved from the elements of the candidate record list or the neighbor list.

The method developed in the thesis uses similar approach for the retrieval of data stored in Branch Grafted R-tree. The implementation details have been discussed in the Chapter 4.

Figure 3.8 Nearest Neighbor Algorithm Pseudo Code

```
To find "n" closest records from the search window
Root – T
Search Window – S
Distance – D

S1 –    Search subtree
        if node is not a leaf node
                then check each data record of the node to determine
                if the distance of the data record E from the
                search window is less than D

                for all records that match call the search routine
                on the tree whose node is pointed to by the child
                pointer of index record E
                end for
        end if

S2 –    Search leaf node
        if the node is a leaf node
                for all data records E of the node
                if the distance from the search window is less than
                D then return E

                call insert neighbor function to insert the record into the nearest neighbor list
                end for
        end if
S3 –    Find nearest neighbor
        for all records in the nearest neighbor list find n closest neighbors
        return
        end for
```

3.4 Extending Spatial Queries for Spatial Data Mining

The search algorithms and spatial queries mentioned in the previous sections retrieve the relevant information from the large spatial data set. The representation and analysis of the information, thus obtained as result of spatial queries, is important for the better understanding, usage, and meaningful insight into the spatial data. Recent studies of mining spatial rules have led to a set of interesting techniques to represent the patterns and features in a spatial data set. Strong spatial rules indicate the patterns and implication relationships in the large spatial data set. The following section contains a review of four spatial rules: association rules, classification, aggregation, and discriminant rules.

3.4.1 Spatial Association Rule

Spatial association is a rule that describes the implication of one or a set of features by another set of features in spatial database. A spatial association rule is of the form "X → Y" where X and Y are the sets of predicates and some of them are spatial ones. In a large database many relationships may exist but some may occur rarely and may not hold in most cases.

Koperski and Han [7] suggested a method in order to find various rules. First, the set of relevant data is retrieved by executing of the data retrieval methods of the data mining query, which extracts the following data sets whose spatial portion is inside British Columbia: (1) towns: only large towns; (2) roads: only divided highways; (3) water: only

seas, oceans, large lakes and large rivers; (4) mines: any mines; and (5) boundary: only the boundary of B.C., and U.S.A.

Secondly, the "generalized close to" (g_close_to) relationship between (large) towns and the other four classes of entities is computed at a relatively coarse resolution level. This is achieved by using a less expensive spatial algorithm such as the MBR data structure and a plane sweeping algorithm or R*-trees and other approximations. The derived spatial predicates are collected in a *g_close_to* table, which follows an extended relational model: each slot of the table may contain a set of entries. The support of each entry is then computed. The entries with support below the minimum support thresholds are removed from the table.

Notice that from the computed *g_close_to* relation, interesting large item sets can be discovered at different concept levels and the spatial association rules can be presented accordingly. For example, the following two spatial association rules can be discovered from this relation.

is_a(X, large town) ^ g_close_to(X,water) : (80%)

is_a(X, large town) ^ g_close_to(X,sea) → g_close_to(X,us boundary) : (92%)

3.4.2 Spatial Classification Rule

Classification is a data mining technique where the data stored in a database is analyzed in rder to find rules that describe the partition of the database in a given set of classes [15].

Geographic data consists of spatial objects and non-spatial description of these objects. Non-spatial descriptions can be stored in a relational database with a pointer to the spatial description of the object. The process of spatial classification is to find a rule to partition a set of classified objects into a number of classes using spatial relations of the classified objects to other objects in the database.

The example of the classification rule is "big cities in North America are closer to big rivers while smaller cities are closer to smaller rivers". In this example the cities have been classified based upon the size of the river closer to them.

3.4.3 Spatial Aggregation Rule

Aggregate values for areas close to spatial objects plays a very important role in the analysis of many business objects like stores, gas stations, etc. [15]. To handle the aggregate values of non-spatial attributes in thematic maps one can calculate aggregate values for the intersecting objects. The aggregate data can also be generalized and merged with the predicates. Finally each object can be classified using a set of predicates describing properties of both thematic map and other intersecting objects. One example

of aggregation rule is the total number of cities in Canada located close to USA Canada border.

3.4.4 Spatial Discriminant Rule

A spatial discriminant rule is a general description of the contrasting or discriminating features of a class of spatial-related data from other classes. Discriminant-rule mining is basically mining of a set of comparison rules that contrast the general features of different classes of the relevant sets of data in a database. This is a comparison of one set of data, known as the target class, to the other sets of data known as contrasting class (es).

For example comparison of the weather pattern in two geographic regions is a spatial discriminant-rule.

3.5 Motivation and Focus of the Thesis

There are several spatial data mining methods discussed in the Chapter 2. The purpose of the discussion of several data mining methods is to provide the overall picture of spatial data mining process. Each method has been discussed with its advantages and shortcomings. Individual description of each method for spatial data mining also provides a brief comparison with the method developed in the thesis.

The main issues and challenges with the spatial data mining are to retrieve data faster, to avoid heavy computations, to provide a simple user interface, and to represent results comprehensively.

The method developed in this thesis focuses on the above challenges. The data structures used for the storage and organization of spatial data play an important role in the mining of spatial data. Many data mining methods use relational database structure for the organization and storage of data. The various types of data structures have been discussed in the section 3.1. R-tree is a general-purpose algorithm used in the mining of spatial data. The data structure used in the method developed in the thesis is Branch Grafted R-tree. The Branch grafted R-tree is an improved data structure. The grafting process results in a reduced height tree. The reduced height tree helps in faster data retrieval i.e. better performance of Branch Grafted R-tree over R-tree. The spatial data in the form of coordinates of bounding rectangles, is stored and organized using Branch Grafted R-tree. The spatial data stored in the Branch Grafted R-tree is retrieved with the help of algorithms similar to the search algorithms described in the section 3.3. The idea behind the search is to traverse the tree to find nodes within a certain distance from a given node. Minor computations used in the search process help to make mining efficient and fast.

A good user interface is an important factor for spatial data mining process. The method developed in the thesis is initiated by SQL-like query. Several functions and search

algorithms developed in the thesis solve queries that retrieve data stored in the Branch Grafted R-tree. The end user can mine data using queries as an easy to use and flexible interface. The data retrieved as a result of several queries performed on a given spatial data set. The result data set is then analyzed to mine spatial rules viz. spatial association rule, discriminant rule, aggregation rule, and classification rule.

The approach to mining spatial rules in the thesis is different from the previous methods discussed in the literature survey. This is a unique bottom up approach where the performance of the data structure (Branch Grafted R-tree) has already been tested and analyzed. There is a hope of developing a spatial data mining method using Branch Grafted R-tee.

In the Chapters 2 and Chapter 3 the focus was to review the data structures to store spatial data and different methods to search the data structure. The next chapters describe the development of a method to mine spatial rules using the Branch Grafted R-tree for storage of the spatial data.

Chapter 4 - Spatial Data Mining Program Construction

In the previous chapter different data structures, algorithms to access data in Branch Grafted R-Tree data structures, spatial queries and their design were discussed. The extension of spatial queries for mining spatial rules was also discussed. This was then followed by a brief discussion on motivation and focus of the thesis. Next, specific algorithms that are used in this thesis, to retrieve data from data structures and for mining spatial rules, are discussed.

This chapter starts with the description of the modules of the programs and their objectives. The chapter continues with the description of spatial queries developed in the thesis. The queries designed in this method are implemented for the retrieval of data stored in the Branch Grafted R-tree. The data stored in the Branch Grafted R-tree is in the form of coordinates of bounding rectangles of spatial objects. The chapter ends with the discussion on complexity analysis of the algorithms in the thesis.

4.1 Module design layout

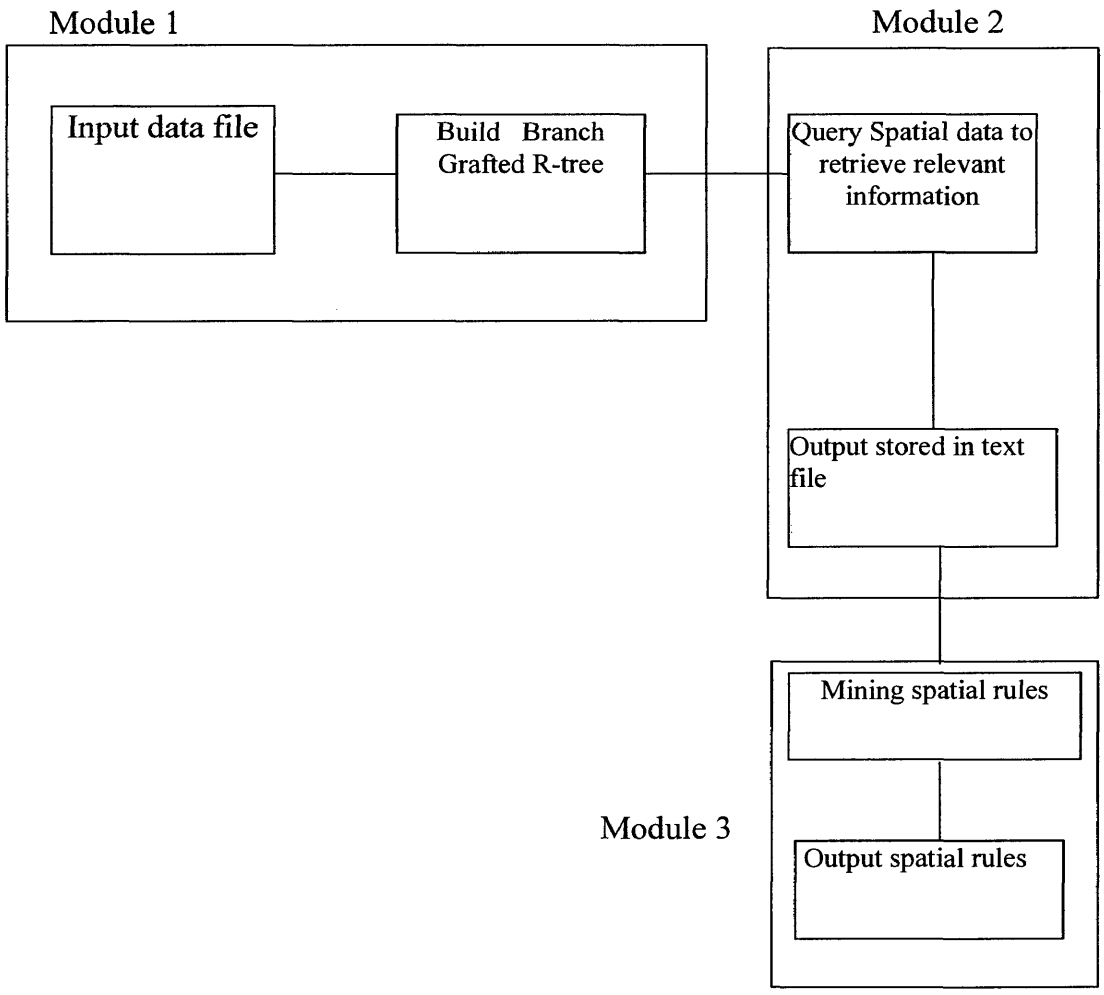
The program to mine spatial rules contains three parts viz. building Branch Grafted R-tree, spatial query to retrieve relevant information from the pool of spatial data stored in the tree, and mining spatial rules from the relevant data.

The link between the three modules described above has been shown in the figure 4.1. The Module 1 builds a data structure using input data files and stores the data in the Branch Grafted R-Tree. The algorithm of Branch Grafted R-tree [2] has been used for this purpose. The details of the data used in the program are described in section 5.1. The algorithm itself has been described in section 3.1.4.1 of the Chapter 3. The input data file is the spatial data used for data mining e.g. town, river, water body data, schools, mines, hospitals etc. The input user queries in the program are also fed in the form of a text file.

The Module 2 is designed to retrieve relevant data stored in the Branch Grafted R-Tree with the help of queries and stores the relevant output in a text file. The program first takes one query and two data files as input and finds the coordinates of the query object, e.g. a 'water_body', and searches the tree that stores 'town' data to find the matching coordinates that satisfy the given criteria such as nearest, close to etc. The matching coordinates are stored in an output text file.

The Module 3 describes the mining of the spatial rules viz. spatial association rule, aggregation rule, and discriminant rules from the pool of matching coordinates. These rules are basically statistics about a certain criteria for example percentage of major cities in US located close to a water body. These spatial rules have been explained in detail in the later part of the chapter.

Figure 4.1 Module design layout of the program



4.1.1 Spatial Query Design

The purpose of the query is to search the spatial data stored in Branch Grafted R-tree. The data records stored in the Branch Grafted R-tree are in the form of bounding rectangles of the spatial objects. These records are stored at the leaf nodes.

Functions were designed and written in C language to help in the implementation of such queries. Combinations of these functions were then used to design meaningful queries. These queries were then run on the Branch Grafted R-tree.

The main spatial operators used for the query design in the thesis can be mainly divided in two categories; topological and directional. The spatial operators such as 'containment', 'intersection', 'adjacency', 'inside', and 'within' fall under the category of topological operators while 'northwest', 'distance_compared_to', 'near', and 'close_to' are directional operators.

The operators described above play a key role in the implementation of the queries. These operators have been implemented in the form of functions to access spatial data. Spatial objects are interrelated objects. Spatial objects affect neighboring objects. A query using both topological and directional operators must be used to find neighboring objects.

The target spatial objects may intersect, overlap, be very close to, adjacent to, or exist inside the query object. I have implemented a number of functions to find all pairs of objects that satisfy a join condition, which may be intersection, containment, equality or others. These functions--'close_to', 'distant_from', 'adjacent_to', 'intersects', and 'within'--guide the search of spatial objects stored in the Branch Grafted R-tree. The role and details of these functions are described later in chapter (figure 4.2)

Main functions used for the query design are as follows:

Function name	**Parameters**	**Description**
close_to	(name,d) name: description of spatial object	Returns true if calculated distance between coordinates of two spatial objects (rectangles)<= d
distant_from	(name, d) name: description of spatial object d: distance	Returns true if calculated distance between coordinates of two spatial objects (rectangles)<= d
intersects	(name) name: description of spatial object	Returns true if calculated distance between coordinates of two spatial objects (rectangles) is <= 0
adjacent_to	(name) name: description of spatial object	Returns true if calculated distance between coordinates of two spatial objects (rectangles) is within the range of 1 to 5
near_to	(name) name: description of spatial object	Returns true if calculated distance between coordinates of two spatial objects (rectangles) is within the range of 1 to 10
contains	(name) name: description of spatial object	
inside	(name) name: description of spatial object	
within	(name) name: description of spatial object	

Figure 4.2 functions used in the spatial queries

The following example shows a method to solve topological operators such as 'intersects', 'adjacent_to', 'contains', 'within', and 'inside' in a spatial query. Figure 4.3 is a graphical representation of spatial data records. The search window is represented in the form of dotted lines represents the coordinates to be searched in a given spatial data pool stored in Branch Grafted R-tree.

The rectangles A and C are completely inside the dotted rectangle, B overlaps and D is completely outside. The search algorithm only considers rectangles A, C and B as 'hit'.

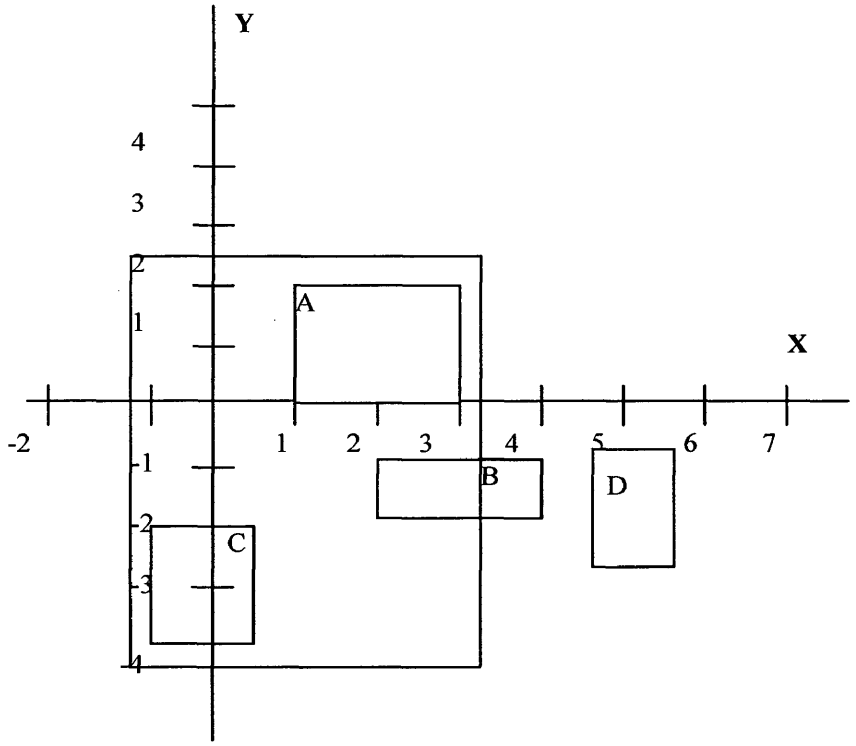


Figure 4.3 graphical representation of spatial data record set and topological queries

The following example shows a method to solve directional operators such as distance compared to, close to, northwest, near in a spatial query. Figure 4.4 is a graphical representation of spatial data records. The search window is represented in the form of dotted lines represents the coordinates to be searched in a given spatial data pool stored in Branch Grafted R-tree.

The distances between the centers of the rectangles A, B and D are 4.6, 7, and 8.2 respectively. If the specified distance is 7 then only A and B meet the criteria of the specified distance. The distances between the search rectangle and A, B are less than or equal to the specified distance to be considered as hit.

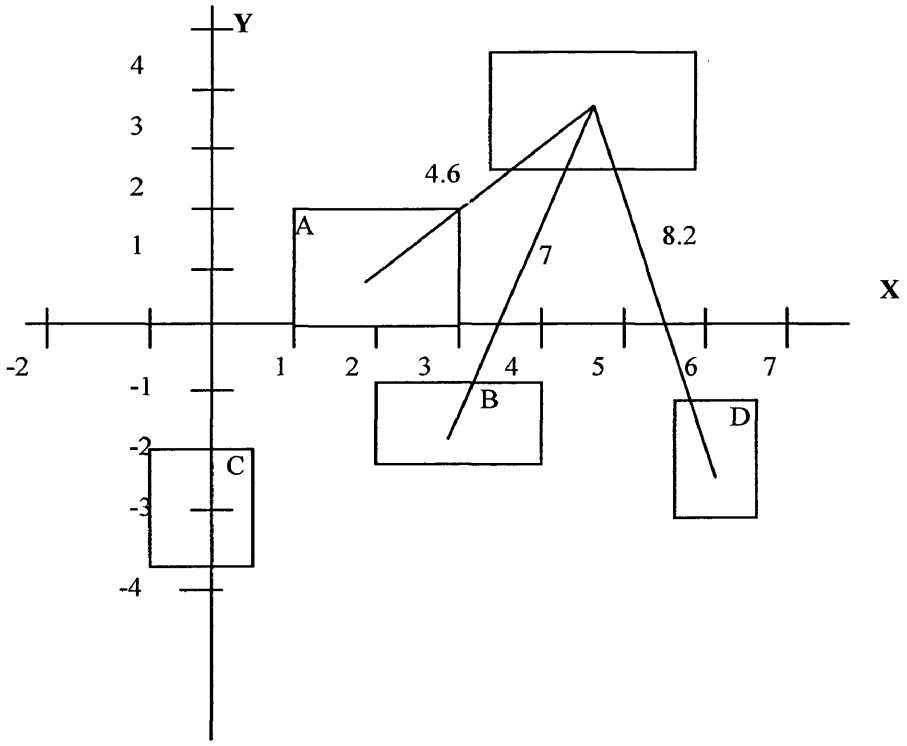


Figure 4.4 graphical representation of spatial data record set and directional queries

4.1.2 Spatial Query Algorithm

The algorithm defined in figure 4.5 reads input queries from the keyboard or input text file. The text files are used in the thesis to process multiple queries in short time. The input queries are SQL like queries and extract task relevant objects or information from the data stored in the Branch Grafted R-tree data structure. In order to extract information from more than one data source e.g. 'water body' and 'city' the Branch Grafted R-trees are built first to store the required data mentioned in the 'FROM' clause of the query. The next step is to analyze the 'WHERE' clause of the query. The 'WHERE' clause of the query contains, conditions and functions figure 3.9 specified by users to retrieve the data.

For example: WHERE CITY_LOCATION CLOSE_TO (WATER_BODY) AND WATER_BODY = 'Lake Michigan'

In the above example the 'WHERE' clause contains the information about the location of the object to be retrieved i.e. close to water body and name of water body as Lake Michigan.

In order to retrieve a spatial object, based upon the conditions mentioned in the 'WHERE' clause of the query, first the coordinates of the spatial objects mentioned in the 'WHERE' clause are determined. The coordinates of the objects mentioned in the 'WHERE' clause act as a search window in the tree. The functions such as 'close_to' etc.

are solved with the help of the search window coordinates. The various functions developed for the queries in the thesis have been discussed in detail in the next section.

Figure 4.5 Algorithm to execute the queries is as follows:

S1 - Build required trees

S2 - Read user defined queries from input files or key boards

S3 - Get coordinates to be searched (search window) from the tree defined in the 'where' clause of the query

S4 - Call function defined in the 'where' clause of the query

S5 - Update the output file with the number of 'hits' returned by the functions

The examples of various functions are as follows:

4.1.2.1 Algorithm for 'close_to' function

```
close_to()
d = 30
if node is not a leaf node
then check each data record of the node to determine
        if the distance of the data record from the
                search object (rectangle) is less than d

        for all records that match call the  routine
                on the tree whose node is pointed to by the child
                pointer of index record
        end for
end if

if the node is a leaf node
        for all data records of the node
                if the distance from the search object (rectangle) is less than
                 d  then return the node
                  hit = hit +1
        end for
end if
```

Calculation of distance

dx → distance along the x direction between
search window and data record
dy → distance along the y direction between
search window and data record

Distance → $Sqrt((dx_1 * dx_2) + (dy_1 * dy_2))$

```
if distance between centers of data record and
        search object  < specified distance
        return hit
else
          calculate all possible combination of distances between the four coordinates of
          bounding rectangle of target
          object stored in Branch Grafted R-tree and four      coordinates of search objects

        store the distances in and array
      for all distances in the array
              find min distance
        end
       if minimum distance between the bounding rectangle of the
              target object and the coordinates of search rectangle < specified distance
              return hit
       else
              return miss
       end if
```

The 'close_to' function finds spatial objects stored in the Branch Grafted R-tree within the specified or at a given distance of the given spatial object. The algorithm searches the tree in top down manner and calculates the distance between the given spatial object (query) and records stored in the leaf nodes of the Branch Grafted R-tree. All possible distances are calculated between the bounding rectangle of given (query) object and the records stored in the leaf nodes of the Branch Grafted R-tree. To minimize calculations first the distance between the centers of the two objects is calculated. If the distance between the centers of the two spatial objects is less than or equal to the given or

specified distance in the query then the object is considered as hit. Otherwise the all-possible distances are calculated between the two objects and if the minimum distance between the two objects is less than or equal to the specified distance then the object is considered as hit.

```
SELECT CITY_NAME, CITY_LOCATION
FROM   CITY, HYDRO
WHERE CITY_LOCATION LOCATION CLOSE_TO (WATER_BODY)
 AND WATER_BODY = 'Lake Michigan'
```

Figure 4.6 An example of spatial query using 'close_to' function

The query in figure 4.6 is designed to select the city name and its location and the city from the database that contains all the cities in North America and located within 60 miles radius of Lake Michigan.

The information related to all the water bodies in North America is stored in the database called HYDRO. There are various functions that govern search such as 'close_to', 'intersects', 'adjacent_to', 'distance_from' (figure 3.1) etc.

The 'close_to' function accepts the water body name and distance as the parameters. It then finds the coordinates of the bounding rectangle of the water body and, using the coordinates to find the cities that are within 60 miles of the water body. The user can use

the query and thus find cities close to a water body and within a desired distance from the water body.

Calculating the distance between the coordinates of the given spatial object and the target node of the tree solves the functions described above. The calculated distance is compared to the specified distance sent to the functions in the form of parameter. To the parameter distance figure a ±5% of tolerance figure is added in some functions in order to consider range of distance.

4.1.2.2 Algorithm for 'adjacent_to' function

```
adjacent_to()
distance_range = 1 to 5
if node is not a leaf node then
    check each data record of the node to determine
        if the distance of the data record from the
                search object (rectangle) is within distance_range

        for all records that match call the routine
                on the tree whose node is pointed to by the child
                pointer of index record
        end for
end if

if the node is a leaf node then
        for all data records of the node
                if the distance from the search object (rectangle) is less than
                 the distance_range then return the node
                 hit = hit +1
        end for
end if
```

The 'adjacent_to' function finds the spatial objects stored in the Branch Grafted R-tree within the specified or a given distance of the given spatial object. The algorithm searches the tree in top down manner and calculates the distance between the given

spatial object (query) and records stored in the leaf nodes of the Branch Grafted R-tree. All possible distances are calculated between the bounding rectangle of given (query) object and the records stored in the leaf nodes of the Branch Grafted R-tree. To minimize calculations first the distance between the centers of the two objects is calculated. If the distance between the centers of the two spatial objects is less than or equal to the given or specified distance in the query then the object is considered as hit. Otherwise all the possible distances are calculated between the two objects and if the minimum distance between the two objects is less than or equal to the specified distance then the object is considered as hit.

```
SELECT CITY_NAME, CITY_LOCATION
FROM   CITY, HYDRO
WHERE CITY_LOCATION ADJACENT_TO (WATER_BODY)
AND    CLOSE_TO    (US_BUNDARY,30)    AND    WATER_BODY    =
'ATLANTIC_OCEAN'
```

Figure 4.7 An example of spatial query that uses 'adjacent_to' function

The ADJACENT_TO function accepts the water body name as the parameters. It then finds the coordinates of the bounding rectangle of the water body and, using the coordinates to find the cities that are within 5 miles of the water body and at 30 miles of distance with us boundary. The user can use the query and thus find cities adjacent to a water body and within a desired distance from the water body.

Calculating the distance between the coordinates of the given spatial object and the target node of the tree solves the functions described above. The calculated distance is compared to the specified distance sent to the functions in the form of parameter. To the parameter distance figure a ±5% of tolerance figure is added in some functions in order to consider range of distance.

4.1.2.3 Algorithm for 'distant_from' function

distant_from( )

```
if node is not a leaf node
then check each data record of the node to determine
        if the distance of the data record from the
                search object (rectangle) is less than specified distance d

        for all records that match call the routine
                on the tree whose node is pointed to by the child
                pointer of index record
        end for
end if

if the node is a leaf node
        for all data records of the node
                if the distance from the search object (rectangle) is less than
                  the specified distance d then return the node
                  hit = hit +1
        end for
end if
```

The 'distant_from' function finds the spatial objects stored in the Branch Grafted R-tree within the specified or a given distance of the given spatial object. The algorithm searches the tree in top down manner and calculates the distance between the given spatial object (query) and records stored in the leaf nodes of the Branch Grafted R-tree. All possible distances are calculated between the bounding rectangle of given (query) object and the records stored in the leaf nodes of the Branch Grafted R-tree. To minimize

calculations first the distance between the centers of the two objects is calculated. If the distance between the centers of the two spatial objects is less than or equal to the given or specified distance in the query then the object is considered as hit. Otherwise the all - possible distances are calculated between the two objects and if the minimum distance between the two objects is less than or equal to the specified distance then the object is considered as hit.

```
SELECT CITY_NAME, CITY_LOCATION
FROM    CITY, HYDRO
WHERE CITY_LOCATION DISTANT_FROM (WATER_BODY,50)
AND CLOSE_TO (US_BOUNDARY, 20) AND WATER_BODY = 'PACIFIC OCEAN'
```

Figure 4.8 an example of spatial query using 'distant_from' function

The 'distant_from' function accepts the water body name as the parameters. It then finds the coordinates of the water body, using the coordinates to find the cities that are at a 50 miles of distance with Pacific Ocean and 20 miles of distance with US boundary. The user can use the query and thus find cities that are distant from a given object.

Calculating the distance between the coordinates of the given spatial object and the target node of the tree solves the functions described above. The calculated distance is compared to the specified distance sent to the functions in the form of parameter. In the

calculated distance a tolerance of ±5 % has been added, as the objective is to find intersecting objects.

4.1.2.4 Algorithm for 'intersects' function

intersects( )

```
if node is not a leaf node
then check each data record of the node to determine
        if the distance of the data record from the
                search object (rectangle) is less than d

        for all records that match call the  routine
                on the tree whose node is pointed to by the child
                pointer of index record
        end for
end if

if the node is a leaf node
        for all data records of the node
                if  x1 or  y1 coordinate of search object rectangle are greater than
                     x2 or y2 coordinates of target data object rectangle or
                 x2, or y2 coordinated of search object rectangle are less than
                  x1 or y1 coordinates of target data object rectangle
                        return miss
                else
                        return hit = hit +1
        end for
end if
```

The 'intersects' function finds the spatial objects stored in the Branch Grafted R-tree within the specified or a given distance of the given spatial object. The algorithm searches the tree in top down manner and calculates the distance between the given spatial object (query) and records stored in the leaf nodes of the Branch Grafted R-tree. In order to find whether the bounding rectangles of two objects intersect the coordinates are tested. If x1 or y1 coordinate of search object rectangle are greater than x2 or y2 coordinates of target data object rectangle or x2, or y2 coordinated of search object rectangle are less than x1 or y1 coordinates of target data object rectangle then object are

not considered to be intersecting objects. Otherwise in all other conditions the objects are considered to be intersecting objects (hit).

```
SELECT CITY_NAME, CITY_LOCATION
FROM   CITY, HYDRO
WHERE CITY_LOCATION INTERSECTS (MAJOR_US_HIGHWAY)
 AND  CLOSE_TO  (WATER_BODY,  30)  AND  WATER_BODY  =  'LAKE
MICHIGAN' AND MAJOR_US_HIGHWAY = 'I-90'
```

Figure 4.9 an example of spatial query using 'intersects' function

The 'intersects' function accepts the major us highway name as the parameters. It then finds the coordinates of the highway, using the coordinates to find the cities that are at a zero miles of distance and of the water body and at 30 miles of distance. The user can use the query and thus find cities that intersect a given object.

Calculating the distance between the coordinates of the given spatial object and the target node of the tree solves the functions described above. The calculated distance is compared to the specified distance sent to the functions in the form of parameter. In the calculated distance no tolerance is added, as the objective is to find intersecting objects.

4.1.2.5 Algorithm for 'within' function

within ( )

```
if node is not a leaf node
then check each data record of the node to determine
        if the distance of the data record from the
                search object (rectangle) is less than d

        for all records that match call the routine
                on the tree whose node is pointed to by the child
                pointer of index record
        end for
end if

if the node is a leaf node
        for all data records of the node
                if  x1 and y1 coordinate of search object rectangle are less than
                      x1 and y1 coordinates of target data object rectangle
                    and x2 and y2 coordinated of search object rectangle are greater than
                       x1 and y1 coordinates of target data object rectangle
                 then return node
                     hit = hit +1
        end for
end if
```

The 'within' function finds the spatial objects stored in the Branch Grafted R-tree within a specified or a given distance of the given spatial object. The algorithm searches the tree in top down manner and calculates the distance between the given spatial object (query) and records stored in the leaf nodes of the Branch Grafted R-tree. In order to find whether the bounding rectangles of target or data object is within the search object or window the coordinates are compared. If x1 and y1 coordinates of the search objects are less than the x1 and y1 coordinates of the target or the data object and x2, or y2 coordinates of search object rectangle are greater than x2 and y2 coordinates of target data object rectangle then data object is considered to be within the search object window (hit).

4.2 Mining Spatial Rules

The output data of the queries is stored in a text file. The text file contains a variety of the result sets from different queries. This result set cannot be directly used to mine spatial rules. The grouping of the data is necessary to find the predicates. Spatial predicates are the topological relations like (close_to, water). The predicates determination is the first step in mining of the spatial rules. The algorithm described in figure 4.5 computes large predicates. For example for each row of the output file resulted from queries (i.e. each large town), if the water attribute is non empty the count of water is incremented by one such a count contributes to 1-predicate rows. The program also computes the support counts for each predicate set. If the support count is less than the minimum threshold the row is not considered for further analysis. The 2-predicate rows are formed by the pair-wise combination of the large 1-predicates with their count. In the thesis only 1 and 2-predicate sets are considered. The example of 2-predicate sets is (adjacent_to, water), (close_to, us_boundary).

The algorithm, developed in the thesis work, to determine predicates and support count reads the results of queries and stores them in linked list. The records are categorized into different categories such as size (big, small), type of the spatial object (e.g. 'water body', 'us boundary', 'gas station') etc. If each categorized attribute of the record matches with the attributes of a record stored in the sample node of the linked list, the support count of

the record stored in the linked list is incremented by one. Otherwise a new node is created with support count equal to one. In this way all the data obtained as a result of queries is grouped in the form of predicates with their support count.

For example the output of a query: CITY B CLOSE_TO MINNESOTA RIVER B indicates that there is a big city close to big Minnesota River. This record when processed to determine the predicate then the result of predicate is (close_to, big_water_body)

Figure 4.10 algorithm for mining spatial rules is as follows:

n → existing node of linked list
r → record from output file

S1 - Read output file (result of queries) records
S2 - **for** each output record **r** of output file
if first node of linked list **the**
head = create_node(**r**)
else
for each node **n** of the linked list
if (data stored in **n** = **r**) **then**
n->support_count = n->support_count + 1? increment support count
else
tail = create_node (**r**)
n->.support_count = n ->support_count + 1
S3 - mine rules

4.2.1 Spatial Association Rule

In order to mine the spatial association rules described in the section 3.4.1 first the predicates need to be determined. Spatial predicates are the topological relations like (close_to, water). The result data set of the queries is first analyzed to determine predicates and support for each predicate. Once all the predicates and support for each predicate are determined then this result set is analyzed for mining association rule.

The method used in the thesis is based upon the calculation of confidence of the predicates. The confidence (percentage) of a rule is calculated with the help of support count of predicates. The following is an example of extraction of association rule.

Predicates	**Support Count**
CITY (CLOSE_TO, water)	11
CITY (CLOSE_TO, 'US boundary'), (CLOSE_TO, water)	6

In the above example the association rule is extracted with the help of confidence (%) calculation

(6/11) * 100 = 54%

The mined spatial **association rule** is:

CITY ^ CLOSE_TO (water_body) → CLOSE_TO 'US boundary' (54%).

The above mined rule means that 54% of cities in US close to water body are close US boundary.

The algorithm developed to mine association rule in the thesis (figure 3.14) takes the grouped data records (predicates) stored in the linked list as input. For each node of the linked list that contains 1-predicate set all the nodes with 2-predicate sets are compared. If the 1-predicate set and its attributes (big, small) matches with one of the 2-predicate set and attributes then the confidence is calculated. Confidence of rule is calculated by the fraction of the support counts of 1 and 2-predicate sets.

Figure 4.11 algorithm for mining association rule is as follows:

mine_association_rule()

S1 - **for** each predicate set stored in the node **n1** of list **L1** containing 1- predicate set

Traverse the list to find nodes that contain 2-predicate sets

Compare the predicate set from **L1** stored in **n1** and predicate sets stored in **n2**

If 1-predicate set (n1) matches with one of the 2-predicate sets (n2) **then**

? Calculate Confidence **C**

C =(support count of 2-predicate set / support count of 1-predicate set) * 100

Update the output list **L2**

S2 - Update output file with the help of output linked list **L2**

4.2.2 Spatial Aggregation Rule

In order to mine the spatial association rules described in the section 3.4.3. The result data set of the queries is the first analyzed to determine predicates and support count for each predicates. Once the predicates and support count for each predicate are determined then this result set is analyzed for mining aggregation rule. The method used in the thesis is based upon the counting of the support count of the predicates. One example of mining of aggregation rule is to find the total number of big cities in North America close to big water bodies.

The predicates are determined with the help of the algorithm described in the figure 3.15 and stored in the linked list. For each predicate-set in the node of the linked list the algorithm searches the complete linked list in order to find predicates having same attributes in 1-predicate and 2-predicate sets. The support counts of the matching predicates are added to mine aggregation rule.

	Support Count
CITY B CLOSE_TO water_body B	11
CITY B CLOSE_TO water_body B CLOSE_TO 'US boundary'	4

B → Big size of city and water body

In the above example the aggregation rule is extracted by adding the support counts of the big cities close to big water bodies. Both the records have at least one predicate set in common along with the size attribute.

The mined spatial **aggregation rule** is:

Total number of big cities in North America located close to big water bodies are 11+ 4 = 15

The algorithm developed to mine association rule in the thesis (figure 3.4) processes the grouped data records (predicates) stored in the linked list. For each node of the linked list that contains 1-predicate set remaining nodes of the linked list are compared. If a predicate set and its attributes (e.g. big, small) match with other sets of predicates and attributes then their support counts are added for mining aggregation rule.

The algorithm for mining aggregation rule is as follows:

Figure 4.12 algorithm for 'mine_aggregation_rule'

S1 - For each predicate set stored in the node **n1** of list **L** containing k - predicate sets

Compare the predicate sets and attributes of **n1** and **n2**

If n2 contains 2-predicate sets **then**

Compare the predicate sets of n2 with n1

If matching predicate is found **then**

Total_count = n1. Support_count +n2. Support_count

Update output list

S2 - Update output file

4.2.3 Spatial Discriminant Rule

Mining of spatial discriminant rules described in the section 3.4.4 starts with the analysis of data set obtained as a result of spatial queries. The purpose of the analysis is to determine predicates and calculate support count for each predicate. Once the predicate sets and support count for each predicate set are determined, the result set is then further analyzed for mining of aggregation rule. The method used in the thesis is based upon the grouping of similar predicates. This method is similar to the mining of aggregation rules method and there is no need of a separate algorithm for the mining of discriminant rule. The user can mine discriminant rule with the help of the grouped predicates to find contrasting groups.

One example of mining of discriminant rule is that 'big cities are located close to big water bodies while smaller cities are located close to small water bodies in North America'.

The predicates are determined with the help of the algorithm described in the figure 3.16 and stored in the linked list. For each predicated set in the node of the linked list the algorithm searches the complete linked list in order to find predicates having same attributes in 1-predicate and 2-predicate sets. The support count of two predicate sets are added and stored in another. This results in a new list containing the grouped predicates with the total support counts. The resulting output is self-explanatory and two contrasting predicate groups result in discriminant rule.

Figure 4.13 algorithm for mining discriminant rule

S1 - For each predicate set stored in the node **n1** of list **L** containing k - predicate sets
Search the list
Compare the predicate sets and attributes of two nodes **n1** and **n2**
If n2 contains 2-predicate sets **then**
Compare the predicate sets of the n2 with n1
If matching predicate is found **then**
Total_count = n1. Support_count+ n2. Support_count
Update output list
end if
end if

4.3 Computational Complexity of Algorithms

Figure 4.14 summaries the three modules of the thesis and the complexity associated with these modules.

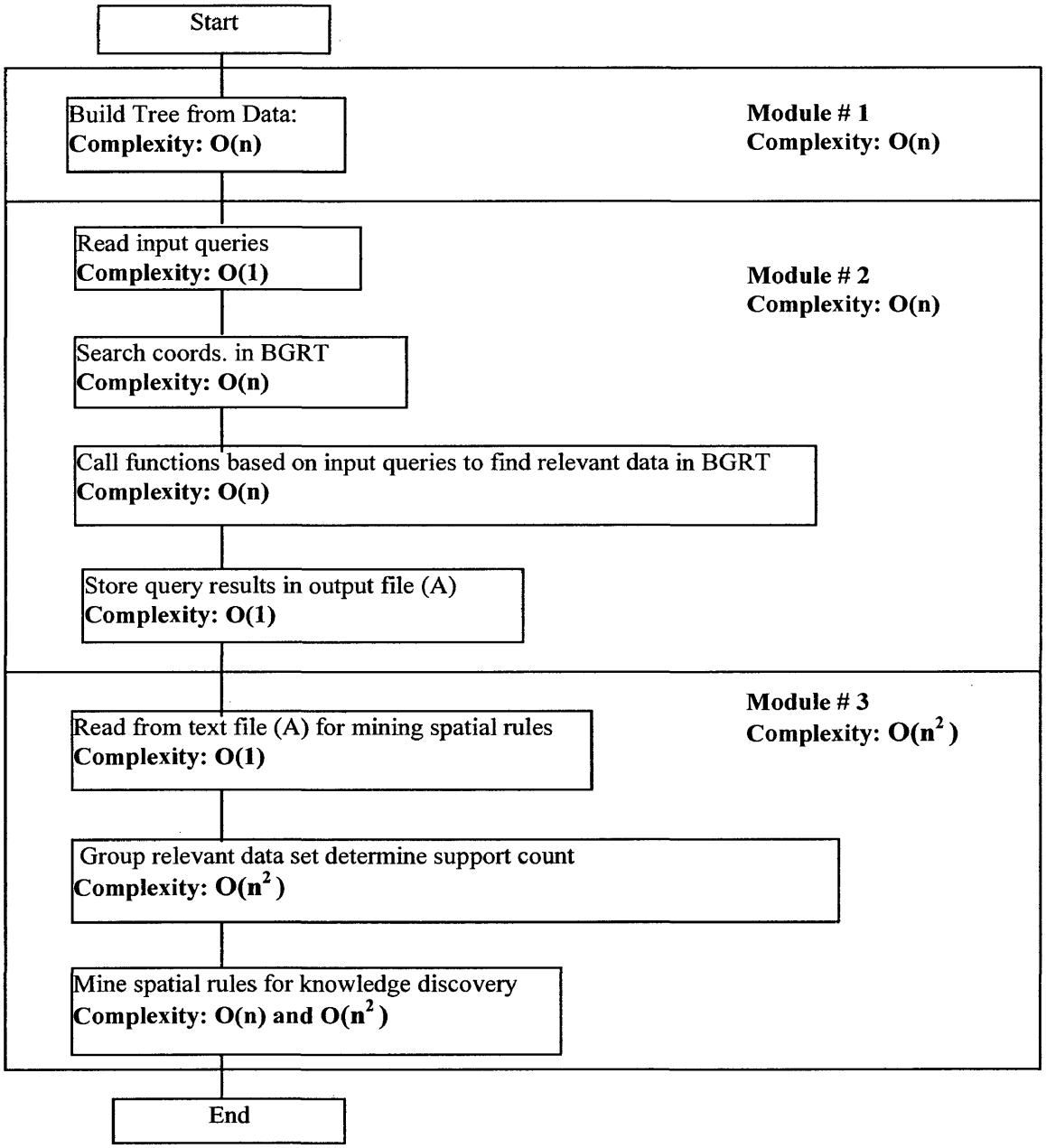


Figure 4.14 Computational Complexity of the algorithms in the thesis

4.4 Chapter Summary

This chapter first lays out the different modules involved in the program structure. The three modules described include: Build the Branch Grafted R- Tree, Retrieving data from the Branch Grafted R-Tree using queries, and mining spatial rules. Each module's function has been described briefly. This is followed by spatial query design for data retrieval, format of the spatial queries, algorithm for spatial queries, and different functions used by SQL like spatial queries. Different algorithms that are used for analysis of the relevant output of the queries to mine spatial rules such as spatial association rules, aggregation rule, and discriminant rule have been discussed. The chapter ends with the summary of complexity analysis of the different algorithms used in the thesis.

The next chapter uses actual data and simulates the above-described steps. These steps are illustrated in detail.

Chapter 5 -Program Testing and Result Analysis

The previous chapter contains the description of the modules of the programs and their objectives. The chapter contains the description of spatial queries developed in the thesis for the retrieval of data stored in the Branch Grafted R-tree. The methods to retrieve spatial data and mining of spatial rules have been discussed at the algorithmic level. Next, the implementation details of the spatial queries and spatial data mining methods used in the thesis are discussed.

This chapter starts with the description and example of spatial data used in the thesis. The description of the spatial data is followed by spatial query design and method to implement the query used in the thesis with an example. The output of the queries has been shown in the form of a text file. The output is followed by the analysis of the output to mine rules.

5.1 Description of Spatial Data

The spatial data for the work has been down loaded from TIGER/LINE. The TIGER database contains spatial information about hydrology, transportation, and other objects like national and state parks, churches, universities etc. The data used in this work deals water bodies in the North America, cities of North America, and US highways. More information about cities by counties was obtained from LANDVIEW3 site. This data

contains the spatial and detailed non-spatial information as, census, industries, income level, etc. The raw data from different counties and states was rearranged and merged to make a master data file. The data format is as follows:

minimum x coordinate, minimum y coordinate, positive delta x, positive delta y, name

```
-121.950938        37.685388          0.008408     0.005066     TOWN_14
-121.690552        37.591956          0.001539     0.001361     TOWN_15
-121.689014        37.591354          0.000001     0.000602     TOWN_16
-121.689014        37.585831          0.001264     0.005523     TOWN_17
-121.68775         37.585831          0.001245     0.000007     TOWN_18
-121.686505        37.585838          0.003179     0.000051     TOWN_19
-121.683326        37.585883          0.001179     0.000006     TOWN_20
-121.680127        37.583727          0.000001     0.000522     TOWN_21
-121.680126        37.576767          0.000011     0.00696      TOWN_22
-121.680115        37.572106          0.000007     0.004661     TOWN_23
-121.687508        37.564131          0.0074       0.007975     TOWN_24
-121.688781        37.564122          0.001273     0.000009     TOWN_25
-121.689319        37.564122          0.000538     0.00121      TOWN_26
-121.691202        37.565332          0.001883     0.00135      TOWN_27
-121.694097        37.566682          0.002895     0.002076     TOWN_28
-121.699571        37.569861          0.003951     0.002855     TOWN_29
-121.716276        37.577567          0.009862     0.008694     TOWN_30
-121.733522        37.595505          0.003644     0.003436     TOWN_31
-121.690948        37.593306          0.000396     0.000011     TOWN_32
-121.706282        37.593072          0.015334     0.000234     TOWN_33
-121.708758        37.597509          0.001437     0.002799     TOWN_34
```

Figure 5.1 Spatial data format

Data Set Name	Data Set Type	Records
rail_tiger95	Real – railroad spatial data	31,059
hydro_tiger95	Real – hydrographic spatial data	360,330
town_tiger95	Real – town borders spatial data	234,251
rail_river	Real – railroad river spatial data	128,971
street	Real – street spatial data	131,461

Figure 5.2 Data set used in the thesis

5.2 Implementation of Queries

Two different trees are built, namely, 'Hydro' and 'Town' to store data related to the above query. The program first searches the name of the water body in the data stored in the Branch Grafted R-tree returning the coordinates of the bounding rectangle of the water body. Then with the help of the coordinates thus obtained, the instance of the Branch Grafted R-tree that stores CITYT DATA is searched with the help of the window search algorithm. The Branch Grafted R-tree is traversed in top-down manner to find the leaf nodes that satisfy the given condition in the query. The conditions could be the distance between the coordinates of the bounding rectangle stored in the leaf node and the coordinates of the water body. If the leaf node satisfies the given condition then it is considered to be a 'hit'. The number of such 'hits' are counted and returned to the calling program.

The input queries are in the form of a text file, the program reads to process multiple queries at once at a faster rate. The program has also been designed to take the input from keyboard.

The important queries used are listed below:

CITY	CLOSE_TO	MISSISSIPPIRIVER	20
CITY	CLOSE_TO	MISSISSIPPI(INTERNMENT)	20
CITY	CLOSE_TO	TR-MINNESOTA	30
CITY	CLOSE_TO	CANNON	I0
CITY	ADJACENT_TO	TR-MINNESOTA RIVER	0
CITY	ADJACENT_TO	VERMILLION RIVER	0
CITY	ADJACENT_TO	OFFSTREAM	0
CITY	DISTANT_FROM	LAKE ONTARIO	20
CITY	DISTANT_FROM	MISSOURIE RIVER	15
CITY	DISTANT_FROM	CHICAGO RIVER	15
GAS_S	CLOSE_TO	I-80	5
GAS_S	CLOSE_TO	I-94	5
GAS_S	CLOSE_TO	I-35W	5
CITY	INTERSECTS	I-80	I5
CITY	INTERSECTS	I-94	10
CITY	INTERSECTS	I-35W	20

CITY	CLOSE_TO	MISSISSIPPIRIVER	20	CLOSE_TO	US_BOUNDARY	20
CITY	CLOSE_TO	TR-MINNESOTA	30	INTERSECTS	I-35W	30
CITY	CLOSE_TO	CANNON	I0	CLOSE_TO	ST.PAUL	10
CITY	ADJACENT_TO	TR-MINNESOTA RIVER	0	CLOSE_TO	ST PAUL	I0
CITY	ADJACENT_TO	VERMILLION RIVER	0	CLOSE_TO	ST PAUL	10
CITY	DISTANT_FROM	LAKE ONTARIO	20	DISTANT_FROMUS_BOUNDARY		20
CITY	DISTANT_FROM	MISSOURIE RIVER	15	DISTANT_FROMUS_BOUNDARY		I5
CITY	CLOSE_TO	ATLANTIC OCEAN	20	CLOSE_TO	US_BOUNDARY	20

Figure 5.3 Combination of such queries was used to mine rules from the result data set obtained as a result of the above queries.

5.3 Output Files Obtained By Running Queries

The results of the above queries are stored in an output file with the help of a C program. An example of the output, in reference to the above listed query, is in the form listed below:

CITY CLOSE_TO Lake Michigan X, where X is the number of hits

The above suggests that X number of cities have been located in the spatial database within 30 miles of the water body 'lake Michigan'.

Such output obtained is a result of several queries performed on the spatial data with the intent of knowledge discovery. The queries designed have thus helped in mining the database and in reaching a higher refinement level. Although conclusions cannot be drawn at this stage, meaningful insight has certainly been obtained. The information obtained at this stage requires further processing. Further mining and analysis is then carried out.

Figure 5.4 Example of output file:

object1	size1	function 1	object 2	object type2	size 2	distance1
CITY	B	CLOSE TO	MISSISSIPPIRIVER	W	B	20
CITY	B	CLOSE_TO	MISSISSIPPI (INTERNMENT)	W	B	6
CITY	B	CLOSE_TO	TR-MINNESOTA	W	S	3
CITY	S	CLOSE_TO	CANNON	W	B	2
CITY	B	ADJACENT_TO	TR-MINNESOTA RIVER	W	S	3
CITY	S	ADJACENT_TO	VERMILLION RIVER	W	S	2
CITY	S	ADJACENT_TO	OFFSTREAM	W	S	1
CITY	B	ADJACENT_TO	TR NEWPORT BAY	W	S	2
CITY	B	ADJACENT_TO	ATLANTIC OCEAN	W	B	15
CITY	S	ADJACENT_TO	LAKE ERIE	W	B	10
CITY	B	ADJACENT_TO	PACIFIC OCEAN	W	B	35
CITY	B	DISTANT_FROM	LAKE ONTARIO	W	B	15
CITY	S	DISTANT_FROM	MISSOURIE RIVER	W	B	16
CITY	S	DISTANT_FROM	GREAT EGG HARBOR RIVER	W	S	10
CITY	S	DISTANT_FROM	TUCKAHOE RIVER	W	S	11
CITY	S	DISTANT_FROM	LAKE HURON	W	B	22
CITY	B	DISTANT_FROM	CHICAGO RIVER	W	S	10
CITY	B	CLOSE_TO	GULF OF MEXICO	W	B	17
CITY	B	CLOSE_TO	ATLANTIC OCEAN	W	B	50
GAS_S	B	CLOSE_TO	I-80	H	B	32
GAS_S	S	CLOSE_TO	I-94	H	B	14
GAS_S	S	CLOSE_TO	I-35W	H	S	9
GAS_S	S	CLOSE_TO	I-494	H	B	5
GAS_S	S	CLOSE_TO	I-680	H	B	6
CITY	B	INTERSECTS	I-80	H	B	15
CITY	S	INTERSECTS	I-94	H	B	6
CITY	S	INTERSECTS	I-35W	H	B	9
CITY	B	CLOSE_TO	MISSISSIPPIRIVER B W CLOSE_TO		US_BOUNDARY	11
CITY	B	CLOSE_TO	MISSISSIPPI(INTERNMENT)BCLOSE_TO		US_BOUNDARY	0
CITY	S	CLOSE_TO	TR-MINNESOTA B W INTERSECTS		I-35W	5

CITY	S	CLOSE_TO	CANNON	S	W	CLOSE_TO	ST.PAUL	3
CITY	S	ADJACENT_TO	TR-MINNESOTA RIVER	B	W	CLOSE_TO	ST PAUL	5
CITY	S	ADJACENT_TO	VERMILLION RIVER	S	W	CLOSE_TO	ST PAUL	
CITY	S	ADJACENT_TO	OFFSTREAM	S		CLOSE_TO	ST PAUL	2
CITY	S	ADJACENT_TO	TR NEWPORT BAY	S		CLOSE_TO	US_BOUNDARY	3
CITY	B	ADJACENT_TO	ATLANTIC OCEAN	B		CLOSE_TO	US_BOUNDARY	25
CITY	S	ADJACENT_TO	LAKE ERIE	B		CLOSE_TO	US_BOUNDARY	7
CITY	B	ADJACENT_TO	PACIFIC OCEAN	B		CLOSE_TO	US_BOUNDARY	20
CITY	B	DISTANT_FROM	LAKE ONTARIO	B		DISTANT_FROM	US_BOUNDARY	10
CITY	B	DISTANT_FROM	MISSOURIE RIVER	S		DISTANT_FROM	US_BOUNDARY	0
CITY	S	DISTANT_FROM	GREAT EGG HARBOR RIVER	S		DISTANT_FROM	US_BOUNDARY	10
CITY	S	DISTANT_FROM	TUCKAHOE RIVER	S		DISTANT_FROM	US_BOUNDARY	5
CITY	B	DISTANT_FROM	LAKE HURON	S		DISTANT_FROM	US_BOUNDARY	6
CITY	B	DISTANT_FROM	CHICAGO RIVER	S		DISTANT_FROM	US_BOUNDARY	2
CITY	B	CLOSE_TO	GULF OF MEXICO	B		CLOSE_TO	US_BOUNDARY	15
CITY	B	CLOSE_TO	ATLANTIC OCEAN	B		CLOSE_TO	US_BOUNDARY	17
GAS_S	B	CLOSE_TO	I-80	B		INSIDE	COLORADO	5
GAS_S	B	CLOSE_TO	I-94	B		INSIDE	MINNESOTA	4
GAS_S	S	CLOSE_TO	I-35W	S		OUTSEDE	MINNESOTA	7
GAS_S	B	CLOSE_TO	I-494	S		OUTSIDE	MINNESOTA	2
GAS_S	S	CLOSE_TO	I-680	S		CLOSE_TO	NEBRASKA	3
CITY	B	INTERSECTS	I-80	B		CLOSE_TO	MISSOURIE RIVER	3
CITY	B	INTERSECTS	I-94	B		CLOSE_TO	MISSISIPI RIVER	5
CITY	B	INTERSECTS	I-35W	S		CLOSE_TO	MISSISIPI RIVER	6

5.4 Analysis of Output for Mining Rules

In order to analyze the results obtained above a C program was written. The program reads the output file generated by the execution of the queries and processes it to mine rules. The program categorizes the result sets based on the functions and given condition parameters.

The algorithm in figure 4.5 for mining spatial rules reads the output record one by one and makes a linked list. If the record is already present in the linked list then the record counter is incremented by one otherwise a new node is added to the linked list. The objective of the algorithms is to categorize the result obtained by query processing. The criteria for categorizations are the type of spatial object (W -> water, H->highway, U-> US boundary including US Canada, US Mexico boundary and shore lines), size (B-> Big, S-> Small), and function (close_to, intersects, distant_from, adjacent_to, within, inside etc.).

Once the final linked list is formed then the linked list is traversed to mine spatial rules.

For example:

CITY B CLOSE_TO B 'lake Michigan' 5

CITY B CLOSE_TO B 'Atlantic ocean' 6

CITY B ADJACENT_TO B 'Mississippi' river 8

CITY S ADJACENT_TO S 'Off streams' 13

CITY B CLOSE_TO 'Atlantic Ocean' B CLOSE_TO 'US boundary' 4

The programs categorizes the above example data as follows and stores it in a linked list:

CITY CLOSE_TO water 5+6 = 11

CITY ADJACENT_TO water 21

CITY CLOSE_TO 'US boundary' and water 4

For mining a rule the program now traverses the linked list and performs the following calculation:

(4/11) * 100 = 36.36%

The mined spatial **association rule** would then be:

CITY ^ CLOSE_TO (big_water_body)→ CLOSE_TO 'US boundary' (36.36%).

The above mined rule means that 36.36% of cities in US close to water body are close_to US boundary.

The mined spatial **aggregation rule** mine from the above example data is:

CITY CLOSE_TO (big_water_body) = 5 + 6+ 4 = 15

Total 15 big cities in North America are close to big water body

The spatial **discriminant rule** mined from the given example data is:

Big CITY ADJACENT_TO (big_water_body) while Small CITY ADJACENT_TO (small water_body)

The above-mentioned result is new information (knowledge) that has been generated by using the current information (present in the database), performed specific queries, and by carrying out analysis on the results of the query. The new information thus obtained is now available for mining application, as for natural resource management, city planning, or by real estate industry.

5.5 Mining Spatial Rules

Following are some spatial rules discovered by the implemented programs:

Spatial Association Rules

- is_a (big_town) ^ close_to (big_water_body) → close_to (us_boundary): 70%
- is_a (small_town) ^ close_to (big_water_body) → close_to (us_boundary): 85%
- is_a (big_town) ^ close_to (small_water_body) → close_to (us_boundary): 32%
- is_a (small_town) ^ close_to (small_water_body) → close_to (us_boundary): 40%
- is_a (small_town) ^ adjacent_to(big_water_body) → adjacent_to (us_boundary): 78%
- is_a (big_town) ^ adjacent_to(big_water_body) → adjacent_to (us_boundary): 67%
- is_a (big_town) ^ close_to(big_water_body) → intersects(major_highway): 97%
- is_a(small_town)^close_to(small_water_body)→ intersects(major_us_highway): 56%

Spatial Aggregation Rules

- Total number of big cities located close_to(big_water_bodies) in North America: 43
- Total number of gas stations located close_to (major_us_highways) in NY: 107
- Total number of small cities close_to(big_water_bodies) in North America: 134
- Total number of big cities located close_to(us_boundary) in North America: 32
- Total number of small cities located close_to (us_boundary) in North America: 59

Spatial Discriminant Rule

- Large number of gas stations are located close_to (major_us_highways) while small number of gas stations are located close_to(small_streets)
- Big cities are located close_to(big_water_bodies) while small cities located close_to(small_water_bodies)
- Large number of industries are located within(big_cities) while small number of cities located within(small_cities)

5.6 Chapter Summary

This chapter first describes the format, source and preparation of the spatial data used in the thesis. The description of the data is followed by implementation and examples of input files of the spatial queries, designed to retrieve relevant information stored in the data structure. The output file resulting from execution of spatial queries is presented. This is followed by the analysis of the output file to mine spatial rules. The three types of spatial rules viz. spatial association rule, spatial aggregation rule, and spatial discriminant rules are then described. The chapter describes examples to elaborate upon the analysis process in order to mine spatial rules.

The concluding remarks and scope of the future work have been described in the next chapter.

Chapter 6 - Conclusions

This thesis provides a method to mine several spatial data mining rules from the pool of spatial data stored in the Branch Grafted R-tree. The work began with the discussion of issues related to spatial data mining because of their direct relevance to the thesis. Desired characteristics of search algorithms, relevance of efficient data structure, and the relevance of data security were discussed.

The spatial data used for the thesis was described along with the source of the data. The spatial query design implementation was then described. Query design is a key process for spatial data mining. Queries were developed as a combination of functions (written in C language). The queries were designed to be run on the spatial data and to extract information that could be used for further processing in the data mining process.

The output files that were obtained as a result of the above queries were then described. A program (in C language) was written to process the output files in order to mine rules. The significance of the mined rules was then discussed.

The process discussed in the work can be used to mine spatial data efficiently and effectively, using efficient Branch Grafted R-tree data structure. The work was performed on real data. The size of the data was approximately 17,000 records per file.

Three such files (CITY, HYDRO, US_HIGHWAY) were used. Spatial data used in real life would be of much larger size.

The key issues with the spatial data mining in general are fast data retrieval and easy to comprehend representation of results. There has always been a need of an efficient data mining method with fast data retrieval. The Branch Grafted R-tree used for the organization and storage of spatial data in the thesis has been proven to be an improved and efficient data structure [2]. The grafting of nodes results in reduced tree height and helps in faster data retrieval i.e. better performance of Branch Grafted R-tree over R-tree. The data stored in the leaf nodes of tree is retrieved with the help of search methods or functions to find nodes within a certain distance with a given node.

The data mining process developed in the thesis is initiated by user's query. Several functions and search algorithms developed in the thesis have been used to solve queries that retrieve data stored in the Branch Grafted R-tree. The data retrieved as a result of several queries is analyzed to mine spatial rules viz. spatial association rule, discriminant rule, aggregation rule, and classification rule. The results of the mining process are easy to comprehend and represented in the form of spatial rules. The data mining method developed in the thesis can be used by end-users for mining spatial rules. The method developed in the thesis is a complete data mining method in itself and thus provides convenience to the end user.

6.1 Future Work

The previous works related to Branch Grafted R-tree [2] and [3] were simulations with small size spatial data sets. This thesis work also is a simulation with small size data. For the large sets of real data the physical implementation and disk access will be major issues. The small data sets are stored using the Branch Grafted R-tee in memory. The implementation of the thesis work in real world scenario with large data sets would require large memory space. There would be a need to store data on the hard disk in real world scenario. Only the part of the data should be stored in the data structure temporarily to avoid the issues related to frequent disk access. There is a potential of designing a user-friendly graphical interface. A good graphical interface helps users to input the query effectively. The graphical representation of result data set will help an end user to understand the result easily. The thesis can be extended by improvement in the usage of Branch Grafted R-tree for data storage. The data structure implemented in the thesis can be used to design, build, and physically implement relational databases. Data can be more efficiently stored in and easily queried from the relational databases. The query retrieval can be improved with the help of an efficient query builder on top of a relational database schema.

6.2 References:

[1] B. Asato, 'Branch-Grafting Heuristic for R-tree Implementation.' Working Paper, Department of Computer Science University of Nebraska at Omaha (1994)

[2] T. M. Schreck, 'An Implementation Analysis of the Branch Grafting and R*-Treee Algorithms.' Thesis, Department of Computer Science University of Nebraska at Omaha, May 1999

[3] M. Khanijo, 'An Implementation and Performance Analysis of Spatial Data Access Methods for the Branch Grafted R-tree Algorithm' Thesis Equivalent Project, Department of Computer Science University of Nebraska at Omaha, fall 1999

[4] N. Bckmann, H. Kriegel, R. Schneider, and B. Seeger, The R*-tree: An Efficient and Robust Access Method for Points and Rectangles ACM SIGMOND, May (1990), pages 322-331

[5] A. Guttman, R-trees: A Dynamic Index Structure for Spatial Searching, Proceedings of the 1984 ACM-SIGMOND Conference on Management of Data (1984), pages 47-57.

[6] K. Kopcrski, J. Han, and J. Adhikary, Mining Knowledge in Geographical Data, *IEEE Computer*, 1998

[7] K. Koperski, J. Han, and J. Adhikary, Spatial Data Mining: Progress and Challenges, 1996 SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'96), Montreal, Canada, June (1996)

[8] N. Roussopoulos, S. Kelley and F. Vincent, Nearest Neighbor Queries, in Proceedings of the ACM-SIGMOD International Conference on the Management of Data, San Jose, USA, pp. 507 – 518, Sept. 1995

[9] K. Koperski, and J. Han, Discovery of Spatial Association Rules in Geographic Information Databases, 1995

[10] T. Sellis, N. Roussopoulos, C. Faloutsos, The R+ tree: A dynamic index for Multi-Dimensional Objects VLDB 1987 pages 507 – 518

[11] U. Fayyad and P. Smyth Image Database Exploration, Progress and Challenges. In Proc. 1993 Knowledge Discovery in Databases Workshop, pp. 14-27, Washington D.C. July 1993

[12] E. Knorr and R.T. Ng. Applying Computational Geometry Concepts to Discovering Spatial Aggregate Proximity Relationships, in Technical Report, University of British Columbia, 1995

[13] R. Ng and J. Han Efficient and effective clustering method for spatial data mining, in Proc. 1994 Int. Conf. Very Large Databases, pp. 145-155 Santiago Chile, September 1994

[14] P. Smyth, M.C. Burl, U.M. Fayyad, and P. Perona. Knowledge Discovery on Large Image Databases: Dealing with Uncertainties in Ground Truth. In Proc. of AAAI-94 workshop on KDD-95, pp.109-120, Seattle WA, July 1994

[15] K. Koperski, J. Han, and N. Stefanovic, An Efficient Two Step Method for Classification of Spatial Data, *Proc. 1998 International Symposium on Spatial Data Handling SDH'98,* Vancouver, BC, Canada, July 1998, pp. 45-54

[16] S. Fotheringham and P. Rogerson, Spatial Analysis and GIS, Taylor and Francis, 1994

[17] J. Han and Y. Fu Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases, workshop on Knowledge Discovery Seattle WA, July 1994

[18] W. Lu, Jan and B. Ooi. Discovery of General Knowledge in Large Spatial Databases, in Proc. Far East Workshop on GIS Singapore, June 1993

[19] L. Kaufman and P.J. Rousseeuw, Finding Groups in Data, an Introduction to Cluster Analysis, John Wiley & Sons, 1990.

[20] F. Preparata and M. Shamos, Computational Geometry, An Introduction, Springer-verlag NewYork 1985

[21] E. Thomsen. *OLAP Solution, Building Multidimensional Information Systems*, John Wiley & Sons, 1997

[22] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently, In *SIGMOD'96*, pp. 205-216, Montreal, Canada, June 1996

[23] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, The R*-tree: An Efficient and Robust Access Method for Point and Rectangles, in proceedings of 1990 to ACM SIGMOD Intl. Conf on Management of Data pp. 322-331 Atlantic City USA May 1990

[24] G. Shaw and D. Wheeler. Statistical Techniques in Geographical Analysis, London David Fulton, 1994