An - Najah Univ. J. Res. (N. Sc.) Vol. 30(1), 2016

Global Center Point Splitting: New Linear Node Splitting Algorithm for R-Trees

التقسيم حول نقطة المركز العام: خوارزمية خطية جديده لتقسيم عقدة في دليل R-tree لقواعد البيانات

Manar Arafat

منار عرفات

Department of Computer Science, Faculty of Engineering & Information Technology, An-Najah National University Nablus, Palestine

Email: manar_arafat@najah.edu

Received: (4/4/2015), Accepted: (21/12/2015)

Abstract

We introduce a new linear algorithm to split overflowed nodes of an R-tree index called the Global Center Point Splitting (GCPS) algorithm. The proposed method is an enhancement of the Quadratic splitting algorithm proposed by Guttmann (Guttman A, 1984; 47–57). Most known algorithms do not take advantage of the fact that most spatial objects data is known beforehand, and these objects are relatively easy to identify. In this paper we have adopted an informative approach by making use of spatial information provided by the problem space. Objects in the problem space are scanned and the Global Center Point (GCP) that the objects are concentrated around is determined. The GCPS algorithm uses the proximity between the Global Center Point (GCP) and the remaining objects in selecting a splitting axis that produces the most even split. We conducted several experiments using both real and synthetic data sets. Results show that the proposed splitting method outperforms the quadratic version in terms of construction time especially for nodes with high capacity. The query performance approximately remains the same.

Keywords: R-Tree index, spatial databases, query performance.

ملخص

في هذا البحث سنقوم بتقديم خوارزمية خطية جديده لتقسيم عقدة ممتلئة في دليل R-tree لقواعد البيانات تسمى خوارزمية التقسيم حول المركز العام (GCPS). هذه الخوارزمية هي تحسين للخوارزمية ذات الدرجة الثانية التي تم طرحها من قبل Guttmann (Guttman, A. 1984; 47–57). معظم الخوارزميات لا تستفيد من حقيقة ان معظم البيانات ثنائية الفضاء معروفة مسبقا. في هذا البحث سنستفيد من هذه البيانات في ايجاد نقطة المركز العام (GCP) التي تتجمع حولها هذه البيانات، وبناء على تقارب البيانات مع هذه النقطة, تقوم خوارزمية (GCPS) بتحديد المحور الامثل لتقسيم البيانات. لقد قمنا باجراء اختبارات للخوارزمية الجديدة باستخدام بيانات حقيقية وبيانات مصطنعة. النتائج تشير الى ان الخوارزمية الجديدة مقارنة -بالخوارزمية ذات الدرجة الثانية- تعطي نتائج افضل. حيث ان وقت بناء دليل قاعدة البيانات يصبح اقل مع المحافظة على نفس الاداء.

كلمات مفتاحية: دليل R-tree، قواعد البيانات المكانية، اداء الاستعلام.

1 Introduction

The real-world spatial databases are very large, with sizes can reach to millions of objects. The spatial indexes are used to quickly access spatial database systems. Efficient processing of queries manipulating spatial relationships between objects in the database depends on the spatial index used. R-tree is the most widely used structure for indexing multi-dimensional information (Guttman, A. 1984; 47–57).

R-Trees can organize any-dimensional data by representing the data by a minimum bounding rectangle (Papadias, D. & *et al.* 1995). Each node bounds its children. A node can have many objects in it. The leaves point to the actual objects (stored on disk probably). Objects are added to the node that will get the least enlargement in its MBR size to minimize node coverage. It must be noted also that the MBRs may overlap; consequently, the search operation will visit more nodes before getting a result. Efficient R-trees require minimizing both node overlap and coverage.

The insertion of a new entry to a full leaf node in R-tree requires node splitting. Node splitting is a critical process for the overall performance of the access method since it determines the final shape of the structure (Fu, Y. & *et al.* 2002; 766–770); (Sleit, A. 2008; 711–720);

(Sleit, A. Al-Nsour, E. 2014; 222–236). An efficient splitting algorithm may be used to achieve different goals, such as reducing the R-tree index creation time, and producing an R-tree structure which has the minimum overlapping of the MBRs and minimum total coverage of the MBRs.

In literature, a number of R-tree structure variants present (Sellis, T. & *et al.* 1987; 507–518); (Beckmann, N. & *et al.* 1990; 322–331). and several splitting algorithms have been proposed (Guttman, A. 1984; 47–57); (Sleit, A. Al-Nsour, E. 2014; 222–236); (Ang, C. H. & Tan, T. C. 1997; 15–18); (Al-Badarneh, A. & *et al.* 2010; 3–18); (Al-Badarneh, A. & Tawil, M. 2009).

In this paper we focus on the Quadratic node splitting algorithm proposed by Guttmann (Guttman, A. 1984; 47–57). A pair of objects is selected from the over-flown node as the seeds of the split. These are picked to be the pair of objects that if put in the same node will create the largest empty space. Each of the two nodes resulted from the splitting will contain exactly one seed. Each of the remaining objects is inserted to the nearer node (i.e. the node that requires smaller enlargement of its MBR). The requirement of the min/max number of objects per node must be satisfied. It is obvious that the goal of minimizing the total coverage may be compromised.

In this paper we present a new enhancement to the Guttmann's quadratic node splitting algorithm. We propose a new algorithm upon which the pick seeds operation is done. This new pick seeds algorithm will cause a linear cost when compared with the original quadratic cost algorithm. Most known algorithms do not take advantage of the fact that most spatial objects data is known beforehand, and these objects are relatively easy to identify. In this paper, we have adopted an informative approach by making use of spatial information provided by the problem space. The objects in the problem space are scanned and the point that the objects are concentrated around is determined. This point is used to determine the axis of splitting. We will show that when using the new splitting algorithm in building the index, the index creation/update time will be reduced and the query performance remains nearly the same when compared with the quadratic version. We think that the new technique would be very efficient for the most recent DB systems which in general have high query frequency.

The rest of this paper is organized as follows. Section 2 describes the new algorithm, section 3 will provide some testing and comparison results. Section 4 is the conclusion and future work.

2. Global Center Point Splitting (GCPS) Algorithm

The R-tree Quadratic splitting algorithm proposed by Guttmann (Guttman, A. 1984; 47–57) consists of two phases:

Phase 1: a pair of objects is selected from the over-flown node as seeds. These are picked to be the pair of objects that if put in the same node will create the largest empty space. This phase of picking the seeds requires a Quadratic time complexity.

Phase 2: Each of the two nodes resulted from the splitting will contain exactly one seed. Each of the remaining objects is inserted to the nearer node (i.e. the node that requires smaller enlargement of its MBR). The requirement of the min/max number of objects per node must be satisfied. It is obvious that the goal of minimizing the total coverage may be compromised. This phase requires a linear time complexity.

In this paper we present a new enhancement to **phase 1** in the Guttmann's quadratic node splitting algorithm. **Phase 2** will not be changed. We propose a new algorithm upon which the pick seeds operation is done. This new pick seeds algorithm will cause a linear cost when compared with the original quadratic cost algorithm.

The new proposed **phase 1** consists of three parts:

Part1: Determining a global center point of the (M+1) objects of the over-flown node. Each object *MBRi* will have its own center point (x_i, y_i). These points will be used in determining the global center point (x_{gcp}, y_{gcp}) of the whole object (node) space as shown in the following two equations:

$$x_{gcp} = \frac{\sum_{i=0}^{M+1} x_i}{M+1} \text{ and } y_{gcp} = \frac{\sum_{i=0}^{M+1} y_i}{M+1}$$

Part 2: A splitting axis intersecting with the global center point (GCP) is determined. The GCP is used to map the objects into concentration regions by inspecting how their concentration is oriented vertically or horizontally. The axis must split the object space into two highly concentrated areas. Consequently, the splitting axis will be horizontal -passing through the GCP- if more than half of the objects are concentrated to the left or to the right of the GCP. On the other hand, the splitting axis will be vertical -passing through the GCP- if more than half of the objects are concentrated above or below the GCP. Otherwise both vertical split and horizontal split must be considered in Part3.

Part 3: Picking the seeds. The seeds of split will be determined according to the splitting axis orientation. There are three cases:

Case1: If the splitting axis is horizontal, the proposed pick seeds algorithm suggests selecting the farthest object above the splitting axis as the first seed, and the farthest object below the splitting axis as the second seed.

Case 2: If the splitting axis is vertical, the algorithm suggests selecting the farthest object to left of the splitting axis as the first seed, and the farthest object to right of the splitting axis as the second seed.

Case 3: If the splitting axis cannot be determined from part 2, the algorithm proceeds as follows: Let pair1 be the seeds that are picked when the splitting axis is horizontal, and let pair2 be the seeds that are picked when the splitting axis is vertical. Then, calculate the difference in y-value between seed1 and seed2 in pair1, and the difference in x-value between seed1 and seed2 in pair2. Then choose the pair with max difference to be the seeds of the split.

From Part 3, it is obvious that the goal of minimizing the total overlap may be compromised. This algorithm has a linear complexity; the calculations performed in the three parts of the algorithm require linear time complexity.

2.1 The GCPS Algorithm

The MBR of an R-tree node *i* is specified by two points: the lower left corner (x_{l_i}, y_{l_i}) and the upper right corner (x_{h_i}, y_{h_i}). An over-flown node 'N' with 'M + 1' objects should be split into two new nodes N1 and N2. The Pseudocode for the **GCPS** algorithm is formally described as follows:

```
//Calculate the center of each MBR i ( $x_i, y_i$ ) in node N:
For (i=0;i<M+1;i++){   $x_i = \frac{x_{l_i} + x_{h_i}}{2}$ ;   $y_i = \frac{y_{l_i} + y_{h_i}}{2}$ ;}

//Determine the global center point ( $x_{gcp}, y_{gcp}$ ):
For (i=0;i<M+1;i++){   $x_{gcp}$ += $x_i$ ;   $y_{gcp}$ += $y_i$ ;}
 $x_{gcp}$ /= (M+1);   $y_{gcp}$ /= (M+1);

//Setting counters: define four counters to store the number of MBR
centers that are to the left, right, above, below the GCP.
For (i=0;i<M+1;i++){
If  $x_i > x_{gcp}$  increment right_counter
If  $x_i < x_{gcp}$  increment left_counter
If  $y_i > y_{gcp}$  increment above_counter
If  $y_i < y_{gcp}$  increment below_counter
}
```

```
//Setting flags: for each counter, define a flag that is set to be true if the corresponding counter is greater than half the number of nodes.

if(left_counter>(M+1)/2){left_flag=true;}else{left_flag=false;}

if(right_counter>(M+1)/2){right_flag=true;}else{right_flag=false;}

if(above_counter>(M+1)/2){above_flag=true;}else{above_flag=false;}

if(below_counter>(M+1)/2){below_flag=true;}else{below_flag=false;}

//Determine the splitting axis: the seeds will be the farthest two points around the splitting axis.

//Case 1: the splitting axis is horizontal.
if(XOR(left_flag,right_flag)= =1&&(above_flag||below_flag)= =0){
        for (i=0;i<M+1;i++){
            if ( $y_i < y_{gcp}$ ){
    //object is below splitting axis
    //pick the object with max $y_{gcp} - y_i$ to be seed1}
            else if ( $y_i > y_{gcp}$ ){
            //object is above splitting axis
            //pick the object with max $y_i - y_{gcp}$ to be seed2}
        }
    }
```

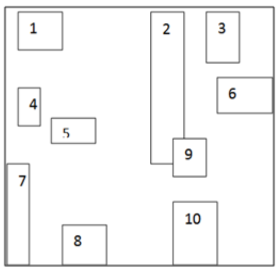
```
//*Case 2:* the splitting axis is vertical.
If (XOR(above_flag,below_flag)= =1&&(left_flag||right_flag)= =0){
    For (i=0;i<M+1;i++){
        if ( $x_i < x_{gcp}$ ){
            //object is to the left of splitting axis
//pick the object with max $x_{gcp} - x_i$ to be seed1}
        else if( $x_i > x_{gcp}$ ){
            //object is to the right of splitting axis
//pick the object with max $x_i - x_{gcp}$ to be seed2}
        }
    }
```

//***Case 3:*** compare $y_{Seed\,2} - y_{Seed\,1}$ when the splitting axis is horizontal with $x_{Seed\,2} - x_{Seed\,1}$ when the splitting axis is vertical, and then choose the splitting axis with max difference.

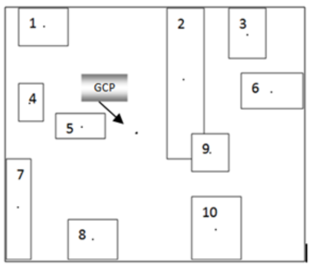
```
if((XOR(left_flag,right_flag)==1&&
XOR(above_flag,below_flag)==1)
||(left_flag==0&&right_flag==0&&above_flag==0&&below_flag==0))
{     //suppose that the splitting axis is horizontal
    d1= $y_{Seed\,2\_horizontal} - y_{Seed\,1\_horizontal}$
//suppose that the splitting axis is vertical
    d2= $x_{Seed\,2\_vertical} - x_{Seed\,1\_vertical}$
if(d1>d2)
//The splitting axis is horizontal, the seeds are the farthest in y value around the axis
else if(d1<=d2)
//The splitting axis is vertical, the seeds are the farthest in x value around the axis
}
```

2.2 Illustrative Example

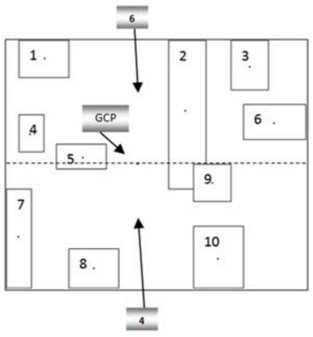
To demonstrate the GCPS algorithm's work mechanism, Figure 1 provides an example.



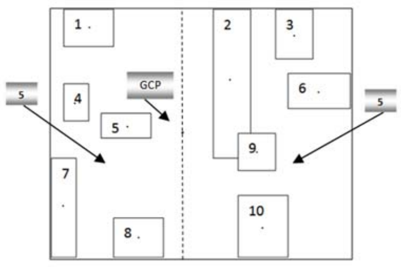
Over flown node N



Determine the Global Center Point (GCP)



Getting the above/below counters



Getting the left/right counters

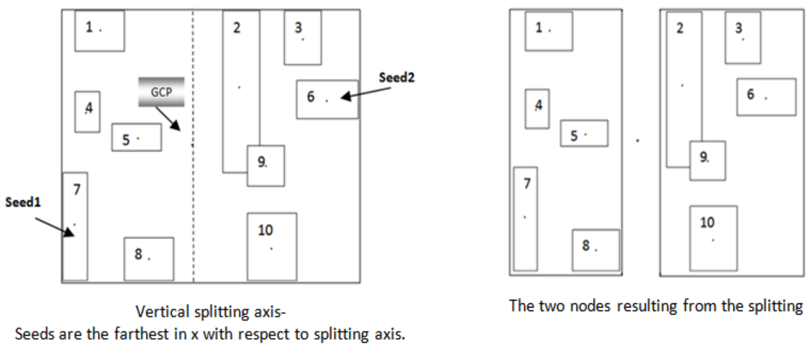

1 .
2
3
GCP
Seed2
6 .
4
5 ·
9.
Seed1
7
10
8 .
Vertical splitting axis-
Seeds are the farthest in x with respect to splitting axis.
The two nodes resulting from the splitting

Figure (1): The GCPS algorithm’s work mechanism: a step by step example.

3. Experiments and Results

The programs used in the experiments are all written in C++ and are run on 1.73 GHz Intel Core i7 machine with 6 GB of RAM running Windows 7. Both synthetic and real world data files were used for testing. We conducted several experiments to test the GCPS algorithm against the Quadratic algorithm. Experiments are grouped in two types: Index creation experiments and Index Query experiments.

3.1 Index creation experiments

The new GCPS algorithm differs from the Quadratic algorithm only in the splitting process of an over-flown node. This will affect the final structure of the tree. The effectiveness of the two index creation algorithm can be measured by the index creation time. In Table 1, five R-trees were generated using n-nodes synthetic data files with Uniform distribution, with n ranging from 10,000 to 100,000 and with Max fill=50 and Min fill=Max/2. The average time taken in creating R-trees is measured when the new GCPS algorithm and the Quadratic algorithm are used.

Table (1): Index creation time using synthetic data sets.

n	Quadratic		New GCPS		Improvement %
	time (seconds)	Number of splits	time (seconds)	Number of splits	
10,000	0.559	293	0.432	293	23%
30,000	1.809	876	1.448	880	20%
50,000	3.340	1478	2.531	1466	24%
80,000	5.377	2339	4.494	2319	16%
100,000	7.056	2939	5.711	2926	19%

In Table 2, two R-trees were generated using real world files (with Max fill=50 and Min fill=Max/2). The average R-trees creation time is measured when the new GCPS algorithm and the Quadratic algorithm are used. The real data sets are from TIGER/Line data distributed by United States Bureau of Census (TIGER/Line^{TM} Files, 2005). The first data set containing 53145 MBRs of Long Beach county roads (LB). The second data set containing 76999 MBRs hypsography data, Germany Hypsogr.

Table (2): Index creation time using real data sets.

	Quadratic		New GCPS		Improvement %
	time (seconds)	Number of splits	time (seconds)	Number of splits	
LB	3.718	1577	2.693	1565	28%
Hypsogr	6.021	2664	4.751	2658	21%

From Table 1 and Table 2, the new linear algorithm manages to reduce the R-tree creation time based on the improvement of splitting time. There are two important factors that affect this improvement in the R-tree creation time: the number of splits and the split cost. It is obvious that the splitting cost in the quadratic splitting algorithm $O(n^2)$ is higher than that of the new GCPS linear algorithm $O(n)$.

In Table 3, the average R-tree creation time and number of splits for both algorithms are measured using synthetic data file with n=100000 and different node capacities (Max fill values). The percentage of improvement in the R-tree creation time gained by GCPS is determined.

Table (3): Index creation time with different Max fill values.

Max fill	Quadratic (seconds)	Quadratic Number of splits	New GCPS (seconds)	New GCPS Number of splits	Improvement %
50	7.056	2939	5.711	2926	19%
100	10.897	1459	8.625	1452	21%
200	19.855	716	15.417	731	22%
300	28.637	484	21.182	468	26%
400	35.362	352	25.807	357	27%

From Table 3, as the (Max fill) increases, the number of splits will decrease and the split cost will increase. Consequently, the percentage of improvement in the R-tree creation time gained by the GCPS algorithm will increase. In low node capacity situations the number of splits will be high, while in high node capacity situations the split cost will be high with less number of splits. It is obvious that the bulk of performance improvement is more for high node capacity situations, in which quadratic splitting algorithm has the complexity of O(n^2), while the new GCPS algorithm has a linear complexity O(n) as it was shown in section2.

3.2 Index Query experiments

The difference between the new GCPS algorithm and the Quadratic algorithm is only in the splitting process of an over-flown node. Note that the node splitting process is called only when a node overflows. This will affect the final structure of the tree and the query performance of the index. The most commonly used query type in literature for testing node splitting algorithms is the intersection query type (Sleit, A. Al-Nsour, E. 2014; 222–236); (Theodoridis, Y. & Sellis, T. 1996; 161–171) ; (Ang, C. H. & Tan, T. C. 1997; 15–18). The performances of the two splitting algorithms are compared, and the query time is measured for various window query sizes. In Table 4, we test the following query sizes as a percentage of the test data space: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9. The average query time of 10 uniformly distributed intersection queries for each query size is measured. The indexes are built by the two splitting algorithms using the synthetic Uniform data files with n=100000.

Table (4): Query time (synthetic data set).

Test	Synthetic data file		
	Quadratic (milliseconds)	**New GCPS (milliseconds)**	**Improvement %**
1	5.863	5.718	2%
2	5.879	5.791	2%
3	5.805	5.968	-3%
4	5.856	5.698	3%
5	5.690	5.585	2%
6	5.756	5.560	3%
7	5.312	5.242	1%
8	5.094	5.084	0%
9	4.942	5.061	-2%

In Table 5 and Table 6, we performed the same window query tests for indexes built by the two splitting algorithms using the (LB) file and the (Hypsogr) file respectively. The average query time of 10 uniformly distributed intersection queries for each query size is measured.

Table (5): Query time (LB data set).

Test	LB		
	Quadratic (milliseconds)	**New GCPS (milliseconds)**	**Improvement %**
1	2.501	2.447	2%
2	2.547	2.566	-1%
3	2.537	2.521	1%
4	2.501	2.532	-1%
5	2.559	2.519	2%
6	2.709	2.629	3%
7	2.699	2.684	1%
8	2.709	2.747	-1%
9	2.676	2.664	0%

Table (6): Query time (Hypsogr data set).

Test	**Hypsogr**		
	Quadratic (milliseconds)	**New GCPS (milliseconds)**	**Improvement %**
1	3.568	3.571	0%
2	3.694	3.577	3%
3	3.687	3.672	0%
4	3.603	3.568	1%
5	3.660	3.624	1%
6	3.701	3.671	1%
7	3.635	3.651	0%
8	3.655	3.643	0%
9	3.696	3.725	-1%

The results in Table 4, Table 5 and Table 6 show that the query performance of the indexes built by the Quadratic algorithm and the new algorithm are very close.

4 Conclusions

In this paper, we present a simple efficient linear node splitting algorithm in an R-tree index. This algorithm is a linear enhancement of the Quadratic algorithm proposed in (Guttman A, 1984; 47–57). With the use of the new splitting algorithm in building the index, the index creation time will be improved without affecting the query performance of the index. We performed an experimental study using both real and synthetic data sets. For future work we think that this technique can be also applied to 3-D object problem spaces in a very similar approach. This technique could be very useful for modern databases with high frequencies that employ approximate query processing. A concurrent approach can be applied to this algorithm for example we can apply the concept of multithreading, while combining each of the two sub regions which has resulted from the original region.

References

- Guttman, A. (1984). *R-Trees, a dynamic index structure for spatial searching*. In: Proceedings of ACM SIGMOD conference, 47–57.
- Papadias, D. Sellis T. Theodoridis, Y. & Egenhofer, M. (1995). *Topological relations in the world of minimum bounding rectangles: A study with R-trees*. In: ACM SIGMOD.
- Fu, Y. Teng, J. & Subramanya, S. (2002). *Node splitting algorithms in tree-structured high-dimensional indexes for similarity search. In: Proceedings of the 2002 ACM symposium on applied computing (SAC '02).* New York: ACM, 766–770.
- Sleit, A. (2008). *On using B+ tree for efficient processing for the boundary neighborhood problem*. WSEAS Transactions on Systems; 11(11): 711–720.
- Azzam, Sleit. Esam, Al-Nsour. (2014). *Corner-based splitting: An improved node splitting algorithm for R-tree*. Journal of Information Science, Vol. 40(2), 222–236.
- Sellis, T. Roussopoulos, N. & Faloutsos, C. (1987). *The R+-tree: A dynamic index for multidimensional objects*. In: Proceedings of the VLDB conference, 507–518.
- Beckmann, N. Kriegel, H. Schneider, R. & Seeger, B. (1990). *The R*-tree: An efficient and robust method for points and rectangles*. In:Proceedings of the ACM SIGMOD conference, 322–331.
- Theodoridis, Y. & Sellis, T. (1996). *A model for the prediction of R-tree performance*. Proceedings of the fifteenth ACM SIGACTSIGMOD- SIGART symposium on principles of database systems (PODS '96). New York: ACM, 161–171.
- Ang, C. H. & Tan, T. C. (July 15–18, 1997). *New Linear Node Splitting Algorithm for R-trees*. Proceedings of the 5th International Symposium on Advances in Spatial Databases (SSD '97), Berlin, Germany.

- Al-Badarneh, A. Yaseen, Q. & Hmeidi, I. (2010). *A new enhancement to the R-tree node splitting*. Journal of Information Science; 36(1): 3–18.
- Al-Badarneh, A. & Tawil, M. (2009). *LINEAR R-TREE REVISITED*. International Journal of Computers and Applications, Vol. 31, No. 2.
- TIGER/Line^{TM} Files. (2005). *Technical Documentation*, US Bureau of Census, Washington, DC, accessible via URL: http://www.census.gov/geo/www/tiger/