GridTree and its application in kNN and CRQ algorithm

Yulong Qiu, Yun Lu
HPDRC Lab, ECS, Florida International University, Miami, FL, 33199, USA
yuqiu@cs.fiu.edu

Abstract

In the field of spatial index, R-tree \cite{1} is widely used in spatial database for its intuitive, simple, efficient, high space utilization rate features, and has been the most popular algorithm used for spatial data storing and management, and the most developed one among lots of spatial data index algorithms. Plenty of algorithms using spatial index are built based on R-tree, which has made it a standard of reference when evaluating spatial index structures and algorithms.

In recent years, with the rapid development of mobile internet, the storage and management of massive mobile data from an increasing number of mobile devices has brought great challenges to the traditional spatial databases. The characteristics of R-tree spatial data index are slow updating, fast querying, and small storage requirement, can adapt well to the traditional spatial databases which have the demands of little updating, more querying, and high cost storage. But mobile application data requires continuously updating, continuously querying, and rapidly response, and the traditional R-tree index structure cannot meet the demands. More and more research work begin to shift to those spatial data indexes which can update quickly, query continuously, and response rapidly.

The GridTree researched in this paper is such a kind of spatial data index structure which has the natural features of fast updating, and suiting for continuous query which inherit from Quadtree \cite{2} while its tree height is less than that of Quadtree. Although the tree height of GridTree may be larger than that of R-tree, which will normally lead to more IO seek operation, in actual application, the data space is always limited by resolution which can prevent the GridTree height to reach very high (in most cases, five level is enough when using 16*16 GridTree) and with the trend of more and more Solid State Disk being used, the seek operation cost will no longer a critical problems and factors when evaluating an spatial index structure. Therefore, GridTree has a large research value in the future.

This paper focuses on the k Nearest Neighbors \cite{3} (kNN) and Continuous Range Query \cite{4,9} (CRQ) algorithm which based on the GridTree spatial continuous search feature to assess the performance of spatial continuous searching and the ability of fast response. Our evaluation showed that the new GridTree structure can offer faster and better index creation, and at same time, good update efficiency, especially much better parallel operation then most of popular spatial data index structures.
1. Introduction

Spatial searching is a fundamental primitive in non-traditional databases such as GIS, CAD and multi-media applications. With the rapid development of these databases in the past years, extensive research has been conducted on the design of efficient data structures to enable fast spatial data retrieving. Several data structures have been invented in this circumstance. These include Quadtrees, R-trees, hB-trees, SS-trees and SR-trees. Our research has improved these basic structures further by proposing the new Girdtree structure and accompanied query techniques, faster and better index creation and better update efficiency, especially quicker parallel operation for abundant concurrence. In this article, we will discuss the Girdtree structure and the kNN and CRQ algorithms with it. The reason why we choose kNN algorithm is that it is a very good algorithm for evaluating continuous searching performance, and CRQ algorithm is for progressive searching evaluation.

2. GridTree

In this part, we will introduce the GridTree and its operations.

2.1 GridTree structure

In this section, we will introduce the structure of GridTree. The outline of GridTree is using the M*M Grid to decompose the space into multiple level hierarchy structure. Take the example of 4*4 Grid, the space decomposing process may be like as figure 1[5].

![Figure 1 example of 4*4 GridTree space decomposing](image)

Every GridTree node is divided into M*M cells. Cell is used for storing spatial data and has a storage capacity. When the amount of spatial data in one cell outnumbers the storage capacity (overflow), one new GridTree node will be created as the sub-node or child node of the cell, and then all the data in the cell need to be moved to the cells of sub-node. The cell contains data is called leaf-cell, else called non-leaf cell.
The GridTree is built based on this logic, and the follow sections will introduce the UPDATE and RANGE QUERY operations on it.

For simple, we only discuss point data set, because the multiple-dimensional data set use the similar way.

2.2 Update

Update operation includes 3 kinds: insert, delete and modify point data from GridTree.

1) Insert operation

Insert operation is for inserting one point data into GridTree. The steps of this process is as flow:

i. Firstly, find the leaf-cell which overlaps the data point.
ii. Insert the data into the leaf-cell.
iii. If the leaf-cell is overflow, create one new sub-node for this cell, and then move all its data into sub-node cells, at the same time, set the leaf-cell as non-leaf cell.
iv. If the leaf-cell is not overflow, exit.

2) Delete operation

Delete operation is for deleting one point data from GridTree. The steps of this process is as flow:

i. Firstly, find the leaf-cell that contains the data point.
ii. Delete the data from the leaf-cell.
iii. If the GridTree node containing the leaf-cell is underflow, move all the cells’ data into the parent cell. And then delete the GridTree node.
iv. If the GridTree node containing the leaf-cell is not underflow, exit.

3) Modify operation

Modify operation is for modify the data value from the GridTree. The steps of this process is as flow:

i. Firstly, find the data from the GridTree.
ii. Modify the value of the data.
iii. If the data is out of the spatial area of the leaf-cell which contains the data, use the Delete and Insert operation to reinsert it into GridTree.
iv. If the data is still in the spatial area of the leaf-cell which contains the data, exit.

2.3 Range query

The Range query is for searching GridTree using a query window to find all data located in the window. An example is shown as figure 2.
The process of Range query is very simple, with only two steps:

i. Get all leaf-cell that intersect with the query window.
ii. Get all data from the leaf-cells that located in the query window, and exit.

2.4 Neighbor query

The Neighbor query is for searching the neighbor cells of one specified cell. For the inner cell of GridTree node, we can get its neighbor cells easily by increase or decrease its cell index. For the cell at GridTree node boundary, we need its parent cell to help to jump to its neighbor cell. The example is shown as figure 3. The right neighbor of cell A[2,3] is A[3,3], and the lower-right neighbor of A[3,3] if B[0,0].
The process of this Neighbor query has 2 steps:

iii. Get the index of neighbor cell, if the index is within the range of GridTree node index, return the neighbor cell directly.

iv. If the index is out of the range of GridTree node index, use the parent cell to jump to its neighbor cell.

3. kNN based on GridTree

In this part, we will introduce the kNN algorithm based on GridTree.

K Nearest Neighbors (KNN) algorithm is for searching the nearest k points of the querying point. It concept is show as figure 4.

Figure 4 k Nearest Neighbors

The main idea of kNN algorithm based on GridTree is that get the center leaf-cell that overlap the query point, and then expand the searching circle neighbor by neighbor. All the cells in searching circle is put in one priority queue along the distance between the cell and the query point with the ascending order. When expand the circle, we just expand the cell with the minimum distance. The expand operation will remove the cell from the priority queue and push its unvisited neighbors in eight directions(south, north, west, east, southwest, southeast, northwest, northeast) into priority queue. Because the visited neighbors will not be added to the queue, so the new circle will be a circle expanded from the old circle, only the outer cell added. The process is shown as figure 5 and figure 6.
The steps of the process are shown below:

i. Get the leaf-cell that overlap the query point, and put it into the priority queue.

ii. Visit the first cell in the priority queue. If the distance between the cell and the query point is less than the maximum distance of the result set, go step iii; else return result set and exit.

iii. Remove the first cell in the priority queue, and add its unvisited neighbors in eight directions into priority queue. Back to step ii.

4. CRQ based on GridTree

In this part, we will introduce the CRQ algorithm based on GridTree.
The Continuous Range Query (CRQ) is for continuous searching the objects within range \( r \) when one point moving along a path. The figure 7 shows how CRQ works.

![Figure 7 continuous range query](image)

When the point moving along the path, its searching circle sweeps an area which we called as query region with the shape of one rectangle and two semi-circle, which shown as figure 8, the region with the edge of dash line.

![Figure 8 continuous searching area](image)

Before introduce the algorithm of CRQ, we need to introduce two relative definitions used for CRQ, the Domain region and progressive searching.

4.1 Domain region and progressive searching

The domain region of the GridTree node cell is a region that has a boundary which get from expand the boundary of cell by range \( r \). the diagram is shown as figure 9.

![Figure 9 domain region](image)
The physical meaning of the domain region is that when the moving point is out of the domain region, its searching circle does not intersect with the inner cell, so there is no need to search the cell’s data, and only when the moving point goes into the domain region, we need to search the cell’s data to find those data located into searching circle. This is used to speed up the searching process.

For the sake of simplicity, we do not use the accurate domain region for computing, but use an approximate domain region shown as figure 10, which is got by expand the cell’s rectangle by range \( r \). The location where moving point enter the domain region is call Domain Entry Point (DEP), shown as point \( s \) in figure 10.

Figure 10 approximate domain region

The progressive searching is one searching method that searching cells that along the moving path one by one, instead of getting all cells at one time. Figure 11 and figure 12 shows the logic process of progressive searching. In figure 11, cells that is visiting is cell A0 and cell B0. For the reason that DEP of A0 is in leaf of DEP of B0, so we choose A0 as the next visiting cell, instead of B0 (this method is the same as that used in kNN algorithm, actually they can both use priority queue to implement). After we search cell A0, we expand the search circle by replace A0 with its right neighbor A1, as shown in figure 12.
In this section, we will introduce the details of CRQ on GridTree.

The algorithm of CRQ on GridTree is based on the domain region and progressive searching referred in 3.1 section. Its main idea is very simple, use the domain region to decide whether to visit one cell or not and use the progressive method to visit cell one by one, instead of getting all cells that intersect with the query region. When we visit one cell, we need to calculate the split point for every data in cell. The split point is the point on the moving path that has a distance of $r$ between itself and the data point, as shown in figure 8, the point $s$ and $t$ is two split point of data point $p$.

The output result of CRQ algorithm is one two-dimensional array. The first element of the row is the path segment that is split by two adjacent split point. For example, the figure 13 is an example of CRQ split point. Point $\{p_0, p_1, p_2, p_3\}$ is the result data point located in the query region, and point $\{p_0e, p_{1s}, p_{1e}, p_{2s}, p_{2e}, p_{3s}, p_{3e}\}$ is the split point of result data point. Point $s$ is the start point of moving path. Then the output result is shown as sheet 14.
### Sheet 14 output result of CRQ algorithm

<table>
<thead>
<tr>
<th>Split segment</th>
<th>Point set</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-p1s</td>
<td>p0</td>
</tr>
<tr>
<td>p1s-p0e</td>
<td>p0,p1</td>
</tr>
<tr>
<td>p0e-p1e</td>
<td>p1</td>
</tr>
<tr>
<td>p1e-p2s</td>
<td>NULL</td>
</tr>
<tr>
<td>p2s-p3s</td>
<td>p2</td>
</tr>
<tr>
<td>p3s-p2e</td>
<td>p2,p3</td>
</tr>
<tr>
<td>p2e-p3e</td>
<td>p3</td>
</tr>
</tbody>
</table>

So the detail steps of CRQ on GridTree is as below:

i. Draw one circle based on the start point of the moving path, with the radius r.

ii. Get all the cells that intersect with the circle, and push them to the cell priority queue.

iii. If the cell priority queue is NULL, exit.

iv. Else, visit the cells which has the earliest DEP in the cell priority queue. Calculate all data’s split points and add them to the split point priority queue.

v. Output the two-dimensional result array whose split segment located before the earliest DEP of the cell priority queue.

vi. Remove the cells visited and get their neighbor along the moving path using the progressive method, and push these neighbor cells into the cell priority queue. Back to step iii.

### 5. Related work

Currently, in spatial index research area, R-tree\cite{1,10,11} is the most popular structure used for spatial index algorithms. However, the R-tree failed in parallel insertion. There is no way to let R-tree insert two individual points at the same time, even the two points are far away or have no relationship which could be easily succeed by GridTree structure. Also Quadtree\cite{2,11} is widely used in GIS\cite{11} and image processing domains. But limited by the native of the Quadtree itself, the depth of the tree would be very deep and out of control, which lead it low efficiency. None of the current popular index structures can achieve both simple and contracted index and efficient concurrent update.

### 6. Conclusion

In this report, we introduce the algorithms of kNN and CRQ on GridTree. This indicates one common way for how to use GridTree to solve the traditional problems based on continuous spatial search. The future research will be using experiments to verify the performance of algorithms on GridTree and using this theory to solve more traditional problems.

### Acknowledgement
This material is based in part upon work supported by the National Science Foundation under Grant Nos. I/UCRC IIP-1338922, AIR IIP-1237818, SBIR IIP-1330943, III-Large IIS-1213026, MRI CNS-0821345, MRI CNS-1126619, CREST HRD-0833093, I/UCRC IIP-0829576, MRI CNS-0959985, FRP IIP-1230661, SBIR IIP-1058428, SBIR IIP-1026265, SBIR IIP-1058606, SBIR IIP-1127251, SBIR IIP-1127412, SBIR IIP-1118610, SBIR IIP-1230265, SBIR IIP-1256641. Includes material licensed by TerraFly (http://terrafly.com) and the NSF CAKE Center (http://cake.fiu.edu).

Reference


