Semantic Spatial Objects Data Structure for Spatial Access Method

Kalum Priyanath Udagepola, Zuo Decheng, Wu Zhibo, and Yang Xiaozong

Abstract**—**Modern spatial database management systems require a unique Spatial Access Method (SAM) in order solve complex spatial quires efficiently. In this case the spatial data structure takes a prominent place in the SAM. Inadequate data structure leads forming poor algorithmic choices and forging deficient understandings of algorithm behavior on the spatial database. A key step in developing a better semantic spatial object data structure is to quantify the performance effects of semantic and outlier detections that are not reflected in the previous tree structures (R-Tree and its variants). This paper explores a novel SSR^{O}-Tree on SAM to the Topo-Semantic approach. The paper shows how to identify and handle the semantic spatial objects with outlier objects during page overflow/underflow, using gain/loss metrics. We introduce a new SSR^{O}-Tree algorithm which facilitates the achievement of better performance in practice over algorithms that are superior in the R*-Tree and R^{O}-Tree by considering selection queries.

Keywords**—**Outlier, semantic spatial object, spatial objects, SSR^{O}-Tree, topo-semantic.

I. Introduction

THE naïve solution to answer any proximity problem (in any space) is the brute force approach that compares every object to every other object *$O(n^2k)$* which is clearly unacceptable for few points. To allow for faster searching, objects must first be sorted somehow into some type of data structure. Such a structure is called an index or spatial access method (SAM). Here, *k* is the dimension of vector space, *n* is the number of points, and O is the Big O notation. The goal of spatial access methods is to organize spatial data in such a way that it will enable the efficient retrieval of relevant objects according to the topological properties of their spatial attributes. Solving a proximity problem using a SAM is typically divided into two phases:

1) *Building the index:* For each SAM, there may available several indexing algorithms to initially construct the structure. The algorithms to insert and delete objects at a later stage may be different again. For example, [26] describes how different reinsertion policies and metrics can be used in three common variations of the R-Tree (see Fig. 1) which is a popular tree-based SAM.
2) *Executing queries by searching the index:* For each SAM there are several search algorithms that answer the various proximity problems. For example, algorithms to execute nearest neighbour and range search are usually quite different [21].

Coarsening Results: To improve performance, many search algorithms techniques have been used that approximate/coarsen their results for spatial queries; especially for queries in non-vector, and metric space [5]. Instead of returning an exact answer to a spatial query, they initially return a set of candidate elements i.e. actual results ⊆ candidate elements. For such indexes, the executing of each query is divided into two additional phases which are depicted in Fig. 1.

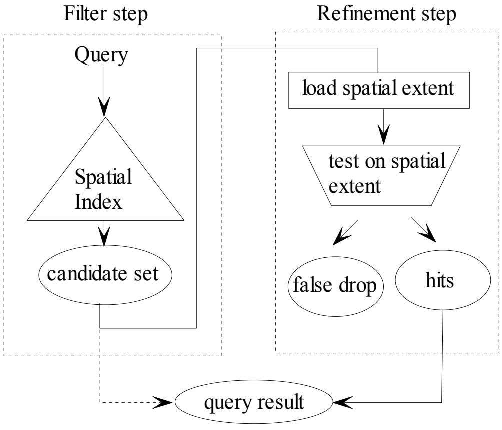


Fig. 1 Multi-step semantic spatial query processing [3]

1) Filter step: searching for a set of candidate elements. The time this takes is called internal complexity.
2) Refinement step: Checking candidate elements are exhaustively used for the required condition or relation. The time it takes is called external complexity. The more candidate elements returned (the more false objects which must be eliminated) the higher the external complexity.

Manuscript received February 28, 2007. This work was supported in part by the by the High Technology Research and Development Program of China (No. 2006AA01A103) and the National Natural Science Foundation of China (No. 60503015).

K .P. Udagepola is with the School of Computer science and Technology, Harbin Institute of Technology, Harbin, PR China.(phone: +86-45186400863; fax: +86-45186414093; e-mail: kalum@ ftcl.hit.edu.cn).
Z. Decheng is with the School of Computer science and Technology, Harbin Institute of Technology, Harbin, PR China (e-mail: zdc@ ftcl.hit.edu.cn).
W. Zhibo is with the School of Computer science and Technology, Harbin Institute of Technology, Harbin, PR China (e-mail: wzb@ ftcl.hit.edu.cn).
Y. Ziaoxong is with the School of Computer science and Technology, Harbin Institute of Technology, Harbin, PR China (e-mail: xzyang@ hit.edu.cn).

The coarsening search results can reduce internal complexity, but typically increases external complexity in the process. Thus, optimizing spatial queries often involves finding an appropriate trade-off between internal and external complexity.

Measuring Performance: In order to compare different solutions to proximity problems, it is necessary to use some measure of performance. This can be non-trivial. Performance can be divided into time performance and memory space requirements. According Chavez *et al.* [5], total time (T) to evaluate a query can be separated into:

$$\begin{aligned} &T = dis\tan ceComputations \times complexityof\_d() + \\ &extra\ CPU\ time + I/O\ time \end{aligned} \tag{1}$$

Many SAM and search algorithms have been proposed to solve proximity problems [7]. These strategies have been validated and compared using various platforms, different testing methodologies, data sets and implementation choices. The lack of a commonly shared performance methodology and benchmarking makes it difficult to make a fair comparison between these numerous techniques [17]. Different papers have used different performance measures. Earlier papers, such as [21], have used the number of page accesses as their main performance measure (probably because main memory capacity and speed played a much larger factor in the past), while other papers [2],[9] prefer to use total CPU time, and still others use the number of I/O accesses [18]. For searches in metric space, it has been recently accepted that the number of distance computations is an appropriate measure of performance, since each metric distance computation is typically expensive [5]. A comprehensive cost model for query processing, with focus on high dimensions is provided in [4]. To the currently available literature, no one has attended one shot solution to semantic spatial quarries with detection of outlier objects for SAM to Topo-Semantic approach and Geographical Information system (GIS) for their efficiency maintenance. This research also focuses on research findings from [6],[27],[28]. Chen *et al.* [6] discussed semantic objects to the web base applications and Xia *et al.* [27] discussed Outlier object separation on the R*-Tree.

The rest of the paper is organized as follows: in section II we provide taxonomy of popular dimension based indexes for vector space and discuss about Topo-Semantic concept to spatial relation integrity constraints. In section III we define novel SSR^{O}-Tree with many required algorithms. In section IV we focus on the performance evaluation SSR^{O}-Tree and other popular close tree structures. Finally, section V contains the concluding remarks.

II. Search Solutions for Vector Space

Over the past twenty-five years, many different indexes have been proposed and investigated [7], [25], [28]. However, there is no consensus on the best spatial index structure is found. According to Gaede *et at.*[7] observations, these can be split into the following two major categories:

1) *Dimension-based indexes:* indexes proposed specifically for vector space, which use distances along dimensions in their indexing of objects, and
2) *Distance-based indexes:* indexes proposed for the more general metric space, which only use distances between points to index objects.

This paper will only consider the dimension base indexing method for the SAM. Fig. 2 shows many spatial data structures.

Trees is an obvious choice, because most trees have an *O(n log n)* build, occupy *O(n)* space, and have *O(log n)* search time (per element searched). All index structures presented here are dynamic, and allow inserting and deleting operations in *O(log n)* time. Most of the trees divide space into hyper-rectangles, which will be called *cells*. During the construction of all the trees, search space is split recursively using fan-out *f* until each cell contains at most minimum elements. Here, *k* is not included in any Big O notations. Since co-ordinate information must be stored and retrieved for all indexes, it is important to remember that all build and search times are (at least) linearly dependent on *k* [15].

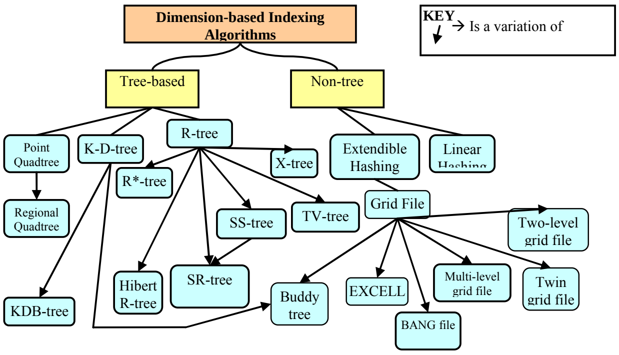


Fig. 2 Simplified taxonomy of popular dimension based indexes for vector space [11],[28]

However, the R-tree [9] and its variant the R*-tree [11] are widely implemented and have found their ways into commercial systems [1], [12], [14] that allows efficient accesses to the spatial data in a GIS system which is based on a heuristic optimization [8], [10], [13], [17]. Our discussion focuses on exploiting R-Trees for the efficient processing of spatial joins using Spatial Data Access Method (SAM). Previous approaches for processing joins are based on access methods others than R-Trees or address the problem in a quite different context. Orenstein [16] has proposed B+-Trees in combination with z-ordering as the underlying access method for performing spatial joins. For a class of Tree structures, Günther [8] has presented a general model for estimating the cost of spatial joins. In a different setting, Rotem [19] has discussed the number of facts of spatial join indices.

Servigne *et al.* [22] introduced concept of Topo-Semantic and Udagepola *et al.* [23] developed a model for MSRIC (Model of Spatial Relation Integrity Constraint) but they have not given any solution to the semantic spatial object for the SAM. They only considered normal data structure of R-Tree (and variants). Udagepola *et al*. [23] have developed Topo-semantic integrity constraint rules (TSICRs). Those rules are used to find specified objects according to their relations to the base object as semantic spatial join queries. For an example, the semantic spatial join query is "Find a Building on the Roads within relationship Overlap." In this query, the base object is query identification number the specified object (i.e., the semantic object) is "Building," and the relationship is "overlap". On the other hand, a query such as "Find an object on the Roads within relationship Overlap," where no attribute of this object is specified, is a general spatial query. In comparing these two queries, a semantic spatial object is defined as an object with some information that a user asks for. In the above example, the system automatically wants to find a Building. This "Building" is one type of semantic spatial information a user may specify. If a spatial data structure can be built with this type of information, query performance can be greatly enhanced. Another problem is that our set of spatial data does not spread as symmetrically in the real world. For example, some regions have closely spread semantic spatial objects and some place exits very far distances between each other. If we use R-Tree (or its variant R*-Tree) structure the time for spatial access, it is more time consuming and the internal time complexity obtains are much higher. Xia *et al.* [27] introduced outlier handling technique with R*-Tree to help identify such kind of objects. We can then separate those objects with other normal semantic spatial objects because our research focuses on real world objects. Our tree structure adopts that technique for the semantic spatial objects, therefore the data structure called Semantic Spatial Rectangular outlier Tree (SSR^O-Tree) is proposed for the MSRIC.

III. SSR^O-Tree Structure

A spatial data structure with built-in semantic spatial information is better able to answer semantic spatial join queries. In addition, it needs to identify outlier semantic spatial objects. For this purpose, a spatial data structure with built-in semantic information with facility for detection of outlier objects, called a SSR^O-Tree, is proposed. The following way is a brief our SSR^O-Tree (see the Fig. 3): Semantic R-Tree [6] + Outlier R*-Tree (R^O-Tree) [27] → Semantic spatial-object outlier R-Tree (SSR^O-Tree) [24].

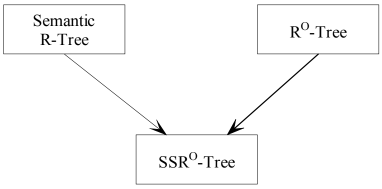


Fig. 3 The taxonomy of SSR^O-Tree

Fig. 4 shows overview of SSR^O-Tree which can be used with the SAM.

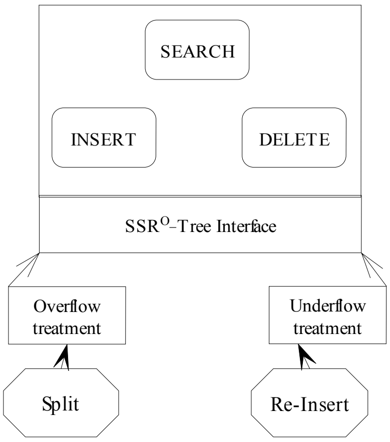


Fig. 4 Overview of SSR^O-Tree architecture

Without such built-in semantic information, a spatial data structure has difficulty in answering a query such as "*Find all Buildings on the Roads within relationship Overlaps*" efficiently. Searching the SSR^O-Tree will get all the objects on the Roads within relationship Overlaps and further processing is then required to get the desired objects (i.e., the Buildings) in answer to this query. With built-in semantic information, some sub-trees containing unrelated information can be pruned, which makes semantic searching quite efficient. In our design, the semantic spatial object class (Example: the semantic spatial object is a road and its subclass is a highway, street cycle path…etc.) is used to build a part of SSR^O-Tree and another part considers detection of outlier objects within the class. The algorithm used is based on R*-Tree (Because it has well re insert capabilities and minimum overlapping). For each node, its semantic information is assorted and organizes the semantic spatial objects that detect outlier objects (outlier identification) and divide small minimum bounding rectangle (MBR*)* so that a search will visit as few spatial objects as possible before returning the result. The decision on which nodes to visit is made based on the evaluation of spatial predicates. In addition, the MBRs are sorted on the x or y coordinates of one of the corners of the rectangle. Sorting MBRs is similar to the method proposed by Roussopoulos and others [20]. In each class, the most of the time is depicted on the levels one and the rest of level are shown on the same category node which satisfies m and M. Finally, leaf node has categorized objects that make the scan very simple. But there might be some underflow nodes (less than $M/2$ children). Since only a fixed number of elements exist in one semantic subset (usually this number is small), there might be only a few underflow nodes. The outlier identification is integrated throughout the construction/update of the SSR^O-Tree, e.g. in the reinsertion process, in the overflow/underflow handing, splitting etc. An SSR^O-Tree satisfies the following properties:

1) Every leaf node contains between m and M index records and outlier objects unless it is the root, but the root can have less entry than m.
2) For each index record in a leaf node, it is the smallest

rectangle that spatially contains the n-dimensional data object represented by the indicated tuple.

3) Every non-leaf node has between *m* and *M* children and outlier objects unless it is the root.
4) For each entry in a non-leaf node, it is the smallest rectangle that spatially contains the rectangles in the child node with/without outlier object/s.
5) The root node has at least two children unless it is a leaf.
6) All leaves appear on the same level. That means the tree is balanced.

Variable m and M: M is the maximum number of entries which is usually given and *m* is the minimum of number entries in one node.

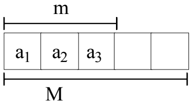
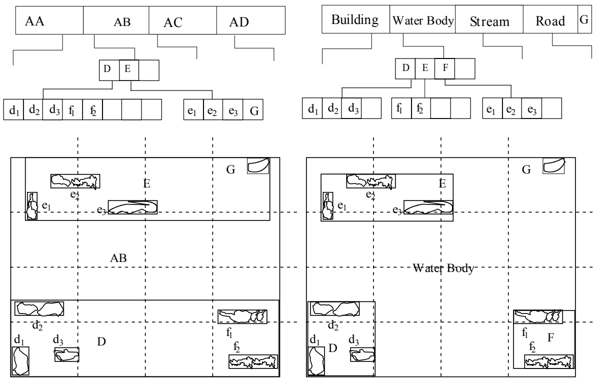


Fig. 5 Graphical representations of m and M

The minimum number of entries in a node is dependent on *M* with $\frac{M}{2} \geq m$. The maximum number of nodes is $\frac{N}{m} + \frac{N}{m^2} + 1$. Here *N* stands for the number of index records of the R-Tree. *m* is jointly responsible for the height of an R-Tree and the speed of the algorithm. The choice of *M* depends on the hardware, especially on hard disk properties such as capacity and sector size. If nodes have more than 3 or 4 entries, the tree becomes very wide, and almost all the space is used for leaf nodes containing index records.

The example: A Topo-Semantic spatial object (indexed by Semantic R*-Tree and SSR^o-Tree) under Road sub class is depicted on the Cartesian space (see Fig. 6).



(a) (b)

Fig. 6 The page layout of Water Body (a) All spatial Object (No subdivision to semantic object subclasses) Indexed by R*-Tree (b) indexed by SSR^O-Tree

The (a) and (b) of Fig. 6 show an example of Road objects and how to index them by R*-Tree and SSR^O-Tree. The R*-Tree is not capable of separating semantic objects class and outlier object (G) but SSR^O-Tree has that facility. In the above example, we can see the root node is also split. Therefore, our motivation to the SSR^O-Tree is that if an object G is far from all the rest of the objects, in R*-Tree it is inevitable that the leaf page that contains *G* will have a large MBR, and consequently all ancestor nodes of the leaf page also have large MBRs.

Quality and Gain/Loss:

Definition 1[28]: Given a rectangle *r* with width w and height *h*, the quality of the rectangle is defined as

$$Q(r) = \frac{1}{w \times h} \times \left(\frac{min\{w,h\}}{max\{w,h\}} \right)^{\alpha} \qquad (2)$$

Where $\alpha \in [0, 1]$ is a constant.

But $\left(\frac{min\{w,h\}}{max\{w,h\}} \right)^{\alpha} \leq 1$, therefore *Q(r)* is dependent only $\left(\frac{1}{w \times h\}} \right)$. But $(w \times h)$is area of the rectangle. Then the small rectangle obtained is of good quality.

Assume w $\geq$ h; where w and h are called width and height respectively.

Let $ratio = \frac{w}{h}$, Therefore $area = \frac{ratia^{-\alpha}}{Q}$

Definition 2: If a rectangle r_1 is shrunk to r_2 (r_1 spatially contains r_2), the gain is defined as

$$G(r_1, r_2) = 1 - \frac{Q(r_1)}{Q(r_2)} \qquad (3)$$

Therefore if r_1 *is* expanded to r_2, then the loss is created at the $\frac{Q(r_1)}{Q(r_2)} > 1$.

Now, the threshold (δ) can be defined because the new rectangle does not need to be very close to previous one. Therefore Gain can be limited to very small value (eg. 0.001).

Theorem1: A successful new rectangle needs satisfy $G(r_1, r_2) > \delta$.

This research also uses four lines method [28] to build a new MBR. The four lines can be defined by their properties according to Fig. 7.

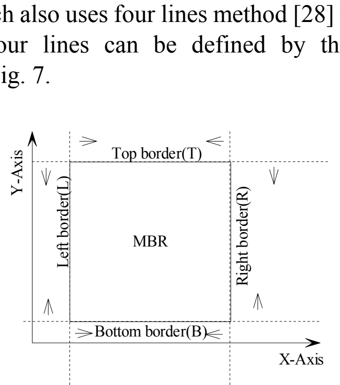


Fig. 7 Four lines on a MBR

The algorithm to handle the outlier object detection is shown in algorithm 1. It is adapted from the R^O-Tree's greedy-pick-p algorithm.

Algorithm 1: pseudo-code of *Greedy-pick-p* Algorithm

```
Greedy-pick-p (S ∈ SpatialObjects , p,m: integer): MBR
1.  For each category
2.  Build the border structure of initial MBR.
3.  MBR(S)=M;
4.  P =0 ;.
5.  while {
6.  $g_{max}$ = 0;
7.  $B_{max}$ = LEFT
8.  $k_{max}$ = 0;
9.  For each border /* LEFT,BOTTOM,RIGHT,TOP*/
10. For each k /* k ∈ {1,..m} such that there exist k more un
    removed levels for B and the number of objects in these k
    levels plus |P| is no larger than p*/
11. Compute the g (gain) per removed object;
12. If g > gmax then
13. gmax = g;
14. $B_{max}$ = B;
15. $k_{max}$ = k;
16. end if/* end step 12*/
17. end for/* end step 10*/
18. end for/* end step 9*/
19. If $g_{max}$ = 0 then stop;
20. Adjust new MBR according border $B_{max}$ and add the
    removed objects to P (outlier object list)
21. }/* end step 1*/
22. Endfor /* end step 5*/
23. End /* end Greedy-pick-p */
```

Search: Search algorithm accomplishes the following task: given an SSR^O-Tree whose root node is *T*, find all index records whose rectangles overlap a search rectangle *S*. C is a subset of the semantic object class. An entry is denoted in a node as *E(Ref, Mbr)*, where *E.Mbr* represents the smallest rectangle bounding the sub-tree or the spatial object, *E.ref* is the pointer to the sub-tree or the spatial object.

Algorithm 2: pseudo-code of *SearchSubTree* Algorithm

```
SearchSubTree(C,T, S)
1.  If  T is under C category then
2.  If T is not a leaf then
3.  for E ∈ T do
4.          if E.MBR ∩ S.MBR ≠ Ø  then
5.  if E is outlier object then
6.  output E
7.  else search(E,S)
8.  Endif/* end step 5*/
9.  else
10. read next entry of t /* recursive call to lower level */
11. Endif/* end step 4*/
12. end for/* end step 3*/
13. else
14. if T.MBR ∩ S.MBR ≠ Ø then
15. output T
16. Endif/* end step 14*/
17. Endif/* end step 2*/
18. Endif/* end step 1*/
19. End. /* end Search*/
```

Insert: The insertion algorithm adds new index records to the leaves under the same category. Nodes that overflow are split and the reinsert changes propagate up the tree. The first operation is to select a category where the subset of insert is the object and check it to see if it's either an outlier or not. If it is an outlier then the object is placed as an outlier object otherwise select a leaf where to place the new record at minimum loss. However, there is the possibility that an overflow situation can occur. This problem is solved by the *overflow Treatment* algorithm.

Algorithm 3: pseudo-code of *Insert* Algorithm

```
Insert (new O object under C category)
  /* new object O will be inserted into a given $SSR^O$-Tree */
1.  If  O is under C category then
2.  Start to select a leaf node L in which to place O /*Find
    position for new record*/
3.  If O is not contained in any MBR of tree then
4.  Stored in top node
5.  Else
6.  If O is contained a MBRs and has enough space then select
    smallest MBR and add O.
7.  Else
8.  If O is contained a MBR and it has enough space(not exceed
    M) then
9.  amalgamate  into the MBR
10. Else
11. invoke overflowTreatment
12. End if/* end step 8*/
13. End if/* end step 6*/
14. End if/* end step 3*/
15. End if/* end step 1*/
16. End/* end Insert*/
```

Delete: The deletion algorithm deletes existing index records from the leaves under the same category. Nodes that underflow are the reinsert changes that propagate up the tree. The first operation is to select a category where the subset of deletion object will be checked whether it is an outlier or not. If it is an outlier, then the object is deleted from the outlier object otherwise be selected as a leaf where and deleted it with adjust *MBR* according maximum gain. After this step an underflow (less than *m* entries) can occur. This problem is solved by the *underflowTreatment* algorithm.

Algorithm 4: pseudo-code of *Delete* Algorithm

```
Delete (O object under C category)
    /* an object O will be delete from a given SSR^O-Tree */
1. If pointer is  in related category(/*  O is under C category */)then
2. Start to select a leaf node L in which place available O /*Find position for required record*/
3. If O is not contained in any MBR of tree then
4. stop
5. Else
6. If O is contained a MBR and it possible to remove without satisfaction of underflow condition then
7. Remove the entry from the MBR and adjust MBR area using maximum gain
8. Else
9. invoke  underflow Treatment
10. End if/* end step 6*/
11. End if/* end step 3*/
12. End if/* end step 1*/
13. End/* end Delete*/
```

Overflow Treatment: The overflow (see Fig. 8) treatment algorithm will first check whether it is the first time an overflow occurs in the current level. If so, then it will try to avoid a split by pushing down an outlier object from a parent node or reinserting some of the *M+1* entries of the overfilled node. Otherwise, it will just invoke the split procedure.

.

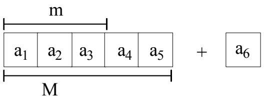


Fig. 8 Representation of overflow

The algorithm to handle the overflow is shown in algorithm 2. It is adapted from the R^O-Tree's Overflow Treatment algorithm.

Theorem 1: SSR^O-Tree can overflow either new object insert to the non leaf node or leaf node.

Algorithm 5: pseudo-code of *OverflowTreatment* Algorithm

```
OverflowTreatment()
1. If (M+ 1) entries include only index entries or M+ 1 entries include index entries and outlier object and reinsert possible then
2. Invoke reinsert
3. Else
4. If  $num\_of\_children < m + \frac{M}{2}$  and outlier object available then
5. Pushdawn outlier object
6. Else
7. Invoke Split
8. Endif/* end step 4*/
9. Endif/* end step 1*/
10. End. /* end overflowtreatment*/
```

Split: The node splitting uses a routine for partitioning a region containing a set of rectangles into two sub-regions that satisfy a maximum gain. In the case of node splitting this routine takes as input the MBRs of the entries in the node (N). If all the *M+1* entries are index entries, the only choice is to split N. To the other extreme, if N has less than *2m* index entries, splitting is not a choice. Otherwise one of the resulting pages will violate the minimum fan-out requirement. Xia et al.[27] experimental results show (for R^O-Tree) that a good breaking point is $m + M=2$. That is, if num of $children(N) < m + M=2$, push down an outlier, it will otherwise split. To push down an outlier object, we choose the sub-tree which has minimum loss to accommodate the new object. The way of these also is followed to SSR^O-Tree.

Algorithm 6: pseudo-code of *Split* Algorithm

```
Split()
1. Compute the distance between the centers of their rectangles and the center of the bounding rectangle of N and identify longest distance two MBR.
2. Start the create new MBR from one corner with satisfy minimum loss until total number of object entries m+1
3. Check outlier objects availability in this New created MBR and adjust according either outlier object and small MBR or minimum loss MBR
4. Start the create new MBR from other corner with satisfy minimum loss until touch previous created MBR
5. Check outlier objects availability in this New created MBR and adjust according either outlier object and small MBR or minimum loss MBR
6. Insert this two seed to parent node
7. End. /* end split*/
```

Underflow Treatment: The underflow (Fig. 9) treatment algorithm will first check whether or not it is the first time an overflow occurs in the current level. If so, then it will try to avoid a split by pushing down the outlier object from parent node or reinserting some of the *M+1* entries of the overfilled node. Otherwise, it will just invoke the split procedure.

.

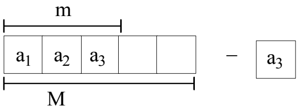


Fig. 9 Representation of underflow

The algorithm to handle the overflow is shown in algorithm 2. It is adapted from the R^O-Tree's Underflow Treatment algorithm.

Algorithm 6: pseudo-code of *UnderflowTreatment* Algorithm

```
UnderflowTreatment(underflow leaf node L)
1. N = L;
2. S =0;
3. If N is root, re-insert all entries in S into the tree (Notice that if an entry is an index entry, it should be inserted into a proper level);

4. If N is a leaf node and there exists an outlier object O in N's parent node which could be pushed down into the entry pointed to N then Insert O into N and adjust its MBR accordingly
5. else
6. invoke reinsert;
7. End if/* end step 4*/
8. If N is an index node then
9. While num of children(N) < m and N contains some outlier object{
10.     Find an outlier O in N which, if pushed down to a lower level, introduces the minimum loss;
11.     Insert O to the next lower level;
12.     Propagate and adjust the subtree rooted by N properly;
13.     }/*end while*/
14.     End if/* end step 8*/
15.     If N has less than m entries then
16.     Remove the index entry $E_N$ in the parent node P;
17.     Add N's entries to S;
18.     N = P
19.     Goto step 3;
20.     End if/* end step 15*/
21.     End./*end UnderflowTreatment */
```

Re-Insert: The reinsertion consists of an algorithm for determining the list of entries that are to be removed from the current node and reinserted in the tree. The number p of entries in the list is a parameter that can be determined experimentally as part of performance tuning. For all *M+1* entries, the distances d_i between the centers of their MBRs and the center of the MBR of the node is calculated. Then, the entries are sorted in descending order of the values d_i. The first *p* entries on the sorted list are removed from the current node and its MBR is adjusted in the parent node. The *p* entries are the list of entries to be reinserted in the tree. In [2] it has been determined that reinsertion in the decreasing order of the distances d_i (close reinsert) outperforms reinsertion in the ascending order (far reinsert).

Algorithm 7: pseudo-code of *ReInsert* Algorithm

```
ReInsert()
1. For all M+l entries of a node N, compute the distance between the centers of their rectangles and the center of the bounding rectangle of N
2. Sort the entries in decreasing order of their distances computed in step1
3. Remove the first p entries from N and adjust the bounding rectangle of N
4. In the sort, defined step 2, starting with the maximum distance (= far reinsert) or minimum distance (= close reinsert), invoke Insert to reinsert the entries
5. End. ./*end re insert */
```

IV. Selection Query Performance

We used three data structures to find the spatial objects of the building subclass where the area of the object is less than or equal 100m^2 on the different size of rectangles. Table I shows the information which is used for the experiment. The experiment has been done on three types of tree structures because the performance evaluation test needs to compare SSR^O-Tree with most famous tree structures.

TABLE I
Number of Spatial Objects in Eleven Size of Rectangle

Total Spatial Objects of Building subclass	Total Spatial Objects of Path subclass	Total Spatial Objects of Landuse subclass	Total Spatial Objects	After selection query execute (Find number of Buildings)
179	13	196	388	74
628	42	459	1129	250
1030	52	750	1832	375
2171	84	981	3236	738
3540	113	1413	5066	1468
5274	129	1827	7230	2189
6228	163	2188	8579	2535
8858	193	2563	11614	3497
10366	219	2889	13474	3983
11216	271	3174	14661	4163
12036	301	3531	15868	4494

The experiment uses Spatial database (SD) which contains three type of semantic spatial objects (Building, Path, and Landuse).We have used 11 data sets for the selection query (select * from SD where building area <= 100m^2). Fig. 10 shows the significant deference of the SSR^O-Tree among the other two trees. It is seen that the results of I/O time are always less than that of the other two trees. Fig. 10 shows a higher response time of R*-Tree for finding while satisfying the requirements but SSR^O-Tree shows the opposite response because it has two facilities which are semantic object categorization and outlier detection. But R^O-tree has only outlier separation facility that performs better when compared with R*-Tree but shows less performance when compared with SSR^O-Tree. After the number of objects becomes 5066 in R*-Tree it has shown a significant performance (increased gradient but rest of the tree shows a less gradient) because those rectangles contains more outlier objects. According to the results of the experiment, the better performance of SSR^O-Tree over the other two trees is obvious.

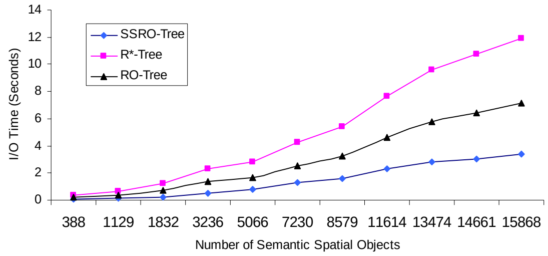


Fig. 10 I/O time of finding objects according a building object's area <=100m^2 under SSR^O-Tree, R^O-Tree, and R*-Tree

V. Conclusion

A novel SSR^O-Tree was created in this research by using some properties of Semantic R-Tree and R^O-Tree because this model needs an efficient semantic spatial object access method that guarantees the better performance. Eight new algorithms have been described with proper notations. Semantic spatial selection query performances were discussed and the experimental results revealed that that the SSR^O-Tree outperforms over R^O-Tree and R*-Tree in the I/O time.

Acknowledgment

The authors extend sincere gratitude to Dr. Li Xiang, Mr. A.W. Wijeratne, Dr. Liu Hongwei and Miss Bethany LoPiccolo.

References

[1] N. An, K. Kanth, and S. Ravada, "Improving Performance with Bulk-Inserts in Oracle R-Trees," In *VLDB*, 2003.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," In *Proceedings of the ACM SIGMOD international conference on Management of data*, pp.322-331, 1990.

[3] T. Brinkhoff, H. Kriegel, and R. Schneider, "Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems," *IEEE*, 1993.

[4] C. Böhm, "A cost model for query processing in high-dimensional data spaces," *ACM Transactions on Database Systems,*vol. 25, no. 2, pp.129-178,2000.

[5] E. Chavez, G. Navarro, R. Baeza-Yates, and J. Marroquin, "Searching in metric spaces," Technical Report TR/DCC-99-3, Dept. of Computer Science, Univ. of Chile, 1999.

[6] S. Chen, X. Wang, N. Rishe, and M.A.W. Weiss, "A Web Based spatial data access system using semantic R-Trees," *Elsevier Science*, 2003.

[7] V. Gaede, and O. Günther, "Multidimensional access methods," *ACM Computing Surveys*, vol. 30, no. 2, pp.170-231,1998.

[8] Günther, O.: 'Efficient Computations of Spatial Joins', Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993.

[9] A. Guttman, "R-trees: A dynamic index structure for spatial searching," In *proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.47-57, 1984.

[10] I. Kame, and C. Faloutsos, "On Packing RTrees," *CIKM*, pp. 490-499, 1993.

[11] K. V. R. Kanth, A. E. Abbadi, D. Agrawal, and A. K. Singh, "Indexing Non-Uniform Spatial Data," In *Proceedings of International Database Engineering & Applications Symposium,*1997.

[12] K. V. R. Kanth, S. Ravada, J. Sharma, and J. Banerjee. "Indexing Medium-dimensionality Data in Oracle," In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, pp.521–522, 1999.

[13] K. V. R. Kanth, S. Ravada, and D. Abugov, "Quadtree and R-tree Indexes in Oracle Spatial: A Comparison using GIS Data," In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, pp. 546–557, 2002.

[14] S. T. Leutenegger, and M. A. Lopez, "The Effect of Buffering on the Performance of R-Trees," *International IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 1, pp. 33-44, 2000.

[15] A. Noske, "Dynamic Range Queries in Vector Space," http://www.andrewnoske.com/professional/publications/Lit_Review_-_Dynamic_Range_Queries_in_Vector_Space.doc.

[16] Orenstein J. "A.: 'Spatial Query Processing in an Object-Oriented Database System," *Proc. ACM SIGMOD Int. Conf. on Management of Data,* pp. 326-333, 1986.

[17] A. Papadopoulos, P. Rigaux, and M. Scholl, "A performance evaluation of spatial join processing strategies," *Proceedings of the 6th International Symposium on Advances in Spatial Databases*, pp.286-307,1999.

[18] S. Prabhakar, Y. Xia, D. V. V. Kalashnikov, W. G. G. Aref and S. E. E. Hambrusch, "Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects," *IEEE Transactions on Computers archive,* vol. 51, no.10, pp.1124-1140, 2002.

[19] D. Rotem,"Spatial Join Indices," *Proc. Int. Conf. on Data Engineering*, pp. 500-509,1991.

[20] N. Roussopoulos, D. Leifker, Direct Spatial Search on Pictorial Database Using Packed R-Trees, Proceeding ACM SIGMOD, 1985.

[21] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries*," *Proceedings of ACM SIGMOD,* 1995.

[22] S. Servigne, T. Ubeda, A. Puricell, and R. Laurini, "A Methodology for Spatial Consistency Improvement of Geographic Databases," *Geoinformatica*, vol. 4, no. 1, pp. 7-34, 2000.

[23] K.P. Udagepola, L. Xiang, A.W. Wijeratne, and Y. Xiaozong, "MSRIC: A Model for Spatial Relations and Integrity Constraints in Topographic Databases," *5th Int. Conf. on Artificial Intelligence, Knowledge Engineering and Database. Research conference*, 2006.

[24] K.P. Udagepola, L. Xiang, A.W. Wijeratne, and Y. Xiaozong, "Semantic Integrity Constraint Violations Check for Spatial Database," *MMT-2007*, to be published.

[25] J. S. Vitter, "External Memory Algorithms and Data Structures," *ACM Computing Surveys*, vol. 33, no. 2, pp.209–271, 2001.

[26] S. Wang, J. M. Hellerstein, and I. Lipkind, "Near-neighbor query performance in search trees,", http://citeseer.ist.psu.edu/92931.html.

[27] T. Xia, , and D. Zhang, "Improving the R*-tree with Outlier Handing Techniques," *GIS'05*, 2005.

[28] D. Zhang, and T. Xia, "A Novel Improvement to the R*tree Spatial Index using Gain/Loss Metrics," *GIS'04*, 2004.

Kalum Priyanath Udagepola is a Ph.D. candidate at the School of Computer Science and Technology, Harbin Institute of Technology, PR China. His research interests are GIS, Spatial database and Mobile GIS. Zuo Decheng is a Professor and senior lecturer of School of Computer Science and Technology, Harbin Institute of Technology, PR China. His research interests are mobile computing, mobile ad hoc network, Bluetooth technology and Mobile GIS. Wu Zhibo is a Professor and doctoral supervisor of the School of Computer Science and Technology, Harbin Institute of Technology. His research interests are Computer architecture, Mobile computing, Dependable computing, Fault tolerant computing and Mobile GIS. Yang Xiao Zong is a Professor, Director and doctoral supervisor of the School of Computer Science and Technology, Harbin Institute of Technology. His research interests are Computer architecture, Mobile computing, Dependable computing, Fault tolerant computing and Mobile GIS.

Mr. Udagepola is a member of the British Computer society, the Computer Society of Sri Lanka, the Royal Institute of Charted Surveyors and the Surveyors' Institute of Sri Lanka