Access Structures for Fuzzy Spatial Queries

Aziz SÖZER
e-mail:e070364@ceng.metu.edu.tr

Adnan YAZICI
e-mail:yazici@ceng.metu.edu.tr

Department of Computer Engineering, Middle East Technical University, Ankara, 06531/ TURKEY

Abstract

Spatial data are complex and have spatial components and uncertain properties. It is important to develop effective spatial and aspatial indexing techniques to facilitate spatial and/or aspatial querying for databases that deal with spatial data. In this study we discuss a number of spatial index structures, such as Multi-level grid file (MLGF), R-tree, and R-tree, for fuzzy spatial and/or aspatial queries.*

Keywords: *Spatial database, Multi-level Grid File, R-tree, Fuzzy Spatial Querying.*

1. Introduction

The new application areas such as Geographical Information Systems (GIS), urban planning, astronomy, anatomy of the human body, very large scale integrated circuits (VLSI), the design of an automobile, and multimedia database applications pose serious challenges to the traditional database technology. At the core of these challenges is the nature of data which is manipulated. In traditional database applications, the database objects do not have any spatial dimension. The underlying data models, query languages and access methods were designed to deal with simple data types such as integers and strings. In a multi-dimensional space this data can be thought as *point* data. Furthermore, all operations on such data are one-dimensional. Thus, users may retrieve all entities satisfying one or more constraints such as employees with certain name and salary.

A spatial database is a database system that offers spatial data types in its data model and query language and supports such data types in its implementation, providing at least spatial indexing and algorithms for spatial join [8]. A spatial database management system has a number of requirements:

1. It is a full-fledged database system with additional capabilities for handling spatial data. Spatial, or geometric information is in practice always connected with "non-spatial" data. Nobody cares about a special purpose system that is not able to handle all the standard data modeling and query tasks.
2. It offers spatial data types (SDTs) in its data model and query language: Spatial data types, e.g. *point, line, region*, provide a fundamental abstraction for modeling the structure of geometric entities in space as well as their relationships *(l intersects r)*, properties *(area(r) >1000)*, and operations *(intersection(l,r)*-the part of l lying within r). In addition they may have some constraints such as *direction* (e.g. north), *topological* (e.g. overlap, inside) or *distance* constraints (e.g. 1 km away). Without spatial types a system does not offer adequate support in modeling.
3. It supports spatial data types in its implementation, providing at least spatial indexing and efficient algorithms for spatial join. A system must at least be able to retrieve from a large collection of objects in some space those lying within a particular area without scanning the whole set. Therefore *spatial indexing* is mandatory.

There are additional requirements for higher dimensional objects. Examples include spatial relationships such as *overlap, containment*, and *shared boundaries* between real objects, *incidence relationship* between a point and a line, and *containment* and *distance relationship* between a point and an area.

There has been considerable amount of work done for dealing with spatial and complex data especially in conceptual level. In the literature, there are a lot of models for spatial data ([4],[5]). In general, a conceptual model is a type of abstraction that uses logical concepts and hides the details of implementaion and data storage. As some researchers ([4],[10]) have been pointed out, the general-purpose conceptual models are not adequate for spatial applications. Therefore, there is a need for utilizing a powerful conceptual model to satisy unique requirements of these applications, such as the specific properties of geographic objects or entities and relationships. There are two common data models for modeling spatial information in GIS applications: *field based models* and *object-based models* [10]. Field-based models treat the spatial information space as a continuous domain such as *altitude, rainfall* and *temperature* as a collection of spatial functions transforming a space-partition to an attribute

0-7803-7461-4/02/$17.00 © 2002 IEEE

domain. The object-based models treat the information space as if it is populated with recognizable objects (spatial and aspatial) that are discrete and spatially referenced. The modeling approach that we take in this study is object-based.

Most of existing conceptual models assume that GIS applications do not involve in any form of *uncertainity*. However, it is not always possible to describe all the semantics of real world information precisely, since the observation and capturing of some real wold objects cannot be always perfect. In general, information may be both complex and uncertain. Uncertainity might arise from the data itself and/or the relations between objects so that complex objects and relationships in GIS applications may involve in various forms of uncertainity. As a consequence of these, queries involving imprecise and uncertain information may be unavoidable.

More specifically, the need for incorporating uncertainty in a conceptual modeling of GIS applications arises from the following reasons:

1. Some geographic information in GIS applications is inherently *imprecise* or *fuzzy*. For example, locations of geographic objects, some of spatial relationships and various geometric and topological properties usually involve in various form of uncertainty.
2. Most of the natural geographic phenomena have uncertain boundaries. For example, mountains, lakes, soils, slopes, etc.
3. Some measurements related to spatial domain are often incomplete. Sometimes forcing such data to be completely crisp may result in false and useless information.
4. In spatial applications, some of spatial domain related knowledge is specified in natural languages by using fuzzy terms and numerous quantifiers (e.g., *many, few, some, almost,* etc.). These quantifiers are fuzzy and are used when conveying vague information.

At the physical database design level, spatial database management systems have challenging parts. The effectivenes of the spatial database systems can be severely compromised if the spatial data are not properly managed in physical databases. Spatial databases are characterized by having large quantities of data associated with them. Since spatial data usually are complex and have a number of unique requirements (e.g. spatial components and uncertainty), classical one-dimensional database indexing structures are not appropriate for multi-dimensional spatial searching. However, many of the conventional indexing methods have been used as the starting point for designing spatial indexes. For example, *kD-trees* and its variations have been derived from binary trees. The other approach is to develop a dedicated spatial access structure to index on spatial attributes. A large number of the spatial access structures supporting spatial search operations in spatial databases have been multi-dimensional access methods [3]. There are different methods for storing and retrieving complex objects such as circles, polygons, etc. These are: (1) transformation approach, (2) overlapping regions, (3) clipping, and (4) multi layers. Among these methods, the R-tree [6] and R* -tree [2] were proposed by overlapping regions method. R+-tree [10] uses the clipping method. Z-ordering and Hilbert curve [1] are the most prominent ones in the transformation methods. Multilevel Grid File is an example of the multi-layers approach.

Many spatial queries may include some fuzzy and uncertain information. Uncertainty is an inevitable property of spatial and geographic data. In many new generation applications, fuzzy queries are usually combined with crisp queries. At the physical database design level, the current access methods [3] are inappropriate for representing and efficiently accessing fuzzy data. For the effectiveness of fuzzy databases, it is necessary to allow both the non-fuzzy and fuzzy attributes to be indexed together; hence a multidimensional access structure is required, so that the user can handle crisp/fuzzy spatial/aspatial queries efficiently.

Looking into the literature, we see that only a few researchers paid attention to fuzzy spatial/aspatial querying. [9] introduced a framework for configuration similarity that takes into account all major types of spatial constraints (topological, direction, distance). Helmer [7] studied indexing fuzzy data. In his study, he demonstrated how signature files (sequential, compressed, hierarchical, partitioned and inverted) can be used to speed up the retrieval of fuzzy data. Most of the existing multi-dimensional index structures process spatial and aspatial data or fuzzy and crisp data seperately. Considering the requirements of spatial database applications, fuzzy data as well as crisp data should be indexed together. In our study we bring spatial and aspatial data (fuzzy and crisp) together on the same index structures. Z-ordering technique is used to transform the location attribute into one dimension. We use Multi-level Grid File [14] for representing the data with spatial attribute (i.e., location) and aspatial attribute (i.e., population).

This paper will include the following subtopics: We will first give an overview of fuzzy spatial index structures, MLGF, R-tree and R*-tree. The queries which are used to evaluate the index structures will be explained in section 3. In Section 4 we give the experiemental results for comparing these spatial index structures. Finally we conclude.

2. Fuzzy spatial index structures

Traditional file structures are designed to handle single-key access to speed up the querying process. However, the requirements of complex applications have made traditional access structures ineffective. Since

spatial data usually are complex, we need spatial indexing structure that facilitates spatial and/or aspatial selection, even under uncertain conditions. The index structures introduced in this chapter are multi-dimensional and are extensions of Multilevel Grid File (MLGF), R-tree and R*-tree combined with z-ordering technique. The basic properties of the structure are to be able to index on fuzzy aspatial and/or spatial data and to process aspatial, spatial, and fuzzy/crisp queries.

2.1. Multi-level grid file (MLGF)

MLGF is a multi-dimensional index structure. Therefore, one can create an index on both spatial and/or aspatial crisp and fuzzy data. Many spatial database applications usually involve in fuzzy and complex information. For instance, if we do not know the exact population of a city, we may specify it as *crowded*. On the other hand, the population of the city is known beforehand and stored as a crisp value, i.e. 1 (one) million. MLGF handles merge and split operations on directories. The grid directory is maintained as a multi-level structure where each directory entry points to a lower level directory block. In MLGF, splitting and merging a directory is performed locally, thereby decreasing the amount of I/O that is required for a global split or merge. This causes the structure to be flexible during record insertion and deletion operations. Empty directory entries do not exist in MLGF. The organizing attributes are turned into bit patterns. Bit patterns of each organizing attribute are merged to form a key bit pattern. In MLGF, a directory entry is formed of a region vector and pointer. The region vector is a composite bit pattern that is composed of hashed bit patterns of the organizing attributes. We use z-code [1] representation for spatial attribute and population for aspatial attribute in index structure.

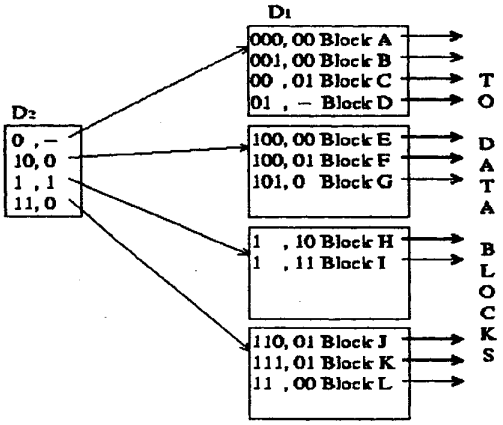


Figure 1. An example of MLGF structure [11]

2.2. R-tree and R*-tree

An R-Tree, proposed by Antonin Guttman [6], is an index structure for point and spatial data at the same time. Insert, delete and search can be intermixed without periodic reorganization. It uses a tuple to represent a spatial data in the database. In order to retrieve the data, each tuple has a unique identifier, tuple-identifier. At the leaf node of a R-Tree, it has index record that can reference the spatial data. The index record is (I, tuple-identifier). I is an n-dimensional rectangle and it is the bounding rectangle of the spatial data indexed. This rectangle is also known as minimal bounding rectangle, MBR, and each entry in tuple-identifier is the upper and lower bounds, [upper, lower], of the rectangle along the dimension. Non-leaf nodes contain entries (I, childnode-pointer) where I is the minimal rectangle bounding all the rectangles in the lower nodes' entries. Childnode-pointer is the pointer to a lower node in the R-Tree.

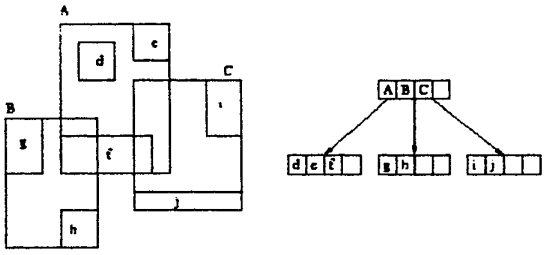


Figure 2. An example of R-tree

Based on a careful study of the behavior of the R-tree under different data distributions, Beckmann [2] identified R*-tree. In particular, they confirmed that the insertion phase is critical for search performance. The design of the R*-tree therefore introduces a policy called *forced reinsert*: If a node overflows, they do not split it right away. Rather, they first remove p entries from the node and reinsert them into the tree. The parameter p may vary; Beckmann suggest p to be about 30% of the maximal number of entries per page.

3. Querying the fuzzy spatial index structures

In this part we describe various query types for both MLGF and R-trees (R-tree and R*-tree). We test our modified index structures, R-tree, R*-tree and MLGF, in a geographic information system (GIS) database. A number of districts in central Ankara/Turkey with their population, z-codes and x-y coordinates are stored into MLGF and R-tree/ R*-tree.

3.1 Exact match query: crisp spatial and aspatial query

In the exact match query, input is a value of an attribute and location (z-code). The output is the points, which has the specific value(s) and z-code. This algorithm can be utilized for queries such as "Give the districts which have a population 1 million and coordinates are (x,y)". Since coordinates are given as (x,y), a transformation is taking place. For MLGF, z-code of the point is found by transforming coordinates into z-code. This is necessary since we hold z-code prefixes of the points in MLGF index structure.

We form a rectangle structure for the exact match in R-tree. The rectangle structure consists of spatial (x,y coordinates) and aspatial (i.e., population) data. Since we hold x, y coordinates in R-tree, we do not need to transform the input coordinates to z-codes as in MLGF. The output is the points, which has the specific population and z-code.

3.2. Fuzzy aspatial queries

Fuzzy aspatial queries are the queries such as "Give the places which are small" or "Give the cities which are 0.8 crowded". To deal with these queries we use partition trees for different fuzzy terms. To construct the partition tree of each fuzzy term, the domain is partitioned and membership values are assigned to the partition tree of fuzzy terms using the membership functions of *small, normal* and *crowded.*

Algorithm finds the leaf level nodes in the partition tree using the input threshold value. Any node below threshold is discarded. As we reach at the leaf level we have a bit pattern for partition value. Whenever a leaf is found in the partition tree, the final bit pattern is used to search the index structures, MLGF or R-tree. Note that we may also search the partition tree(s) of other fuzzy terms as well, in case the similarity of the specified fuzzy term to the other fuzzy terms is above a specified threshold.

3.3. Fuzzy spatial queries

In a fuzzy spatial query, one may ask "What are the districts on the South?" with a specified threshold value. In each direction, the objects are fetched within the threshold value.

3.4. Fuzzy Spatial and Aspatial Queries

This query is the combination of fuzzy aspatial queries and fuzzy spatial queries. An example for this kind of query may be "Give the districts in South 0.6 and 0.8 crowded". Algorithm finds two sets of object data, one for aspatial and one for spatial, then takes an intersection set. The two index structures use the same algorithm. MLGF uses z-code operations for spatial operations while R-tree uses rectangle operations. While MLGF makes prefix searches for population R-tree uses node's population lists.

4. Experimental Results

We performed some performance tests for both access structures. For performance tests we increased the number of records. A Pentium III-866 MHZ with 256 MB RAM PC is used for performance test. In the following figures, the results of tests are shown:

4.1. Crisp spatial and aspatial query

Crisp spatial and aspatial query is the exact match query (see Section 3.1), which accepts *population* and *location* as input.

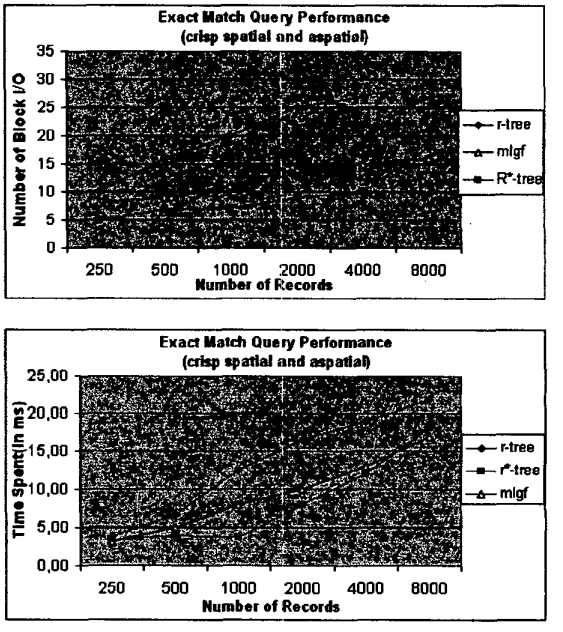
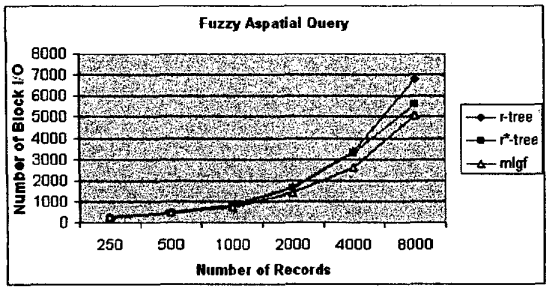


Figure 3. Comparison of the performance results of exact match query (crisp aspatial and aspatial)

4.2. Fuzzy aspatial query



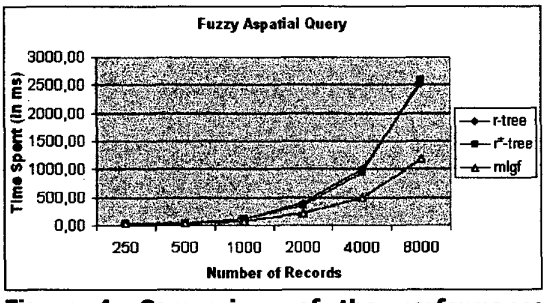
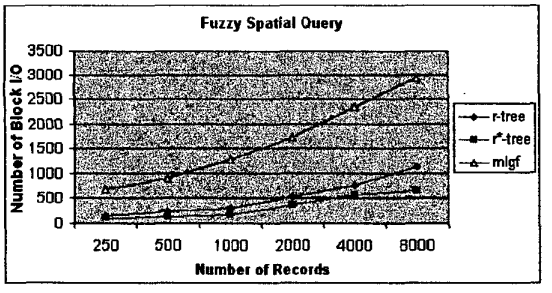


Figure 4. Comparison of the performance results of fuzzy aspatial query

4.3. Fuzzy spatial query



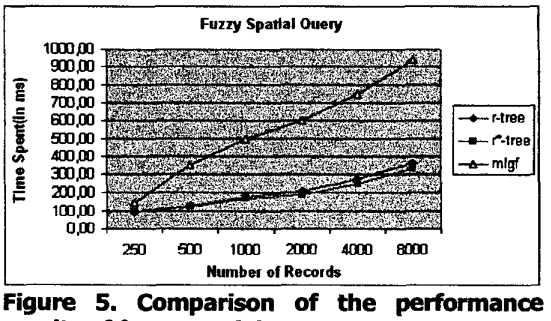
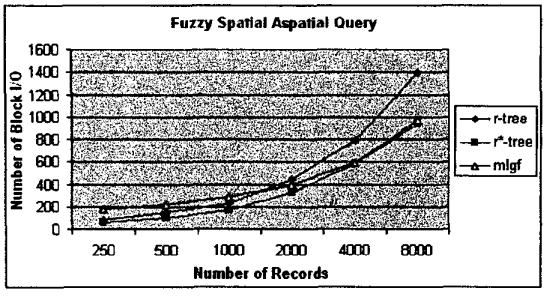


Figure 5. Comparison of the performance results of fuzzy spatial query

4.4. Fuzzy spatial and aspatial query

Spatial and aspatial queries are combined in the following experiment.



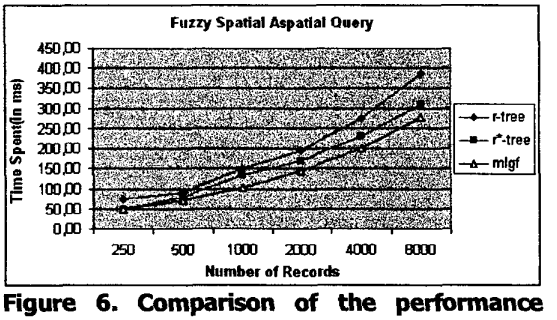


Figure 6. Comparison of the performance results of fuzzy spatial aspatial query

4.5. Comparison of index structures: R-tree/R*-tree vs. MLGF

Several algorithms have been tested with two index structures. The first group of experiments is about aspatial queries. MLGF gives better performance for aspatial queries (crisp aspatial, fuzzy aspatial). Because MLGF uses aspatial attribute (population) and spatial attribute (location) as primary index so index organization algorithm (insertion) takes into account both spatial and aspatial attributes. R-tree family has worse performance than MLGF for aspatial queries. In R-tree and R*-tree spatial attribute is a secondary index and spatial attribute is a primary index. So data distribution is organized by spatial attribute only. So this explains why MLGF is better in aspatial queries.

Second group of experiments are about spatial queries. For spatial queries such as crisp spatial, fuzzy spatial, R-tree family is better than MLGF. The reason is similar to the previous one. In R-tree family, spatial attribute is the main organizer of data (primary index) whereas MLGF data is organized by both aspatial and spatial attributes. So spatial data is retrieved less costly.

In the third group experiments where we combine spatial and aspatial queries such as crisp spatial/aspatial query, fuzzy spatial/aspatial query, R-tree family is better for message volumes up to 4000. MLGF becomes better after this point. The reason for that is that MLGF uses better organization which takes both aspatial attribute and spatial attribute into account. This becomes more effective as the message volume gets higher. Although it is not a direct purpose of this study, we should also mention that R*-tree gives better performance than R-tree for most types of queries.

5. Conclusion

In this paper, we discussed fuzzy and crisp spatial/aspatial queries on MLGF, R-tree and R*-tree index structures. For fuzzy queries we implemented a partitioning technique so that we could use these access structures for fuzzy querying as well as crisp queries.

R-tree/R*tree and MLGF index structures are dynamic. These access methods continuously keep track of the changes. They can use secondary and tertiary storage. It is not always possible to hold database in main memory. MLGF and R-tree/R*-tree index structures support broad range of operations (insertion, delete and retrieval). All index structures are simple and can be integrated with applications easily. MLGF can put index on many attributes because it is multi-level. R-tree is especially designed for spatial attributes (although we could extend it to adapt it to a aspatial attribute in addition to spatial attribute.) All index structures are scalable. They can adapt as database size grows. Spatial searches are fast and there is not a big performance difference between them (time efficient). In the node level they hold just the necessary attributes to reach to the data blocks. We can say that they are space efficient. We tested the access methods in a single user environment so concurrency and recovery issues should be tested in a multi-user environment within a database management system.

References

[1] Asano, T., Ranjan, D., Roos, T., Wiezl, E., and Widmayer, P. Space filling curves and third use in the design of geometric data structures. Theoretical Computer Science, 181 (1):3-15, 1996.

[2] Beckmann, N, Kriegel, H.P., Schneider, R. and Seeger, B. The R* tree: An efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD, 1990, pp. 322-331.

[3] Gaede,V. and Gunther,O., Multidimensional access methods, ACM computing surveys, Vol. 30/2, June 1998.

[4] Gunther O. and Riekert W.F., The Design of GODOT: An Object-Oriented Geographic Information System, IEEE Data Engineering Bulletin 16(3), 1993.

[5] Guting R.H., An Introduction to Spatial Database Systems, Special Issue on Spatial Database Systems of the VLDB Journal (Vol. 3, No. 4 October 1994)

[6] Guttman, A., R-trees: A dynamic index structure for spatial searching, Proceedings SIGMOD International Conf. Management Data, pp. 47-57, ACM, 1984

[7] Helmer, S, Indexing Fuzzy Data. Proc. Joint 9th IFSA World Congress and 20th NAFIPS Int. Conf., Vancouver, 2001: 2120-2125

[8] Orenstein, J., and F. Manola, PROBE Spatial Data Modeling and Query Processing in an Image Database Application. IEEE Trans. on Software Engineering 14 (1988), 611-629.

[9] Papaidas, D., Sellis, T., Theodoris, Y. and Egenhofer, M., Topological relations in the world of minimum bounding rectangles: A study with R-trees, Proc. SIGMOD International Conference for Management Data, pp. 92-103, ACM, 1995.

[10] Sellis, T., Roussopoulos, N. and Faloutsos, C., The R^+- tree: A dynamic index for multi-dimensional objects, Proc. 13th VLDB Conference, Brighton 1987.

[11] Whang K.Y. and Krishnamurty R., The Multi level Grid file- A dynamic hierarchical multidimensional file structure, Database systems for advanced applications, pp.449-456, 1991

[12] Yazıcı, A. and R. George, Fuzzy database modeling, Springer-Verlag, Heidelberg, 1999