Capabilities of R-, R^+- and R^*-Tree Indexing Spatial Data in Network Space

Vladan Mihajlović[1] and Slobodanka Djordjević-Kajan[2]

Abstract – **Because of the complexity and large amount of data in spatial databases, SDBMS demands additional structures, i.e. indexes, for speeding up query processing. Indexes can be organized in three mayor groups of techniques depending on the way that the structure divides data space. This paper consider three hierarchical indexes, R-tree, R^+-tree and R^*-tree. An experiment is performed on data extracted from a telecommunication network in order to examine performance of named indexes in managing network spaces. The results of experiment indicate that R^*-tree is the most efficient and the most robust indexing structure among these three structures.**

Keywords – **R-tree, spatial data, network space**

I. Introduction

Capabilities of computers in nowadays, larger hard disks and faster processors, make possible using large scale of complex data. Spatial data complexity depends on a method applied for modeling real data. The amount of spatial data, growing with new demands, decreases application efficiency. Standard DBMS do not preserve the information about dimensionality of space and do not satisfied new trends. This disadvantage invokes creating Spatial DBMS (SDBMS), which make connection between scalar and spatial data.

Deference between standard and spatial DBMS is in part that defines spatial semantics. This part is additional layer above the DBMS, which provide interface for application. The core of this layer consists of spatial taxonomy, spatial data models, spatial query languages, query processing algorithms and spatial access methods [1].

Spatial taxonomy depends on real problem domain. It introduces restrictions in modeling relationship between the objects in real world. The most popular taxonomies are set based space, topological space, Euclidean space, metric space and network space.

Selected spatial taxonomy affects the selection of spatial data model. There are two groups of spatial data model: the models based on mathematical fields and the object model [2]. Nowadays, object models are used by most applications.

Execution of spatial query is much slower than execution of standard query. The reasons are complexity of spatial data and variety of relationship between them. Comparison of spatial data approximation, instead of real data, speeds up query processing. Nevertheless, this does not satisfy terms set by user due to great amount of data. Organize the data in groups following some spatial criterion will help to solve this problem. Index is a structure that arranges data in groups according to some group property.

An index is a data structure, which create records with property that each record contains data with specific key or that satisfied specific criterion. Thus query processing is reduced to examine particular record or smaller group of records and isolating data in it. In this way, indexes make spatial query processing efficient by decreasing the amount of data, which is needed to be retrieved from hard disk.

Nowadays, index is part of every DBMS, which manage database with large amount of records and objects of complex data types. Development of indexes for complex data types began in the field of spatial data. The indexes in spatial application are necessary due to complexity of spatial data objects. Indexes are usually built of simpler objects, which are approximations of real data. Accordingly they spare checking time and accessed disk pages. Indexes, as par of SDBMS, are used in Geographic Information Systems (GIS), CAD Systems, Computer Vision, Multimedia Information Systems and Data Warehousing Systems.

The rest of this paper considers application of spatial indexes in GIS. The second section gives an overview of different approaches to indexing spatial data. The third section outline definitions of R-tree, R^+-tree and R^*-tree, which are three very efficient and relatively simple indexing structures. Next section describes experiment, which use real data to compare performance of the three structures. Data used in experiment are extracted from telecommunication network. Results of comparison are presented in fifth section. The end of paper emphasizes accomplished research and gives some recommendation for selecting some of tested indexes in our applications.

II. Different Approaches to Spatial Data Indexing

Indexes designed for scalar data types are pretty simple since the key value can be determined strictly. A basic feature of spatial data is multidimensionality. This invokes problems since memory has only one dimension. This difference diminishes performance of indexes designed for scalar data types. This also provokes creation of fully new indexes designed for multidimensional data. Two major features of an index are how it partitions the data space and how it associates data with subspaces. According to these two features, indexing structures for spatial data can be classified in the three following approaches [3]:

[1]Vladan Mihajlović, Faculty of Electronic Engineering, University of Nis, Beogradska 14, 18000 Niš, Serbia and Montenegro, E-mail: wlada@elfak.ni.ac.yu

[2]Slobodanka Djordjević-Kajan, Faculty of Electronic Engineering, University of Niš, Beogradska 14, 18000 Nis, Serbia and Montenegro, E-mail: sdjordjevic@elfak.ni.ac.yu

1. The transformation approach. This approach comes in two flavors:

– *Parameter space indexing*. Object described by n vertices in a k-dimensional space are mapped in an nk-dimensional space. Points generated by such mapping can be stored directly in indexing structure designed for point data. An advantage of this approach is that is does not require creation of new index or modification of existing one. The main drawback is that the spatial proximity is not preserved in most cases.

– *Mapping to a single attribute space*. The k-dimensional data space is partitioned into grid where cells have the same size. The cells are labelled according to some curve-filling methods. So, spatial object is represented by set of numbers depend on cells that intersect with it. Therefore, objects can be indexed using conventional indexes for scalar data, like B^+-tree [4]. Shortcoming of this approach is multiplication of index entries.

2. The non-overlapping native space indexing approach. There exist two classes of techniques:

– *Object duplication*. A k-dimensional data space is partitioned into pairwise disjoint subspaces. These subspaces are then indexed. An object identifier is duplicated and stored in all subspaces it intersects.

– *Object clipping*. This is very similar to previous approach. Instead of duplicating the identifier, an object is decomposed into several disjoint smaller objects so that smaller sub objects are totally included in a subspace.

The most important property of this approach is that data structures used are straightforward extensions of the underlying point structures. This kind of indexes can store both points and objects with extensions together without having to modify the basic structure. Huge drawback of this approach is duplication of objects, which requires extra storage space and, more expensive insertion and deletion operations.

3. The overlapping native space indexing approach. The leading idea of this approach is to partition the data space into manageable number of smaller subspaces, which are hierarchically organized. Point object is included in one subspace, but nonzero sized objects may extend over more than one subspace. To assign nonzero sized object to exactly one subspace, subspaces are allowed to overlap. Spatial objects are indexed in their native space using this approach and that is its main advantage beside hierarchical organization. Main drawback is higher costs of insertion and deletion operations.

The overlapping native space indexing is the newest one. The basic advantage of this approach is preserving proximity relationship between objects that represent real data. Proximity is the basic relationship in every metric space. This indexing approach furthermore preserves proximity relationship between the entries on each level of hierarchical structure. A major design criterion for this approach is to minimize the overlap between subspaces and the coverage of subspace. The R-tree uses this kind of indexing approach.

III. Three Basic Types of R-Tree

An R-tree [5] is a high-balanced, hierarchically organized structure. A leaf of R-tree contains identifiers of database tuples, i.e. pointers to data objects. This tree is based on B-tree. Size of node corresponds to memory page size. R-tree is completely dynamic and no periodic reorganization is required.

R-tree is formed over data objects approximation. In two-dimensional space objects are approximate with minimum bounding rectangle (MBR). In the following we will describe two-dimensional R-tree. A leaf of tree consists of set of (I, id) entries, where id is unique database tuple identifier and I is MBR of data object identified by id. An internal node of tree contains entries of the form (I, ptr) where ptr is address of child node in the tree, and I is MBR that covers all rectangles in the lower node's entries.

Two values are typical for every R-tree. These are maximum number of entries that will fit in one node (M) and minimum number of entries in node $(m < M/2)$. Maximum number of entries is determined according to size of disk page. Minimum can be changed to improve structure performance. Only root node can have fewer entries than minimum. During the tree making node overflow or underflow will appear. The underflow is settled by reinsertion of entries that are rest in node. The overflow is resolved by splitting node in two parts. Two split algorithms are offered, linear and quadratic. Linear algorithm assigns elements to one of two new nodes one by one. Quadratic algorithm form two groups around two furthest elements using the criterion of minimum area covered.

Next variant of R-tree is R^+-tree [6]. Consider the advantage of the non-overlapping native space indexing approach author of R^+-tree modified R-tree so that overlapping between entries in internal node is equal to zero. Since there is no overlapping query, and there is no need to traverse multiple paths and the queries will execute faster. To achieve disjunction property between entries of level immediately above the leaf level same data object can be element of more than one leaf node. R^+-tree has not lower bound in number of node's entry (m) and if deleting is frequent performance of the tree can be deteriorate. This tree is not dynamic structure and requires periodic reorganization.

Objective of authors of R^*-tree [7] is to prove that overlapping between internal node's entries does not lead necessary to index which has low efficiency. R^*-tree has the same hierarchical structure as R-tree. The difference between them is in insertion and deletion algorithms. The authors carried out several criteria, which were believed to have the greatest effect on index performance. These criteria are: minimization of the area covered by rectangles in internal nodes, minimization of the overlap between rectangles in internal nodes, minimization of the margin of rectangles in internal nodes and optimization of storage utilization. The authors of R^*-tree analyzed interdependencies among different parameters and optimization criteria and define insertion and deletion operation. Thus they proposed a split algorithm for R^*-tree that firstly determines the axis perpendicular to which the split will be performed and then determines the best distribution of elements in two groups along that axis. One of following three criteria can be used in both parts of algorithm, unused area minimization, overlapping minimization and margin minimization. The new parameter introduced in

R*-tree is p. R*-tree treats overflow differently if it appears for the first time on particular level. In this case reinsertion is forced. Parameter p defines the number of entries that need to be reinserted. If an overflow appears for the second time in the same level of the tree, the split is performed. This forced reinsertion reorganizes the tree structure and improves its performance.

IV. Experiment Layout

Purpose of the experiment was to compare performance of three types of R-tree tested on real data. For this experiment all three R-trees are implemented in C^{++} language. Implemented trees manage two-dimensional data. Special software was designed in Visual C^{++} 6.0 programming environment. This software is used for performing large scale of tests on real and semantic data space.

In this paper will be presented tests performed on real data space. Data in this space is network of telecommunication canals and cables. Three tables of network data extracted from the spatial database. In the first (P1), the telecommunication cables represent spatial data. This space consists of 192 data and MBRs cover 10% of space area. The second table (P2) contains linear segments of the cables. This table has 848 tuples and data approximation occupy 1% of space area. Overlapping of data MBRs in previous two spaces is minimal. The third set of data (P3) consists of start and end points of cable segments and some key points in the telecommunication network. Total number of points is 5929.

Query represents search operation, which returns a set of data that satisfy requested criterion. For testing index structures, the three basic queries are used: point query and two types of rectangle query. The point query returns data, which MBRs contain given point. The overlap rectangle query returns data, which MBRs overlaps with given rectangle. The second type of rectangle query returns data, which MBRs enclose given rectangle. The first group of queries is point query (U1). The second are enclose query with query rectangle which area is 20% and 50% of elementary data (U2 and U3). Elementary data is square which area is equal to space area divided with number of data in space. Size of query rectangle in overlap query is equal to elementary data (U4), five times bigger (U5) and ten times bigger (U6) then elementary data.

Within each indexing structure the variant with the best performance is chosen for mutual comparison. To determine the best variant of R-tree was selected by changing parameter m from 10% to 50% with step of 5%. R-tree with linear and quadratic split algorithm was observed separately. Considered variants of R$^+$-tree have nodes filled from 75% to 95% of their capacity with step of 5%. To select best variant of R*-tree parameters m and p had values from 10% to 50% with step 10%, and with all possible combination of three criterion for choosing axes and distributions. Trees with different maximum number of elements in node (M) were compared separately because this parameter was imposed by memory page size. This parameter took four values, 13, 28, 56 and 113, according to page sizes of 0.5 kB, 1 kB, 2 kB and 4 kB.

Purpose of index is to minimize the number of disk access. Accordingly performance of three types of R-trees measured by number of memory pages accesses. Because all three structures have property that nodes exactly fit to one page, the efficiency is measured by number of nodes accesses during the query execution.

V. Interpretation of The Results

Because of small number of data in space P1 it is difficult to make general conclusions, but some trends can be isolated. R-tree that use quadratic split algorithm shows better results than linear counterpart. Property of R$^+$-tree that internal nodes must be disjunctive, even in this space with small number of nonzero sized data, demonstrate its main drawback significant increase in number of nodes in tree. All examined variants of R*-tree, with different criteria used in split algorithm, have nearly same performance (difference is below 10%). Variants of R*-tree with greater freedom in selecting minimum number of elements in node (m) have better results than the more restrictive variants.

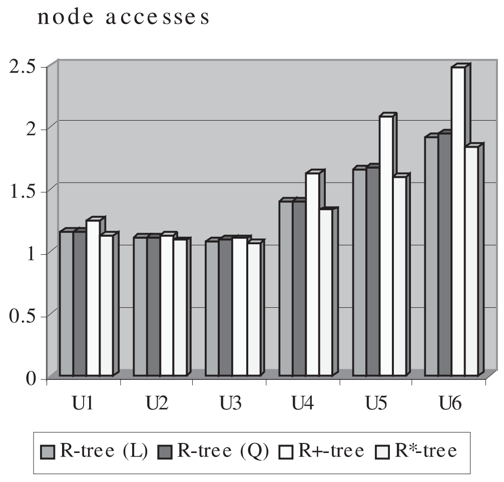

Fig. 1. Performance of indexes tested on linear segments as data objects of real network space

The results of query processing of P2 space demonstrate performance of tested index structures in real network spaces. Both split algorithms for R-tree, linear and quadratic, show almost equal performance. The variants of R-tree with greater freedom in number of elements in node appear to be superior. This advantage, which goes with greater freedom, is result of small MBRs of line segments and that each segment is a part of multi-line string that represent one cable. Same feature show R*-tree. This invokes rising of minimum node elements threshold for improving storage utilization. Criteria that are most efficient for R*-tree split algorithm are minimizing of the margin for choosing the axis and minimizing the overlap for selecting appropriate distribution. Parameter p has minimal influence on R*-tree performance. R$^+$-trees in P2 space show two features. The first is more elements in R$^+$-tree than in R-tree or R*-tree constant value for parameter M. The second is that efficiency of tree does not depend on storage utilization, as it is determined for point data. Figure 1 present performance of linear R-tree, quadratic R-tree, R$^+$-tree and R*-tree for different types of queries and parameter M set to 56. R$^+$-tree demonstrates poor performance, especially for overlap queries. This shortcoming is affected by traversing multiple paths since query rectangle intersect more then one entry in node. This is a consequence of poor algorithm for tree creation. R*-tree has minimum node accesses per query due to good minimization of node area and node overlapping. The other values for maximum number of entries in a node do not make qualitative changes on results.

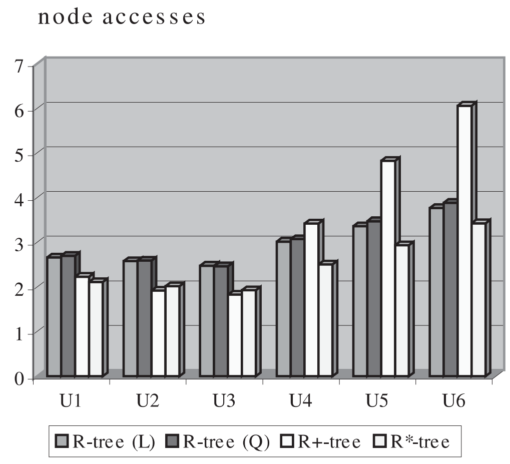

Fig. 2. Performance of indexes tested on points as data objects of real network space

Experiment on P3 space of points shows some distinction from expectation. Previous experiment performed by different authors concludes that quadratic split algorithm has better results than linear. The test on point data extracted from telecommunication network indicates to different conclusion. Linear variant of R-tree has better results than quadratic no matter what value is choused for M. Reason for this is probably the particularity of data space. Tests on synthetic data show that the best R*-tree variant uses minimization of margin and overlap as criteria for splitting node. In this space, minimization of node area results with most efficient tree. Although, the performances of variant that use other criteria are only 1% worse than the best variant. Also, storage utilization for R*-trees is much lesser than in synthetic data. If it is allowed to deteriorate efficiency about 1%, the variant of R*-tree with far better storage utilization can be found. Figure 2 show number of nodes access per query for different types of queries and M set to 56. R$^+$-tree has minimal node access when point and enclosing rectangle query are performed because the overlap between node's entries is equal to zero. But this advantage disappears with overlap rectangle query. That fact can be explained by large volume of unused space in internal node entries. Consider all six groups of queries R*-tree is the best choice overall. More data about the experiments can be found in [8].

VI. Conclusions

In previous papers that consider performance of R-tree authors agreed that a quadratic split algorithm creates better structure than a linear split algorithm. Linear and quadratic R-tree formed over telecommunication network space presented in this paper show nearly same efficiency. Larger overlapping between entries in internal nodes causes degradation in search performance. Basic property of R$^+$-tree, that overlapping between entries in internal nodes is zero, results in very efficient performing of rectangle enclosure query in point space. Disadvantage of this structure is unsatisfactory insert algorithm, which form entries with large unused space and make R$^+$-tree useless for nonzero sized data. Main drawback of this index is that it is not dynamic structure.

Algorithms of R*-tree are designed to enable greater influence of used criteria in creating nodes of index, which improve structure quality, i.e. minimize overlap and unused node area. Number of node access during query processing over R*-tree has logarithmic proportion with the number of data in space, i.e. depends directly on number of levels in tree. Quality of R*-tree has slightly affected by size and shape of data. Involved experiments with network data, which form variety of shapes, strongly affirm this fact. R*-tree is more robust and more efficient structure than the other two.

Acknowledgement

The research was partially supported by the project "Geographical Information System Designed to Improve the Local Municipal Function based on Internet/WWW Technologies", funded by Ministry of Science, Technology and Development, Republic of Serbia and Municipality of Niš, Contract No. IT.1.23.0249A.

References

[1] S. Shekhar, S. Chawia, S. Ravada, A. Fetterer, X. Liu, C. T. Lu, "Spatial Databases - Accomplishments and Research Needs", *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, Jan./Feb. 1999.

[2] M. Worboys, *GIS: A Computing Perspective*, Taylor & Francis, 1998.

[3] E. Bertino, B. C. Ooi, "The Indispensability of Dispensable Indexes", *IEEE Trans. on Knowledge and Data Engineering*, vol. 11, no.1, pp. 17-27, Jan./Feb. 1999.

[4] R. Bayer, E. McCreight, "Organization and Maintenance of Large Order Indices", *Proc. 1970 ACM-SIGFIDET Workshop on Data Description and Access*, pp. 107-141, Houston, Texas, Nov. 1970.

[5] A. Guttman, "R-Trees: A dynamic Index Structure for Spatial Searching", *Proc. ACM SIGMOD Intl. Conference on Management of Data*, pp. 47-57, 1984.

[6] T. Sellis, N. Roussopoulos, C. Faloutsos, "The R$^+$-Tree: A Dynamic Index for Multidimensional Objects", *Proc. of the 13th Intl. Conference on Very Large Databases Conference*, pp. 507-518, Brighton, 1987.

[7] N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", *Proc. ACM SIGMOD*, pp. 322-331, June 1990.

[8] V. Mihajlović, *Spatial data indexing*, Diploma thesis, Faculty of Electrical Engineering, Niš, May 200