International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:5, No:4, 2011

Balancing of Quad Tree using Point Pattern Analysis

Amitava Chakraborty, Sudip Kumar De, and Ranjan Dasgupta

***Abstract*—**Point quad tree is considered as one of the most common data organizations to deal with spatial data & can be used to increase the efficiency for searching the point features. As the efficiency of the searching technique depends on the height of the tree, arbitrary insertion of the point features may make the tree unbalanced and lead to higher time of searching. This paper attempts to design an algorithm to make a nearly balanced quad tree. Point pattern analysis technique has been applied for this purpose which shows a significant enhancement of the performance and the results are also included in the paper for the sake of completeness.

***Keywords*—**Algorithm, Height balanced tree, Point pattern analysis, Point quad tree.

I. Introduction

QUAD tree and its various derivatives are being considered as the backbone for the storage, retrieval and analysis of spatial data. Four children of each node of the quadtree represent the four quadrants of the two dimensional space under it and the X, Y coordinates of point (also area) features are stored and thus the quadtree is formed. This way of storing the point features in a quadtree according to their spatial distribution helps in searching the tree in a depth-first search manner. Searching on quadtree is a frequent operation for routine GIS [12]queries and it becomes a bottleneck when the quadtree is not height balanced and hence the need for height balanced quadtree is felt. In this paper we try to apply point pattern analysis techniques to store point quadtree in nearly height balanced order and have shown the results for searching thereof.

A. *Quad Tree and Its Derivatives:* The quadtree [1] is a hierarchical, variable resolution data structure based on the recursive partitioning of a plane into four quadrants. This data structure is widely used for representing collection of points. In 1974 Finkel & Bentley proposed point quadtree [2] to store points in a multidimensional space. Each node of the point quadtree has four children, each representing a quadrant of four directions, namely, NE, NW, SW, and SE. The first point that is inserted serves as the root node, while the second point is inserted into the relevant quadrant of the tree rooted at the first point and so on. Point quadtree is well suited for searching but it creates significant search overhead when points are inserted into the tree in an arbitrary fashion resulting a highly unbalanced point quadtree. In 1982, Keden proposed bisector list quadtree (BLQT) [3] as a modification to store extended objects. In BLQT extended objects intersecting more than one quadrant are stored in x & y bisector line lists associated with the parent quadtree. But the x & y bisector line list can grow long & then severely can affect the search time. To avoid this concept, Brown proposed Multiple Storage Quad Tree (MSQT) [4], which stores the interesting objects in all of the intersected quadrants. Although it removes the use of the bisector list but also wastes a lot of space by storing objects more than once. In order to avoid this problem objects are marked the first time they are reported, and once marked, are not reported again. Two layer quadtree [5] is another data structure to resolve bisector list problem, which sorts and stores objects in the layer based on their left corner locations & the size attributes of the objects are also stored as additional information in the directory layer. It is useful for both region query and size query. Quad List Quad Tree (QLQT) [6] is a modification of MSQT with four lists in each quadrant. If any object intersects the leaf quadrant, a reference to this object will be included in one of the four lists according to the relative position of the object w.r.t. the leaf quadrant it intersects. QLQT is efficient for large window queries but sometimes it becomes heavily skewed due to the lists in each quadrant. The YAQT [7] is another modified form of MSQT with no list required for storing crossing objects. It improves the region query speed with the cost of increasing memory requirement. The Multicell Quad Tree [8] is a two level tree structure. At the upper level it is a MSQT and at the lower level each leaf quad of the MSQT is further subdivided into equal sized cells. It is useful for large window query and it requires less memory space than MSQT. PR quad tree [9] is another variant of quad tree to store points. It is based on the recursive decomposition of the underlying plane into four similar quadrants until each quadrant contain no more than one point. Although point's insertion and deletion are quite simple with this data structure, the trees may contain arbitrary depth, independent even on the number of input points. Besides points, quad tree is a well accepted data structure for representing regions, curves, surfaces, volumes etc. For more discussions on relevant topics see [10], [11], [13], [14], [15]. It has been observed that most of the research work on quad tree & its derivatives had focused on the storage & retrieval of various geographical features & limitation of one such structure had been taken care of in some other modified version. Whatever are the add-on modifications, the inherent quad tree structure suffers from the height balance issue while a huge number of features are stored in the quad tree in an arbitrary fashion. No significant effort was observed to overcome this height balance issue and to make the searching operation on quad tree more efficient.

1 is with the Asansol Engineering College, Asansol, West Bengal University of Technology, PIN CODE 713305, INDIA(phone: +919474316464; fax:+913323373959 ; e-mail: amitava.c@rediffmail.com).

2 is with the Asansol Engineering College, Asansol, West Bengal University of Technology, PIN CODE 713305, INDIA(email: sudipkumarde@gmail.com).

3 is with the National Institute of Technical Teachers' Training & Research Institute, Block-FC, Sector-III, Salt Lake City, Kolkata, PIN CODE 700106, INDIA(e-mail:ranjandasgupta@ieee.org).

II. Populating Quad Tree With Points In Arbitrary Fashion

The quad-tree node and the four quadrants have been shown in Fig. 1.

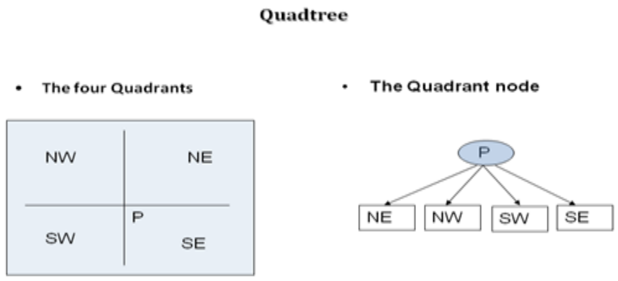

Fig. 1 Quadtree

The point quad-tree is constructed consecutively by inserting the data points one by one. To insert a point, firstly a point search is performed. If no point corresponding to target point (the point which has to be inserted) is found in the tree, then the target point is inserted into the leaf node where the search has terminated. The planar representation of a point quadtree is shown in Figure 2(a).

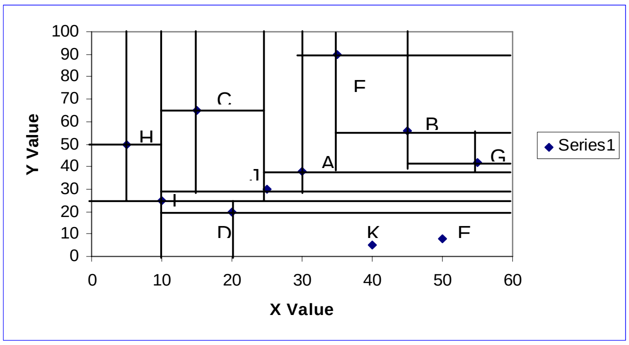

Fig. 2 (a) Planar Representation of Point Quad tree

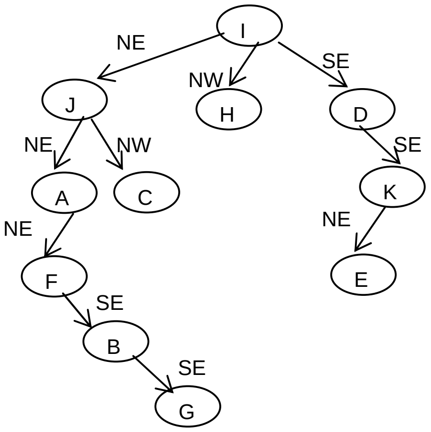

Fig. 2 (b) Point Quadtree (Arbitrary Fashion)

Searching in a quad-tree is similar to searching in an ordinary binary search tree. At each level, one has to decide which of the four sub trees need to be included in the future search. This process is repeated recursively up to the depth of the tree. In case of point quad-tree at each level, three sub trees are rejected and only one sub tree is followed for future search. The point Quad-trees are especially attractive in applications that involve search [1]. The height as well as the shape of the point quadtree highly depends on the insertion sequence. When point quadtree is populated in arbitrary fashion then the height balanced quadtree might not be achieved. As a result the average searching time increases and the advantage of using point quad-tree is reduced.

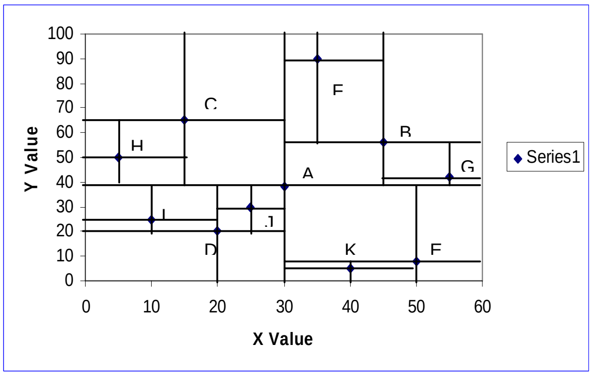

Fig. 2 (c) Planar Representation of Point Quadtree

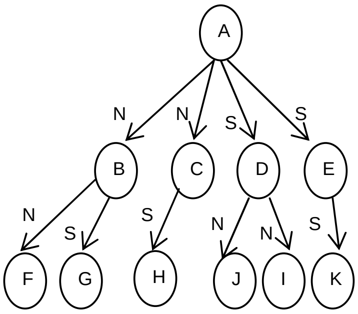

Fig. 2 (d) Point Quadtree

For example, the planar representation of points and its corresponding point quad-tree has been shown in Figure 2(b) when point insertion is in the sequence e.g. I, J, A, F, B, G, C, H, D, K, and E. The depth of the tree becomes 6 and the first leaf is found at level 1.So, it becomes unbalanced. But, if the insertion sequence is- A, B, F, G, C, H, D, J, I, E, K we find a nearly balanced tree. (See figure 2.(c) and 2.(d))

III. Point Pattern Analysis

The Point Pattern Analysis is a technique that is used to identify patterns in spatial data. There are several methods and algorithms that endeavor to describe pattern for a collection of points. One of the common methods for spatial pattern analysis is Quadrant Count Method and a brief review of the method is presented in Sec 4.1 and its use for balancing quadtree is described in Sec 5.0.

A. Quadrant Count Method:

According to this method total data set or total region is partitioned into n equal sized sub regions. These sub regions are also known as quadrants. The size of the quadrant takes an important role to determine the point pattern. If quadrant size is too large then patterns within that large quadrant may be missed. Again if quadrant size is too small and if there exist clustering then due to small scales it (clustering pattern) may be missed. If there is a large amount of variability in the number of points from quadrant to quadrant then this implies a tendency towards clustering. It happens when some quadrants have many points and some quadrants have none. If there is a little amount of variability in the number of points from quadrant to quadrant then this implies a tendency towards a pattern that is termed regular, uniform, or dispersed. It happens when the number of points per quadrant is about same in all quadrants.

Let us partition the total region into n sub regions and:-

a) the total number of points in each quadrant be *Xi*
b) the mean number points per quadrant be M= (Total number of points) / n

Then variance of the number of points per quadrant (V) is

$$V = \frac{\sum_{i=1}^{n} X_i^2 - (\sum_{i=1}^{n} X_i)^2 / n}{n-1}$$

And the Variance to Mean Ratio is defined as

VTMR =V/M

If VTMR >1, it indicates a tendency toward clustering in the pattern.

If VTMR < 1, then it indicates an evenly spaced arrangement of points [12].

If VTMR=1, then the pattern is random.

IV. Use Of Quadrant Count Method In Quad Tree Height Balancing

The main logic behind the balancing mechanism is to choose a point as root such that each quadrant has more or less same number of points. As the quadrant count method helps us to find out the point pattern of the distributed points we apply this point pattern analysis technique in our algorithm with necessary modification.

A. Proposed Algorithm

The algorithm Balance_Maker( ) deals with two functions. Firstly the function recursively calls two sub functions, first one of which selects the seed points & the second one employs the actual physical points based on that seed points. Ultimately the function provides the nearly balanced quadtree as output. In this algorithm the physical points are stored in both X-value & Y-value wise and are stored separately. Suppose the set of physical points is P=[(x3, y1), (x5,y3), (x2,y4), (x1,y5), (x4,y2)] then the set of X-value-wise-sorted-points would be PX=[(x1,y5),(x2,y4), (x3,y1), (x4,y2), (x5,y3)] and the set of Y-value-wise-sorted-points would be PY = [(y1,x3),(y2,x4),(y3,x5),(y4,x2),(y5,x1)].The main logic of this step is to find out the position where the point distribution in each quadrant is more or less same. To find that position some additional points are created whose x values are taken from PX set and y values are taken from PY set. Then the set of newly created points would be V= [(x1, y1), (x2, y2), (x3, y3), (x4, y4), (x5, y5)]. We call these points as Virtual points because these are not the actual physical points. The x value of virtual point is taken from X-value-wise-sorted-array and the y value is taken from Y-value-wise-sorted-array. At each virtual point a partition (we termed it as virtual partition) is made and total number of physical points in each quadrant is counted and then the Quadrant Count Method is applied with an exception. The exception is that the total area is not divided into four equal sized quadrants. According to this method Variance to Mean Ratio (VTMR) is calculated for each virtual point. The virtual point with minimum VTMR is selected. We call this point as 'Seed point'. The Seed point is a virtual point and at each seed point virtual partition has been made. But we have to consider the actual physical partition rather than virtual partition. For this reason the physical point near to the seed point is searched. To find the nearest physical point of the seed point, Pythagorean distance measurement formula is used. The distance between each physical point, within the range, and the seed point is calculated. The physical point for which minimum distance is achieved is selected. We denote that physical point as Candidate Balanced Point. This process executes recursively until all the physical points are treated as candidate balanced point.

A. Algorithms:

i) *Algorithm: Balance_Maker* ()

Input: The range of x & y co-ordinate, total no of physical points, physical points with x, y coordinate

Output: A Balanced Quadtree

Procedure:

Step 1: If the total no. of points within the range is 0, then return
Step 2: Call Virtual_Point_Finder (range) // to find out seed points
Step 3: Call Physical_Point_Finder (range) // to find out physical points
Step 4: Find the new ranges for all four quadrants.
Step 5: Store the physical point, supplied by Physical_Point_Finder (range)
Step 6: Call Balance_Maker () for each quadrant, with specific range and total no. of points within that range.
Step 7: Store all the candidate balanced points using the sortXvalue [] & sortYvalue [] arrays.

ii) *Algorithm: Virtual_Point_Finder* (range)

Input: The sortXvalue [], sortYvalue [] and the range

Output: A list of seed points

Procedure:

Step 1: Extracts those points which are within the range and store those point's reference into Xwise_index[], in increasing order of X value
Step 2. Extracts those points which are within the range and store those point's reference into Ywise_index[], in increasing order of Y value
Step 3: Store the total no. of points within the given range
Step 4. Store the index of the point within the range. // Virtual point's X value & Y value are taken from the X value of sortXvalue & Y value of sortYvalue []
Step 5: Find that virtual point for which point distribution is better than other virtual points. Count the no. of physical points for all four quadrants.
Step 6.1: Calculate the Variance & Mean as accordance.
Step 6.2: Calculate VTMR = Variance/Mean
Step 6.3: Find the Virtual Point for which minimum VTMR is achieved // store the points as seed points.

iii) Algo: Physical_Point_Finder (range)

Input: sortXvalue [], sortYvalue [] and seed points

Output: A set of candidate balanced points

Procedure:

```
    Step 1: Starts with total no. of points within the range
    Step 2: Store virtual point’s X value(vpx) & virtual point’s Y value(vpy)
    Step 3: Find physical point’s x value($x_1$) & physical point’s y value($y_1$)
    Step 4: Calculate the distance,
        $d = \sqrt{}((vpx-x_1)^2+(vpx-y_1)^2)$
    Step 5: Store selected physical point’s coordinate ppx & ppy
    Step 6: Make partition at (ppx,ppy) and count down the number of points for all four quadrants.
    Step 7: Store the index number of selected physical point & also store the total no. of points for all four quadrants.
```

V. Results

In the real world objects within a particular area or zone are not distributed evenly. Rather it is observed that within an area some places are highly populated and some places are lowly populated. To take such effect of the real world situation our input data points are also not distributed within the total area or total zone. The total zone or simply zone is subdivided into sub zones and each sub zone is arbitrarily populated with different percentage (0 to 100%) of data points. We have done our experiment with 4000 data points. We assumed that the level of the root node is 1.

A. Comparison of Results

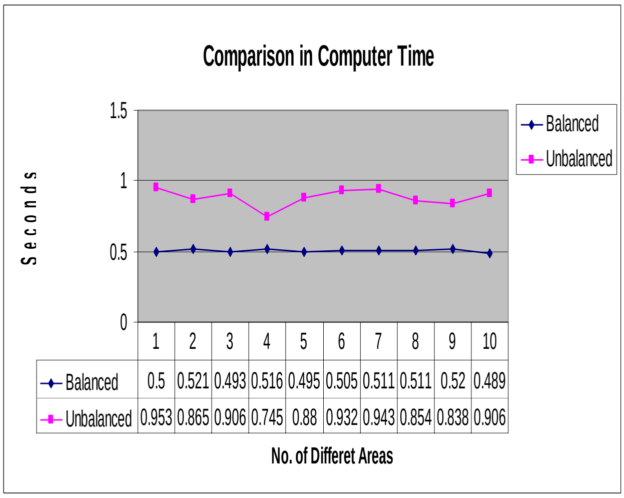

Fig. 3 Average Search Time-wise Comparison

Each row in the Table 1 corresponds to a unique zone, average search time with respect to arbitrary fashion insertion and nearly balanced quad tree according to our algorithm.The average search time for the quadtree created in arbitrary fashion and nearly balanced quad tree are 0.8822667 & 0.5061 respectively. From the result it is clear that if point quadtree is populated in arbitrary fashion then the quad tree may not be a height balanced one. As a result average search time increases. (See Table I) The result shows that the tree becomes nearly height balanced by using the proposed algorithm based on point pattern algorithm. The comparison of the result is also shown in the figure 3.

TABLE I
COMPARISON OF THE ARBITRARY INSERTION & NEARLY BALANCED QUAD TREE

Zones	**Avg. Search Time (for Arbitrary Fashion)**	**Avg. Search Time (Nearly Balanced)**
1	0.953333	0.5
2	0.864667	0.521
3	0.906333	0.493333
4	0.744667	0.515666
5	0.880333	0.495
6	0.932333	0.505
7	0.942667	0.510666
8	0.854	0.510666
9	0.838333	0.520333
10	0.906	0.489333
Avg.	0.8822667	0.5061

VI. Conclusion

Balancing of tree is an age-old problem and several attempts had been made to balance a quad tree to increase the efficiency of overall performance. Here we use point pattern analysis technique on quad tree with due modifications. After implementation of our algorithm, we notice that performance of it has improved.

References

[1] H. Samet, “The quadtree and related hierarchical data structures”, ACM Computing Surveys 16, 2(June 1984), pp. 187-260.
[2] R. A. Finkel & J. L. Bentley, “Quad trees – A data structure for retrieval on composite keys”, Acta Inform., vol. 4, no. 1-9, 1974.
[3] G. Keden, “The quad-CIF Tree: A data structure for hierarchical on lone algorithms”, in Proc. 19th Design Automation Conf. , pp. 352-357, June 1982.
[4] Brown R. L. “Multiple storage quad tree : a simpler faster alternative to bisector list quad trees”, “ IEEE Trans. Computer Aided Design Vol CAD-5 pp 413-419, July 1986.
[5] W. Li, S. Legendre & K. Gardiner, “ Two-layer quadtree: a data structure for high-speed interactive layout tools”, International Conference Computer-Aided-Design, pp. 530-533, 1988.
[6] W. Ludo & D. Wim, “ Quad List Quad Tree: A geometrical structure with improved performance for large region queries” Computer –Aided Design, Vol-8 March 1989.
[7] P. V. Srinivas & V.K. Dwivedi, “ YAQT: Yet Another Quad Tree”, IEEE Trans., pp. 302-309, 1991.
[8] S. J. Lu & Y. S. Kuo, “ Multicell Quad Trees”, IEEE Trans., pp. 147-151, 1992.
[9] J. A. Orenstein, “Multidimensional tries used for associative searching”,
[10] Information Processing Letters 14, 4(June 1982), 150-157.
[11] Pei-Yung Hsiao, “Nearly Balanced Quad List Quad Tree- A Data Structure for VLSI Lay out Systems”, 1996 OPA(Overseas Publishers Association) Amsterdam B. V.
[12] Pei-Yung Hsiao & Lih-Der Jang, “Using a Balanced Quad List Quad Tree to speed Up a Hierarchical VLSI Compaction Scheme”, IEEE, 1991.
[13] David O’Sullivan & David J. Unwin, “ Geographic Information Analysis”, John Wiley and Sons, 2002.
[14] Ralf Hartmut Güting, “An Introduction to Spatial Database Systems”, Special Issue on Spatial Database
[15] A. Klinger, Patterns and Search Statistics , in Optimizing Method in Statistics, J. S. Rustagi, Ed., Academic Press, New York, 1971, pp. 303-337.
[16] H. Samet & R.E. Webber, “ Storing a collection of polygons using quadtrees,” ACM Transactions on Graphics 4, 3(July 1985), pp. 182-222(also Proceedings of Computer Vision and Pattern Recognition 88, Washington, DC, June 1983,pp. 127-132).