2-Level R-tree Index Based on Spatial Grids and Hilbert R-tree

GUO Jing LIU Guangjun DONG Xurong GUO Lei

ABSTRACT Multi-level spatial index techniques are always used in large spatial databases. After a general survey of R-tree relevant techniques, this paper presents a novel 2-level index structure, which is based on the schemas of spatial grids, Hilbert R-tree and common R-tree. This structure is named H2R-tree, and it is specifically suitable for the indexing highly skewed, distributed, and large spatial database. Algorithms and a sample are given subsequently.

KEYWORDS spatial index; GIS; R-tree; H2R-tree

CLC NUMBER TP311;P208

Introduction

Spatial index is the data structure between spatial operation algorithms and spatial data objects, and it is usually used to improve the efficiency of spatial data operations. Spatial index is one of the fundament techniques to manage the spatial data of very large scale. In present, R-tree\(^1\) and its variants Ez-4\(^2\) have been the major methods in research.

R-tree is a dynamic tree. The classical R-tree can be loaded by OBO (one by one) method. Usually, it causes heavy overlaps among the tree nodes, and the utilization of R-tree is no more than 70%. Many practical spatial data applications belong to static applications, in which the data are either rarely updated or the period between two updates is very long. So the static data can be loaded only once, and the concrete way of these methods are well known in the literature as "bulk loading" or "packing". Roussopoulos proposed the first bulk-loading algorithm\(^3\), which groups data according to the x coordinates of the MBRs (minimum bounding rectangle), and then loads the grouped data into each layer's nodes from bottom to top. His method became the basic idea for later developed bulk-loading methods, such as the algorithm based on Hilbert ordering\(^4\) and STR\(^5\) (which groups the data firstly according to x and then y coordinates). More later, Kamel presented Hilbert R-tree\(^6\) by changing the structure of tree nodes. By use of Hilbert R-tree, the newcome data can also be inserted dynamically into the Hilbert order, to guarantee the R-tree's efficiency. A new research topic recently put forward is bulk-inserting, which is based on bulk-loading, focusing on how to insert a set of newcome data into already existed R-trees efficiently. Data updating may cause the database to be locked, so the insertion speed must be considered in prior, and try to avoid damaging the tree's quality during insertion. The representative methods of bulk-inserting are STLT and GBI\(^8\) which insert small tree into big tree, and buffer-tree which makes full use of the main memory.

Recently, the technique of combining grid spatial index and R-tree is drawing more attention. For example, Guo Jing proposed QR-tree\(^9\) which can mix the data stores of quad-tree and R-tree. But QR-tree has some shortages. First-
ly, it is not easy to determine how many levels should the quad-tree be set. Secondly, R-tree made by many fragmentary data may increase the store spending. More importantly, its example did not show definitely that in which mode to store the tree, either the mode of disk-file or that of main memory. Maybe QR-tree could only be realized in main store and couldn’t satisfy the common demand of creating disk-file index.

To manage mass spatial data, it is often necessary to create 2-level index. One typical case in the distributed data service environments is that local R-trees has been created and running, and the upper-level application needs to manage the data with larger bound, which includes the local bounds. As shown in Fig. 1, where the whole mapped region contains local regions of A, B, C and D, and A B C are stored in local data servers. In this case, it is necessary to build an efficient 2-level R-tree index structure which not only conform to the management traditions with spatial grids, but also can manage local R-trees efficiently with avoiding redundancy of store and not sacrifice searching efficiency. What is of more importance is that large-scale database is usually in multi-users state, so data updates should be restricted to local areas and avoid great influences on the index structure of main server, and this updates should be done as soon as possible.

Fig. 1  Distributed spatial data service

To meet the requirements for the management of distributed mass spatial data given in Fig. 1, we propose a novel 2-level index structure which is named H2R-tree. This paper is organized as follows; Section 1 describes the principle of H2R-tree and the algorithms in detail; Section 2 discusses the advantages of H2R-tree and gives an example; Section 3 gives some research directions in the future.

1  H2R-tree

1.1 Motivation

Above all, let’s present a general discussion on the 2-level spatial index based on grids and R-tree. So we nominate the index created directly by the spatial data as lst-level index, and the one created based on the lst-level indices as 2nd-level index.

The 2nd-level index usually adopts grid index, namely spatial grid-file or quad-tree. QR-tree mentioned above is a 2-level index tree combining quad-tree and R-tree. Its nodes, including the inner nodes, in each level are related to a specific R-tree. However, because the R-tree bounds of higher-levels are wider than those of lower-levels, it is difficult for QR-tree to satisfy the requirements for distributed data management in Fig. 1. A more general application is using the spatial data of each leaf node create an R-tree, i.e., each spatial grid is related to a R-tree, so that create a 2-level index. However the rectangles of the lst-level tree are usually not as regular rectangles as the grids of the 2nd-level tree, so it is unavoidable storing repeatedly. There are two cases for such redundancy.

1) If a spatial data MBR crosses different grids and it exclusively belongs to some lst-level R-tree, there must exists information redundancy about this lst-level R-tree in the 2nd-level R-tree. In order to guarantee no missing results, copies of the pointer to this R-tree has to be stored in all grids intersecting this R-tree’s MBR. What is more, if the updates of the lst-level R-tree change the tree’s rectangle, all grids that store this R-tree’s pointer will have to get updated, and so the operation is complex and inefficient.

2) If a lst-level R-tree uniquely belongs to a 2nd-level grid and some spatial object in this R-tree cross grids, a copy of the pointer to this
object has to be stored in all the 1st-level R-trees which belong to the grids crossing this object's MBR. For the same reason, the updating operations of this kind of 2-level index have to visit different paths and affect different 1st-level R-trees, so as to be the same inefficient.

In view of that R-tree is a multi-level height-balanced tree, so another idea of creating 2-level index is to make the 1st-level index to be a height-balanced R-tree, and then creating the 2nd-level index based on it. In fact, it is to create a whole R-tree and take one middle level as the coupling point to combine the 1st-level and the 2nd-level index. Thus the redundancy of store can be avoided. However, In order to satisfy the requirement of height-balance, the differences of the 1st-level R-trees' extent areas may be very significant, and distributions in space may be skewed. Such 2-level structure is a departure from the traditional data management where data are partitioned with regular grids, and it still can not satisfy the requirement for Fig. 1. What is more, local data updating may result in the re-construction of the parts upper than the joint points, so as to reduce updating efficiency.

Based on the above observation, there should provide a novel 2-level index technique that can satisfy the following requirements.

1) Supports R-tree to be the 1st-level index, no redundant store, and has high query efficiency.
2) Conforms to the grid data partition customs and allows builders to take partitions according to application requirement. Each grid can have its 1st-level R-tree with arbitrary height and node capacity according to the distribution of data, and these trees have good performance.
3) Supports local updating of data, and the effects caused by local updates, such as locks and index reconstructs, to be limited in the local grids.
4) Supports bulk-loading and bulk-inserting.

1.2 Basic principle

We call this new method as H2R-tree (Hilbert 2-Level R-tree). It is a kind of 2-level index technique based on Hilbert R-tree. The main idea is partition off spatial data according to spatial grids so that each grid creates its own 1st-level R-tree; then group these grids according to the Hilbert value of their center; in the last, take the 1st-level R-trees in the grids as data objects, then create the 2nd-level index using Hilbert R-tree algorithm. The characteristic of the H2R-tree lies in that the leaf nodes do not store pointers to spatial data, but the pointers to the 1st-level R-tree, which is independent and need not to be height-balanced. Fig. 2 illustrates the structure of H2R-tree.

For the data in each grid, construct local 1st-level R-tree with different height

1.3 Hilbert R-tree

The linear codes of spatial data along Hilbert curve are named Hilbert codes. Hilbert curve is a kind of space-filling curve (SFC). SFC can and only visits each data object in grid space once and never cross itself. SFC family includes Peano curves, Hilbert curves and Gray curves, etc. Generally, Hilbert curves have the best spatial cluster property[11]. \(H_2\) and \(H_3\) in Fig. 3 are the
Hilbert curves of order 2 and 3. The orders that Hilbert curves traverse grids are called Hilbert codes. For example, Fig. 3 shows such an ordering for a \(4 \times 4\) grid (see curve \(H_2\)), the point (0, 0) has a Hilbert code of 0, while the point (1, 1) has a Hilbert code of 2. Hilbert code is recursive, continuous and nonmonotonic.

![Fig. 3 Hilbert curves of order 1, 2 and 3](image)

The idea of bulk-loading based on Hilbert codes is to group the spatial data according to the Hilbert codes of MBR centers, and create R-tree from bottom to top. This algorithm can make the space utilization approximately approach 100\%, and searching efficiency is better than R-tree (nodes splitting with square complexity), R*-tree and the bulk-loading method presented by Roussopoulos\(^5\).

On this basis, Kamel put forward Hilbert R-tree\(^1\), so that newcome data can find appropriate position for inserting into a built R-tree. The node structure of Hilbert R-tree is different from that of R-tree. The inner nodes store data items of Child-MBR, Child-Ref, Child-LHV, where Child-MBR is the MBR of child node, Child-Ref is the pointer to child node, Child-LHV is the largest Hilbert value of the Hilbert values of the data items’ MBR centers. Algorithms for searching, inserting and deleting can be found in Reference \(^4\). Hilbert R-tree can keep spatial data ordering according to Hilbert value when it is updated dynamically. This property guarantees updates be limited in local fixed grids.

### 1.4 Index structure and algorithms

#### 1.4.1 Node structure

1) The 1st-level R-tree node structure is the same as that of common R-tree.

2) Inner node structure of the 2nd-level Hilbert R-tree is the same as that of the common Hilbert R-tree.

3) Leaf node structure of the 2nd-level tree is a little different from that of the common Hilbert R-tree. A leaf node only stores \(C_i\) entries of the form (Cell-R-tree-Ref, Cell-HV), where \(C_i\) is the capacity of the leaf, and Cell-HV is the Hilbert value of the grid center. Cell-R-tree-Ref is a reference of 1st-level R-tree, and such a reference may be either pointer to the index file or the file path. Especially, if a grid contains no data item, then its 1st-level R-tree is created as a null tree, namely its MBR is set to be invalid MBR, and the Cell-R-tree-Ref points to the null tree. Notice that leaf nodes in the 2nd-level Hilbert R-tree do not store the MBRs of the 1st-level R-tree items, for these MBRs can be accessed through Cell-R-tree-Refs, so they need not being stored again in the leaf nodes’ data items of the 2nd-level Hilbert R-tree.

#### 1.4.2 Loading

There are four steps to load H2R-tree. Firstly, builders divide the data space to regular grids and calculate the Hilbert codes of the centers. Secondly, the 1st-level R-tree of each grid is created with the spatial data whose MBR centers lie in the grid, and the process of data loading can be realized by any kind of OBO or bulk-loading method. If some a grid is empty, a null 1st-level R-tree is created and the root MBR is set to be an invalid rectangle, which does not participate in the calculation of MBRs for the father nodes. Thirdly, the 1st-level R-trees are lined up according to the Hilbert codes of the related grids, and then divided into \(C_1\) groups, where \(C_1\) is the capacity of the leaf node. Finally, the 1st-level R-trees of each group are looked as data and the 2nd-level R-tree are created on them.

#### 1.4.3 Searching

The searching algorithm of H2R-tree is similar to the one used in classical R-tree, but there are four steps. The first step is filtering the 2nd-level Hilbert R-tree, namely retrieving the candidate leaf nodes by checking the intersections between the query window and the nodes’ MBRs. The second step is exactly searching the 2nd-level Hilbert R-tree, namely retrieving the 1st-level
R-trees whose root MBRs intersect the query window, by checking up whether or not the searching window intersects the MBRs of the 1st-level R-trees referenced by Cell-R-tree-Ref in the candidate leaf nodes gotten from the first step. The third step is filtering the 1st-level R-tree accessed, namely retrieving the candidate leaf nodes by checking the intersections between the query window and the nodes' MBRs. The final step is the exactly searching among above candidates, retrieving and returning the exact query results, by checking up whether or not the searching window intersects the MBRs of the spatial objects referenced by Obj-Ref in the candidate leaf nodes.

1.4.4 Insertion
The insertion of H2R-tree can be achieved with the following two step.

Step 1: assign new data into local grids according to the value of its central coordinates. Here is the procedure: calculate the Hilbert codes of the data according to their MBRs' central coordinates; and then search the 2nd-level Hilbert R-tree using the Hilbert values to get the target 1st-level R-tree prepared to insert the newcomer data.

Step 2: for each grid that is assigned with new data, insert the new data into the corresponding 1st-level R-tree. As to the practical bulking-inserting of the 1st-level R-trees, any dynamic methods such as OBO, GBI, and buffer tree can be adopted. However, if the old data volume in local grid is not too big, then the methods of loading once such as Hilbert bulk-loading and STR can be used to rebuild the tree with all data.

1.4.5 Deletion
To process deletion based on H2R-tree, a query must be processed firstly to search the 1st-level R-tree storing the object to be deleted, and then delete it from this 1st-level R-tree. If the deletion results in the change of the root node's MBR of the 1st-level R-tree, then the leaf node MBR of the 2nd-level Hilbert R-tree that stores this R-tree must be updated, and update upward all the ancestor nodes affected. However, to delete the 1st-level R-tree may make this R-tree become a null tree. For such special case, we still reserve this 1st-level R-tree (namely an empty index file) and its reference in the leaf node of the related 2nd-level Hilbert R-tree, and set the MBR of the 1st-level null R-tree to be an invalid MBR, and then update the MBR of each ancestor node from the leaf nodes of the related 2nd-level Hilbert R-tree upward to avoid invalid searching. If many a 1st-level R-trees in grids with neighbor Hilbert codes are null, maybe all the MBRs in some leaf nodes of the 2nd-level R-tree be set as invalid MBRs. Then we will set the leaf nodes' MBRs to be invalid ones, and have to update the MBRs of each level upward from the beginning of the father node to the root node.

2 Property analysis and test

2.1 Properties of H2R-tree
H2R-tree has the following superior properties.

1) Conforms to the tradition of grid management, increasing little redundancy. Builders can partition off the grids according to requirements while each data object is indexed only once.

2) Searching H2R-tree is more efficient than searching a whole R-tree. The classical R-tree is height-balanced, and each branch has the same levels from root to leaf. So for highly skewed spatial data, wherever the query window falls into, the searching process has to visit the same levels until leaf node. While H2R-tree can overcome this shortage, since builders can know the distribution characteristics of the spatial data in advance, they can divide the data region into grids more appropriately using their experience. The 1st-level R-trees in local grids can be R-trees with different height corresponding to the numbers of data in the grids. If there is no data object in grid, even null tree can be created. If the query window falls into the spatial area of few data objects, then the height of visiting can be reduced greatly, and then higher efficien-
cy can be achieved.

3) H2R-tree can realize efficient local updating. Whatever insertion and deletion, the influences brought by them have been limited in the 1st-level R-trees. Only when the root MBRs of the 1st-level R-trees are also changed, the MBR of the related leaf nodes of the 2nd-level Hilbert R-tree will have to be updated. Furthermore, by the means of "null R-tree" and "invalid MBR", the structure of the 2nd-level Hilbert R-tree will not change. Therefore, the lock caused by local updating is limited in local grids.

4) H2R-tree supports management for distributed data, and the realizations and operations are simple. The 1st-level R-tree in each grid is stored in the local server in the disk file mode, and the 2nd-level Hilbert R-tree can either be stored in main server in the mode of disk file, or be built in the main memory directly. The query searching operation of H2R-tree is basically the same to common height-balanced R-tree.

5) H2R-tree supports bulk-loading and bulk-inserting, and it is compatible with any existing algorithm of bulk-loading and bulk-inserting.

2.2 An example and test

Fig. 4 shows a map of Hong Kong. There are 16 672 line features in the region of Fig. 4, and the distribution is highly skewed. Data in Fig. 4 are divided into 8 × 8 grids, as is illustrated in Fig. 5. Firstly, we used the spatial data whose centers are in the same grids to construct the 1st-level R-trees, and the node capacity was set to be 16. The bold black rectangles in Fig. 5 are the MBRs of the 1st-level R-trees in each grid. Next, the Hilbert code of each grid's center was calculated, and the Hilbert curves and float numbers was illustrated in Fig. 6. To make float coordinates more accurate, we adopted float Hilbert codes, and the detailed calculating algorithm can be found in Reference [7]. In Fig. 7, the numbers labeled in each grid are the data objects' count and the height of the related 1st-level R-tree in this grid. To test and compare with this Hilbert R-tree, a whole classical Hilbert R-tree was also created with the same data in Fig. 4. The node capacity of this Hilbert R-tree was also set to be 16, and the height was 5 after creating. It is obvious that in Fig. 7 many R-trees' heights in the grids with few data are less than 5, so we can conclude that the searching efficiency in such regions are definitely better than those of the whole Hilbert R-tree.

We test two groups of window queries distributed evenly in Fig. 4 to validate above conclusion. The heights and widths of the query windows in the first group are less than those of the grids, while the heights and widths of the query windows in the second group are wider than those of the grids. Experiments show that the count of disk operations based on the H2R-tree is less than that based on the whole Hilbert R-tree, especially for the first group.

Fig. 4 Sample map with 16 672 Data

Fig. 5 MBRs in the grids after grid division
3 Conclusions

It can be concluded that H2R-tree has many superior properties. So this algorithm is worth to be popularized with the improvement of requirement for the management of mass distributed data. We will proceed with in-depth research with more general data in future work. Furthermore, as the service is aimed to the distributed data, the future work will focus on the synchronization control based on the H2R-tree.

REFERENCES

[1] Guttman A (1984) R-trees: a dynamic index structure for spatial searching[C]. ACM SIGMOD Conference, Boston, USA

[2] Sellis T, Roussopoulos N, Faloutsos C (1987) R* tree; a dynamic index for multi-dimensional objects [C]. The 13th VLDB Conference, Athens, Greece

[3] Beckmann N, Kriegel H P, Schneider R, et al. (1990) The R*-tree; an efficient and robust access method for points and rectangles [C]. The ACM SIGMOD Conference, Atlantic City, USA

[4] Kamel I, Faloutsos C (1994) Hilbert R-tree; an improved R-tree using fractals [C]. The 20th VLDB Conference, Santiago, USA

[5] Roussopoulos N, Leifker D (1985) Direct spatial search on pictorial databases using packed R-trees [C]. ACM SIGMOD Conference, Austin, USA

[6] Kamel I, Faloutsos C (1993) On packing R-trees [C]. The 2nd CIKM Conference, Washington D C, USA

[7] Leutenegger S, Edgington J M, Lopez M A (1997) STR: a simple and efficient algorithm for R-tree packing[C]. The 13th IEEE ICDE Conference, Birmingham, England

[8] Chen Li, Choubey R, Rundensteiner E (1998) Bulk-insertions into R-trees using the small-tree-large-tree approach[C]. The 6th ACM GIS Conference, Washington D C, USA

[9] Choubey R, Chen Li, Rundensteiner E (1999) GBI-a generalized R-tree bulk-insertion strategy[C]. The 6th SSD Conference, Hong Kong, China

[10] Guo Jing, Guo Wei, Hu Zhiyong (2003) QR-tree; an efficient spatial index structure for GIS with very large spatial database[J]. Geomatics and Information Science of Wuhan University, 28(3): 306-310 (in Chinese)

[11] Faloutsos C, Roseman S (1989) Fractals for secondary key retrieval[C]. The 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, USA

[12] Arge L, Hinrichs K H, Vahrenhold J, et al. (2002) Efficient bulk operations on dynamic R-trees [J]. Algorithmica, 33(1): 104-128