Variations of R-tree structure for indexing of spatial objects

M.G. Martynov
St.-Petersburg university
E-mail: ban@niimm.spb.su

Abstract

The data structures suitable for indexing of large spatial objects in databases are considered. Some generalizations of existing tree methods concerned approximation of spatial objects and composite optimization criteria are proposed. The efficient algorithms, including one for spatial join operation, providing for optimize considered methods are described. The results of experimental measurements of different variations of R-trees and proposed methods are presented and compared.

1 Introduction

The physical organization of database systems storing complex spatial objects is considered. Common examples of such databases include geometric and geographical databases, as well as of temporal database systems which support more than one time dimension.

The most important requirements for these data structures are ability to provide fast access to large volumes of data and preserve spatial relations such as nesting and neighborhood for indexed objects. Several tree-like access methods were proposed for spatial objects [4], including different kinds of R-trees [5, 1, 3] and C-trees [2]. All these tree-like structures use approximation of spatial objects with bounding objects: rectangles or arbitrary convex polygons, respectively. The latter provides better precision of search, while the former is generally more efficient during updates.

Methods investigated in this paper do not use clipping technique which applied in R$^+$-trees [5], because of bad performance with uneven data.

The paper is organized as follows. In section 2 the structure of spatial trees and existing R-tree variants are described. New variations of R-tree structure and efficient spatial join algorithm are considered in section 3. The results of performance comparison and experimental setup are reported in section 4. Section 5 concludes the paper.

2 Spatial trees

An spatial tree is a B$^+$-tree like structure which stores spatial bounding objects without clipping them or transforming them to higher dimensional points before.

A leaf nodes contain entries of the form $(Oid, Sobj)$, where Oid refers to a record in the database describing an object of data, and $Sobj$ is enclosing spatial object of it. A non-leaf nodes may contain such an entries and entries of the form $(Ref, Sobj)$, where Ref points to the child node, and $Sobj$ is the minimum bounding object of all entries in that child node. Let M and m be the maximum and minimum number of entries that will fit in one node correspondingly ($1 < m \leq M/2$). A spatial tree satisfies the following properties:

- The root has at least two children unless it is a leaf.
- Every non-leaf node has between m and M children unless it is the root.
- All leaves appear on the same level.
- All non-leaf nodes have at least one entry of the form $(Ref, Sobj)$.

Unlike the B-tree structure an R-trees are non-deterministic in allocating the entries onto the nodes, that allows to optimize tree structure. However full optimization is not possible, and actually heuristic optimization method is usually applied, based on running large volumes of experiments with high varying data.

The most important optimization criteria are minimum of area covered by bounding objects of a nodes, minimum overlap between objects of a node, minimum of margin for bounding objects and maximum of storage utilization. Analysis of these criteria in detail can be found in [1]. Besides them some values of shape and size of the objects including their spatial relations may be considered.

An R-trees use approximation of spatial objects with bounding rectangles with the sides of the rectangle parallel to the axes of the data space. Local optimization of the tree structure is applied during the insertion of a new data rectangle which is subdivided into two steps. First is in choosing insertion path and the second is in splitting of overfilled nodes.

Original variant of R-tree proposed by Guttman in [3] at the first step uses minimization of the area enlargement for objects along insertion path. Greene's algorithm proposed later uses the same criterion. R*-tree algorithm differs from described one in using overlap enlargement criterion on the level preceding to the leaf level. It reduces number of disk accesses but the CPU cost is too high.

Guttman discusses split-algorithms with exponential, quadratic and linear cost with respect to the number of entries of a node. All of them are designed to minimize the area of the two rectangles resulting from the split. Second algorithm gives good performance with not high CPU cost. It divides a set of $M + 1$ entries into two groups. At first two most distant entries of a node are chosen which will be the first entries of the groups, then another entries are distributed in it on the area criterion.

Greene's and R*-tree split algorithms choose axis perpendicular to which the split is to be performed, then sort entries along the chosen axis and divide entries onto groups. Greene's algorithm divides entries in two, and R*-tree does it better using overlap criterion for bounding objects of the groups. Besides R*-tree uses forced reinsert to archive tree structure during the insertion routine. It consists in deleting and reinserting entries removed from the center of a node. These method leads to more exact distribution of the entries and improves storage utilization. Exact description and analysis of these algorithms contain in [1].

3 The A-tree

3.1 Abstract spatial objects

R-tree methods described in section 3 can be modified for use approximation with abstract object instead of rectangles. It will allow to apply these methods with different approximations without significant change. Implementation of the abstract object must include definition of type of spatial object and the set of procedures to work with them, which are common for all approximations. Spatial object must be a convex bounded and closed set. Analysis of the algorithms supporting tree operations allows to select following set of procedures.

- Voiding of the object.
- Extending of the object with the point.
- Extending of the object with the other object.
- Restricting of the object with the other object.
- Getting the center of the object.
- The set of shape and size criteria of the object.

The set of criteria must include procedures that allow getting area diameter and compactness of the objects. In every implementation must be defined fixed number of an axes on which lower and upper boundary of the objects may be found. If an intersection is not closed for implemented type of the spatial object then procedure for restricting objects with other objects must be replaced with procedure which gives geometric parameters of the set resulting after intersection.

Polygons with the sides parallel to the defined axis were implemented for comparison with rectangles corresponding with the described scheme.

3.2 Composite optimization criteria

Algorithms are usually optimized by choosing suitable criterion from the pre-defined set of it. For example R-tree variants use criteria of area, margin, overlap and sometimes their combination e.g. in split of R*-tree.

No single criterion can yield best possible performance, and algorithms work with one value e.g. for searching insertion path. Number of these values can be extended by using functions of basis criteria. Thus infinite set of criteria appears from which most appropriate can be found.

Set of functions must be restricted in order to be represented by the finite number of parameters. The most simple way is in searching for best linear combination of criteria. This task is difficult because correlation between coefficients of linear combination and performance of considered methods is not known. Only certain smoothness can be presumed in order to apply iteration method for improving first plan, choosing direction of decreasing of the aim function by chance. This method is heavy from computational point of view, because every iteration includes building–up and testing the tree.

3.3 Using of the main memory

Special algorithm storing upper levels of the tree in main memory was implemented for caching of disk. It has advantages before resident cache programs, because allows avoid needless page copying and uses dead space of the nodes, applying floating pages technique.

3.4 Spatial join

Spatial join (SJ) is usually defined as a set of pairs intersected objects from two sets of data.

In multi-dimensional case applying different linearizations [4] for SJ is not effective. The other way is to perform SJ with methods saving spatial relations. Tree structures are considered here satisfy this property.

The most simple way of using spatial trees for SJ consists in decomposing SJ on the single queries, when first set of data is organized in a tree and the second gives query objects.

Joining of two trees is more effective. Three methods of this type will be described here. First of them uses combined searching for access paths in the trees. It is extension of the algorithm for searching path when intersection query is performed. Three variants of this method were considered which differ in choosing active tree in every step. First variant takes only first tree, the second takes tree with less current level and the third takes tree with the maximum of area of the current object. Last of them is better than others because it takes into account geometric properties of the spatial objects.

The following algorithm is based on property of the asymmetry of reading pages and yields best performance. This property means that the order of reading pages during SJ influences on less necessary number of pages to be read.

Two ways of finding right reading order will be described below with algorithm of SJ which uses them. Let's consider the first.

Let we have two sets of entries E_1 E_2 which contain ones from first and second trees respectively. Let $E = E_1 \cup E_2$. Set P defined below will contain all pairs of entries which can be add to resulting set of SJ.

$P = \{\{e_1, e_2\} : \, e_1 \in E_1, e_2 \in E_2, \; e_1.o \cap e_2.o \neq \emptyset\}$

Let's define on P relation of influencing between pairs. Two pairs x,y will be named *closely linked* if $x \cap y \neq \emptyset$. Then the relation of *influencing* will be defined as reflexive–transitive closure of the *closely linking* relation. It means that pairs connected with it influence each other, so that reading of element from one pair can change last information about order in which elements of other pairs should be read. That information can be found from geometric relations between objects of the pairs.

Relation of *influencing* divides P into independent classes of equivalence which can be considered separately in any order. The order of reading entries in one class must be defined.

In order to be abstracted from real geometric parameters of the objects function $Fo(e_1, e_2) \rightarrow [0, 1]$ have to be introduced which for pair of entries $\{e_1, e_2\}$ will be return probability that after reading e_2 reading of e_1 will be not necessary.

Two solutions can be proposed for task of choosing right reading order on the one class. First of them is based on calculation probable number of pages to be read in the class after reading one page from it.

Let's introduce some designations to describe the method. We are given one class Cs, then for $X \subset Cs$ following function can be defined $Fe(X) := \{e : \quad \exists x, \; e \in x \in X\}$, and for $e \in Fe(X)$ we define the set $M(e) := \{x : x \in X, e \in x\}$.

Calculations are produced by recursive function $Fd(X, Q)$, where $X \subset Cs$, and $Q \subset Fe(X)$. The set Q will contain entries have to be read after reading other entries. The function will return minimum number of reading pages in (X,Q) and the page must be read first for initial sets. Call $Fd(Cs, \emptyset)$ will give reading order for Cs.

Definition $Fd(X, Q)$**.**

FD1 If $Q = \emptyset$ then

$$Fd(X, Q) = \min_{e \in Fe(X)} \{0, Fd(X, \{e\})\} \, .$$

If $X = Cs$ then e on which minimum is reached is the first entry to be read.

FD2 If $Q \neq \emptyset$ then $Fd(X, Q) = 1 + \sum_{G \subset Em} (L(e, G) \cdot Fd(X \backslash M(e), Q \cup G \backslash e))$, for one entry $e \in Q$, where $Em := Fe(M(e)) \backslash e$, and $L(e, G) = \prod_{x \in Em} p(x)$, where

$$p(x) := \begin{cases} Fo(x, e) & x \notin G \\ 1 - Fo(x, e) & x \in G. \end{cases}$$

If $Em = \emptyset$ then $L(e, G) := 1$.

The second method for choosing right reading order gives less exact decision, but it's implementation cost is lower. It contains in choosing page which has maximum of probability that the page will be read as a first page to be read. For entry e this value is

$$1 - \prod_{\{e,x\} \in Cs} Fo(e, x)$$

.

Algorithm SJ used one of the proposed methods will be presented below.

Algorithm SJ.

SJ1 Place entries of the tree roots in sets A_1, A_2. Initialize stack of classes (SC).

SJ2 Divide A_1, A_2 into classes and push them on stack.

SJ3 Pop from stack all classes which contain only data entries and add their relevant pairs to the result set of SJ.

SJ4 If stack is empty then end.

SJ5 Pop one class from stack and choose entry in it which is not entry of data as described. Place the class in A_1, A_2. Read node is pointed by choosing entry and change it in A_1, A_2 on the entries of read node.Go to SJ2.

Described algorithm is more effective then others because pages are read one time during one SJ and right reading order is supported.

Two variants of function F were considered.

1. $Fo(e_1, e_2) := area(e_1.o \cap e_2.o)/area(e_1.o)$
2. $Fo(e_1, e_2) := \prod\limits_{1 \leq i \leq mg} \frac{area(e_1.o[i] \cap e_2.o)}{area(e_1.o[i])}$

where $e_1.o[i]$ is edge number i and mg is a number of edges. Besides in both variants $Fo(e_1, e_2) := 0$ if e_2 is data entry.

3.5 Addition to the split–algorithm

Experiments was done showed that optimal insertion algorithm of R*-tree when splitting extended nodes prefers to choose large axis perpendicular to which split will be performed. The following modification of the split algorithm can be proposed which uses this fact. It will allow not only decrease calculation cost of the split but also do objects more compact.

SP1 Calculate extent of compactness of the node bounding object. If it is less than pre-defined constant (chosen with regard of parameters of using method) then choose split axis along which bounding object has largest extension else choose split axis by original algorithm.

SP2 Perform distribution of the entries along chosen axis.

4 Experimental results.

Following experiments were carried out.

1. Performance comparison of different methods for varying data and for different bounding object types (BT).
2. Performance comparison for different algorithms of SJ.
3. Testing dependence of choosing split axis for Greene's R-tree and R*-tree.

For experiments 1 and 2 page size was fixed as 1 Kb and memory cache as 300 Kb. M was set to maximum concerning given page size and object type. $m := 40\%$, and number of deleting entries in algorithm of forced reinsert was set to 30% from M, what corresponds to the best values for considered algorithms.

Two classes of spatial objects were selected for testing: standard rectangles and right polygons with sides parallel to the fixed axes on the plain.

Four methods were tested.

1. (Simple) which uses area criterion for choosing insertion path, and splitting in half perpendicular to the axis along which bounding object of a node has largest extent.
2. (qua.Gut).
3. (Greene).
4. (R*-tree).

To compare the performance of the methods data sets of 5 types containing about 20000 objects each were generated. In order to compare different BT the single scheme of generating objects of data was selected . New object was formed by extension of the void object by pre-defined number of points uniformly distributed in sphere with predefined radius which center had pre-defined distribution. Unit cube $[0, 1)^2$ was taken as a working space. Every type of the object generation was described by triplet (Np, Rad, Cd), where Np is number of points, Rad is a sphere radius and Cd is a distribution of the sphere centers. Np influences on extent of compactness of the objects and Rad influences on mean value of its area. Used data types are:

1. Uni.l : (3; 0.01; Uniform).
2. Uni.g : (6; 0.01; Uniform).
3. Uni.mix : (3; 0.01 in 79 from 80 cases and 0.05 in 1 from 80 cases; Uniform).
4. Gauss. : (3; 0.008; Gaussian).
5. Cluster : Centers of the data objects are uniformly distributed in 1000 clusters. In each cluster centers have Gaussian distribution with parameters (center of the cluster, 0.01). Centers of the clusters uniformly distributed. $Np = 3$, $Rad = 0.008$.

A tree was built with testing method for every type of data and a queries of following three types were done.

1. *Point query:* Given a point P find all data objects O with $P \in O$.
2. *Intersection query:* Given an object Q find all data objects O with $O \cap Q \neq \emptyset$.
3. *Enclosure query:* Given an object Q find all data objects O with $Q \subset O$.

400 point queries were performed with uniformly distributed query points and 400 queries for 2 and 3 query types, which objects had following parameters (3; Rd; Uniform), where $Rd \in \{0.1, 0.05, 0.02, 0.001\}$ for 2 query type and $Rd \in \{0.02, 0.001\}$ for 3 query type.

For each generation of the objects mean value and normalized variance of their area were measured. Building-up the tree was estimated by three parameters: coefficient of storage utilization and mean value of the disk accesses with memory cache and without it. For queries two disk access parameters were estimated as described and mean number of the objects returned from queries was measured.

Comparing of the SJ algorithms was done under the following scheme. For each tree method for 3 first data types the trees were built with following number of elements (1000, 1000), (1000, 2000), (2000, 2000). For each pair of the trees 5 SJ algorithms were done, incidentally disk accesses and mean number of result pair for SJ was estimated.

Testing properties of choosing split axis was carried out for Greene's split and split of R*-tree. For this purpose nodes with definite number of entries were generated many times, then it were splitted and coincidence of the axes was determined.

Experiments show that:

- retrieval performance: Greene $<$ Simple $<$ Qua.Gut $<$ R*-tree.

- CPU cost:
  Simple $<$ Greene $<$ Qua.gut $\ll$ R*-tree.

- SJ algorithms: SJ1 $<$ SJ2 $<$ SJ3 $\ll$ SJ5 $<$ SJ4.

- spatial description type: rectangle $<$ pentagon in the point queries, in the enclosure queries with the small query objects and in the intersection queries with the large query objects, and rectangle $>$ pentagon in other queries.

High CPU cost of R*-tree is only due to using overlap criterion in choosing insertion path. This criterion does not give considerable improvement of retrieval performance and it will hardly be used in real systems. SJ algorithms which use asymmetry of reading pages have total advantage before others. In some operations using description with pentagons showed better performance, although packing of polygons was not applied.

5 Conclusion

Abstract type of the bounding object and composite optimization criteria for building-up a tree have been proposed. Efficient spatial join algorithm based on asymmetry of reading pages have been presented. Performance comparison for considered methods, including spatial join have been done.

References

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

[2] O. Günter and H. Noltemeier. Spatial database indices for large extended objects. In *Proc. 7th Int. Conf. on Data Eng.*, pages 520–526, Cobe, Japan, 1991.

[3] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[4] D. Lomet. A review of recent work on multi-attribute access methods. *ACM SIGMOD Record*, 21(3):56–63, 1992.

[5] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R-tree: a dynamic index for multi-dimensional objects. In *Proc. 13 conf. VLDB*, pages 507–518, 1987.