A study on data point search for HG-trees

Joseph Kuan Paul Lewis

Multimedia Research Group
Dept. of Electronic and Comp. Science,
University of Southampton,
Highfield, Southampton,
United Kingdom.
Keywords: Algorithm, databases, information retrieval

1 Abstract

A point data retrieval algorithm for the HG-tree is introduced which improves the number of nodes accessed. The HG-tree is a multidimensional indexing tree designed for point data and it is a simple modification from the Hilbert R-tree for indexing spatial data. The HG-tree data search method mainly makes use of the Hilbert index values to search for exact data, instead of using conventional point search methods as used in most of the R-tree papers. The use of Hilbert curve values and MBR can reduce the spatial cover of an MBR.

Several R-tree variants have been developed; R*-tree, S-tree, Hilbert R-tree, and R*-tree combined with the linear split method by Ang et al. Our search method on the HG-tree gives a superior speed performance compared to all other R-tree variants.

2 Introduction

Multidimensional indexing methods have been widely used on multimedia systems, geographical information systems, and for many other purposes. Among these methods, the R-tree [12] is the most popular model, since it has features such as support for dynamic databases, low I/O costs, and auto-balanced trees. Other successful R-tree variants, R*-tree [4], X-tree [5], and Hilbert R-tree [14] also achieve satisfactory performance.

As the fast growing research continues on multimedia, rapid data retreival on large media databases is required. Media content based retrieval also demands efficient similarity matching, and several techniques for nearest neighbour search for R-tree type models are being developed [19] [13] [17].

In this paper, we investigate the point search issue on the HG-tree. The HG-tree [9] is a simple extension of the Hilbert R-tree to support point data only. However in [9], Cha and Chung do not indicate that the potential of the HG-tree can provide a very fast point search mechanism. The technique of using space filling curves to retrieve data has been studied a long time ago. Some of the early research by Cook [10] applied linear quadtrees (Morton codes or z-order filling curves) for accessing geographical data. Abel and Smith [1] applied linear quadtrees to retrieve geographical data in the form of rectangles. Most of the point search techniques in R-tree variants check whether the query point is overlapped by the bounding rectangle of a node. If so, the search is traversed to a lower level, recursively searching the overlapping nodes until point data is found. Instead of using this technique, we investigate the algorithm for searching points which is based on values from the Hilbert space filling curve.

A brief description of R-tree, Hilbert R-tree, and HG-tree is given in the next section, followed by studies on normal point search and the new point search based on Hilbert values in section 4 and 5 respectively. The experiments on the point search method use a database of point data containing 120,127 items varied from 2 to 16 dimensions, and compared against several R-tree variants; these are R*-trees [4], S-tree [2], Hilbert R-tree [14], HG-tree [9] with normal point search, and R*-tree combined with the linear split method by Ang et al. [3]. A detailed evaluation of these results is described in section 7.

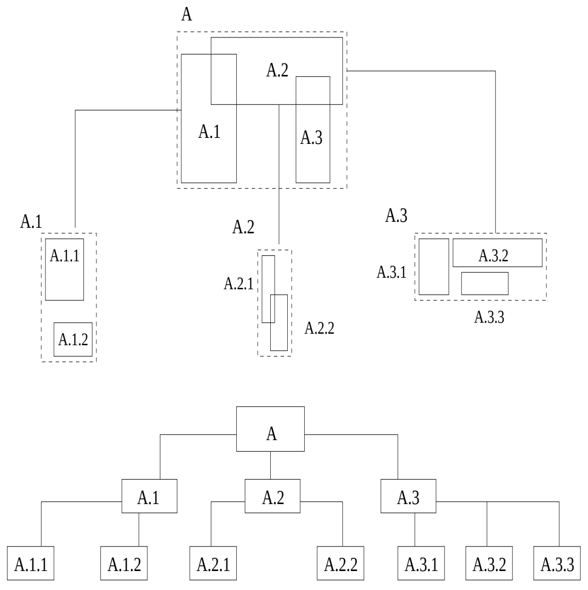

Figure 1: General structure of R-tree

3 R-tree, Hilbert R-tree, and HG-tree

The R-tree [12] is a multidimensional indexing tree that contains three types of nodes: root, non-leaf, and leaf nodes. A root node is located at the top level and it can either have non-leaf or leaf nodes as children nodes. Non-leaf nodes can either have leaf or non-leaf children, and leaf nodes are at the bottom of the tree that contains a set of data objects. The structure of the R-tree is automatically balanced, such that all the leaf nodes are at the same level. Each node in the R-tree accommodates a multi-dimensional minimum bounding rectangle (MBR). This MBR encapsulates all the node children's bounding rectangles. A set of child-pointers and children MBRs are also stored in each node, the child-pointer addresses where the child node is stored.

The Hilbert R-tree [14] uses the index values of the Hilbert space filling curves to represent n-dimensional data for indexing. The Hilbert space filling curve [6] [8] [20] basically maps from n-dimensional data to a one dimensional value, and vice versa. In [15], Jagadish shows that the Hilbert space filling curve has a better clustering property compared to other space filling curves. In the Hilbert R-tree, each child entry of a node is associated with a Hilbert indexing value, and the largest Hilbert value (LHV) among all the children Hilbert values is also kept inside the node. For insertion, the n dimensional data is first mapped to a Hilbert value and then the node that has the closest LHV is selected for insertion. The operation is processed recursively until the leaf node level. Since the single dimensional Hilbert values are used mainly in insertion and split operation, the construction of the Hilbert R-tree achieves a high speed performance and simpler implementation than other R-tree variants.

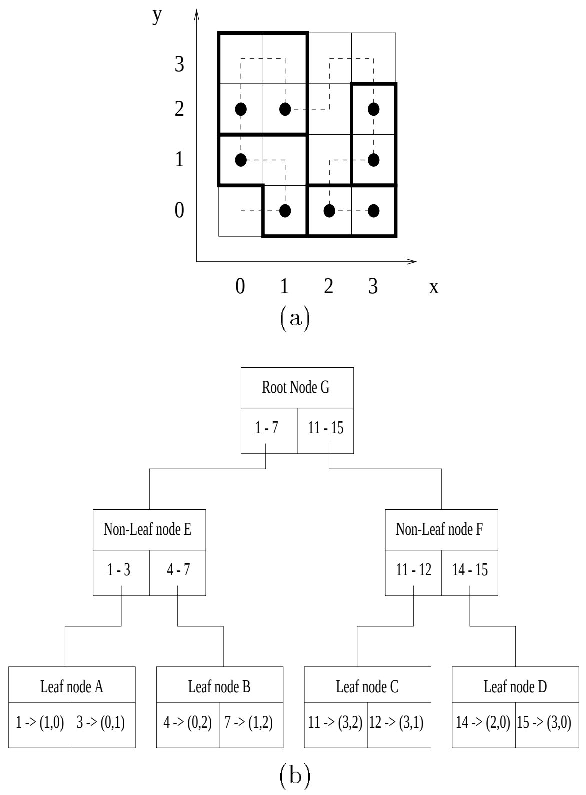

Figure 2: Example of HG-tree structure

The HG-tree [9] is a slight modification of the Hilbert R-tree for indexing point data only. Instead of having a single entry of Hilbert value, Cha et al. [9] use two Hilbert values, LHV and the smallest Hilbert value (SHV), to represent the interval of Hilbert values in each node. Each internal node has a list of tuples as (I, ptr), where I is the Hilbert values interval which plays a similar role to MBR and ptr is the address of a child node. However, we are doubtful about the value of this node structure as it does not seem to extract the correct MBR. For example, figure 2a illustrates a set of nodes with fanout of 2. According to the HG-tree node structure specified in [9], figure 2b shows the corresponding tree indexing structure.

From figure 2b, it appears to us that the only way to regenerate MBR with correct spatial information is to traverse down to the leaf nodes which is impractical. The alternative is to trace the Hilbert values interval and convert to coordinates while expanding the rectangle but this can cause drastic computation and large dead space in the MBR. In fact, it creates a slightly enlarged version of MBR if the stored data are real values which may cause overlap. For this reason, we use the node structure as (I, MBR, ptr) instead for the HG-tree throughout the evaluation.

4 Data object retrieval

To access data rapidly with multiple attributes is essential in multimedia database systems. The R-tree spatial indexing structure offers an efficient model with low disk access to index a data item. It is a well known fact that the performance of the R-tree in range search and nearest neighbour search degrades dramatically with an increase of dimension. The typical R-tree data object retrieval algorithm checks which child node's rectangle overlaps with the query object under all dimensions and proceeds in a recursive manner until the leaf node is encountered. Then all the items in the leaf node are matched against the query item. The following is the conventional algorithm for searching data objects:

```
1) PointSearch(QueryObject, RtreeNode)
2)   if (RtreeNode.type == LEAF)
     then
3)     for i in each child of RtreeNode
4)       for n in each dimension
5)         if (QueryObject.data[n] !=
               RtreeNode.child[i].data[n])
             return(NULL);
6)       return(RtreeNode)
     else
7)     for i in each child of Rtree_node
8)       if (QueryObject cover by
             RtreeNode.child[i])
9)         PointSearch(QueryObject,
                       RtreeNode.child[i])
```

5 Data object retrieval with Hilbert values

In this section, a point search algorithm for HG-tree is presented. The method mainly uses the Hilbert value for matching data items. If there is a data item that has an exact match of Hilbert value against the query item, then a further match on each attribute is necessary. It is likely to happen that close real data may have been mapped to the same Hilbert values, as the attributes are first rounded to the nearest integer coordinates and then transformed to a single dimensional data value. Conversely, it may even happen that a multi-dimensional data point has exactly the same real value attributes in a very large database. The following code demonstrates a new approach to searching for a point data item from the HG-tree:

```
1) HilbertPointSearch(QueryObject,
                      HGtreeNode)
2)   if (HGtreeNode.type == LEAF)
     then
3)     for i in each child of HGtreeNode
4)       if (QueryObject.Hilbert ==
             HGtreeNode.child[i].Hilbert)
         then
5)         for n in each dimension
6)           if (QueryObject.data[n] !=
               HGtreeNode.child[i].data[n])
               return(NULL);
             return(HG-tree_node);
         else
           return(NULL);
     else
7)     for i in each child of HGtreeNode
8)       if ((QueryObject.Hilbert within
              HGtreeNode SHV and LHV) &&
             (MBR of HGtreeNode covers
              QueryObject))
9)         HilbertPointSearch(QueryObject,
                        HGtreeNode.child[i])
```

The above algorithm clearly shows that step 4 is only a simple integer matching operations, which is always in order of 1. Step 4 is an extra boolean operation compared with the normal search algorithm but it helps to reduce a number of calls to step 5. Although the HG-tree uses the range of Hilbert values instead of minimum bounding rectangles, only applying Hilbert values for point search is not robust enough, as unnecessary node access can be involved with query points that are not indexed in the database. This can be avoided by generating minimum bounding rectangles from the Hilbert values but overlap area can be induced between nodes and redundant search can occur.

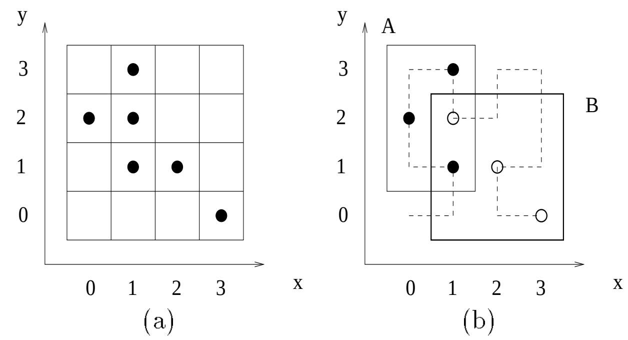

Figure 3: Example of overlapping area generation between nodes using Hilbert values

The reason that the above hybrid algorithm outperforms the PointSearch method is that the size of spatial search is minimized in some cases. Imagine a set of 2D integer data which are indexed into a tree with a branch factor of 3 as shown in figure 3a. According to the indexing algorithm of HG-tree, the Hilbert values are sorted in ascending order. Two nodes are created as demonstrated in figure 3b, rectangles A and B with the black and hollow dots respectively. Node A are indexed with Hilbert range values 2 – 6, and node B with values 7 – 15. Because of the nature of Hilbert space filling curves, overlapping area can be introduced between the minimum bounding rectangles. Therefore the normal spatial search at points (1, 1) and (1, 2) in figure 3b can involve both rectangles, whereas the HilbertPointSearch on points (1, 1) and (1, 2) as in Hilbert values of 2 and 7 respectively traverse to the correct node. If the point queried is location (3, 3) as in Hilbert value 9, the search on only Hilbert values can cause unnecessary processes. Figure 4a shows the reduced spatial size of the HilbertPointSearch, whereas the vertically and diagonally shaded area corresponds to nodes A and B respectively. However, there is an exception that the spatial search would not be smaller than the MBR if the start and end points in the Hilbert values range meet in the same neighbourhood, see figure 4b.

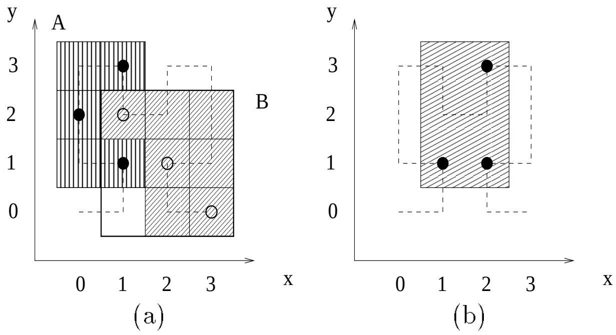

Figure 4: Spatial area of the HilbertPointSearch function

6 Evaluation

A set of image databases with various dimensionalities are constructed; they all have the same size of 120,127 feature vectors. The feature vector dimension ranges from 2 to 16 in steps of 2. The images are textures obtained from Brodatz [7], VisTex [18], and commercial catalogues. A texture feature extractor [16] is applied to these images and the first 16 features are used for the experiment.

The following indexing methods are used for evaluation in this paper:

1. Hilbert R-tree [14]
2. R*-tree [4]
3. S-tree [2]
4. R*-tree using Ang et al.'s new linear split [3].
5. HG-tree [9] with PointSearch function

The R*-tree [4] is one of the most successful R-tree variants and it has been used by a popular image query application, QBIC [11]. The R*-tree achieves a better storage utilization and retrieval than R-tree with re-insertion and improved split techniques. The S-Tree is a height imbalanced tree and it is constructed through two stages, binarization and compression. The binarization process organises all the data into a large binary tree form and the compression process creates an R-tree style tree by recollapsing nodes iteratively. The new linear split [3] by Ang et al. is processed by pushing the rectangles to either side in each dimension, such that the rectangles end up in the furthest separation with less overlap. They evaluate the split method with the R-tree model. However, we are interested in integrating

	Methods				
Dimension (node size)	1	2	3	4	5
2D (1K)	35, 4168	49, 3339	49, 4976	49, 3806	27, 4418
4D (2K)	45, 3294	55, 2891	55, 4263	55, 3535	38, 3337
6D (2K)	32, 4739	38, 4228	38, 6418	38, 4356	28, 4714
8D (4K)	52, 2856	59, 2681	59, 3721	59, 2770	47, 2828
10D (4K)	43, 3462	47, 3369	47, 5330	47, 3386	39, 3483
12D (6K)	55, 2695	60, 2642	60, 3669	60, 2701	51, 2694
14D (6K)	48, 3135	51, 3128	51, 4676	51, 3098	45, 3093
16D (6K)	42, 3571	45, 3523	45, 5406	45, 3584	40, 3540

Table 1: Fanout and Number of Pages For R-tree Variants

	Methods				
Dimension	1	2	3	4	5
2D	85.2%	76.4%	51.3%	66.5%	81.4%
4D	83.3%	77.4%	53.1%	63.6%	82.6%
6D	82.3%	78.1%	51.9%	75.2%	83.2%
8D	82.8%	77.9%	56.4%	75.2%	83.8%
10D	83.0%	78.0%	50.1%	77.6%	82.8%
12D	82.9%	78.1%	56.2%	75.8%	83.0%
14D	81.9%	78.1%	52.3%	78.0%	83.1%
16D	82.5%	78.0%	51.6%	76.7%	83.3%

Table 2: Storage Utilization of R-tree variants' structure

such a method with the R*-tree. The parameters of all experimental methods are those recommended in the papers. The only parameter adjusted for these experiments is the node size and it is shown in table 1.

Tables 1 and 2 show the node size, fanout, storage utilization, and number of pages created in each dimension by each method. In table 1, the numbers from 1 to 5 appearing on the top row represent the methods according to the above enumerated list. The first column is a list of dimensions, and along with each dimension the value inside the bracket is the node size set up in these experiments. For example, the first item in the dimension column, 2D (1K), means that the node size of each method is set to 1K byte in a 2 dimensional case. The column of each method has tuples which each correspond to the fanout of nodes, and number of pages created respectively. The number of pages created for the various methods in table 1 indicates the storage requirement to index all the data. For instance, in order to index 120,127 2D data with node size of 1K, Hilbert R-tree takes 4168 $\times$ 1K to store the data efficiently whereas R*-tree needs 3339K. This is because of the extra Hilbert values stored in Hilbert R-tree.

Table 2 is constructed in a similar manner and it indicates the results of storage utilization of each method. The storage utilization is a percentage of the number of objects stored in each node. The higher the value, the less space is wasted in each node. Among these methods, the S-tree is the method that only supports static databases. The fanout of the HG-tree shown in table 1 is the branch factor of root and non-leaf nodes, whereas the fanout of leaf nodes is the same as for the Hilbert R-tree. Generally, Hilbert curve variant trees (methods 1 and 5) achieve the best storage utilization (above 80%), whereas the R*-tree and R*-tree with linear split (methods 2 and 4) are under 80%. However, the S-tree (method 3) has only just above 50% storage utilization.

A phenomenon of the split algorithm by Ang et al. is observed by experimenting with large databases. The method can lead to a severe imbalance of split. When the dimension is low, in some cases the chances of having inbalanced split in each dimension may occur (please refer to the first 5 lines of code in [3].) Moreover, if there is a case that all the data objects to be split have the same attributes, then the algorithm

	Methods					
Dimension	1	2	3	4	5	6
2D	26.8	16.3	6.7	16.8	24.6	26.1
4D	14.8	63.8	6.2	45.7	16.0	15.9
6D	20.2	40.6	7.8	38.6	11.1	9.6
8D	17.2	41.1	5.3	58.0	8.6	5.5
10D	8.0	37.6	4.0	40.0	7.5	4.0
12D	7.4	35.9	3.5	41.2	7.2	4.1
14D	7.5	35.0	4.7	41.4	7.3	4.2
16D	8.3	29.9	5.6	34.0	7.4	4.3

Table 3: Average nodes accessed for each point search

can even return empty and overflow splits. Although this hardly arises for spatial data, there is more possibility for point data. We use the split technique of the R*-tree instead, if this condition of an empty split occurs.

Table 3 illustrates the main results of this paper, i.e. the average number of nodes accessed to retrieve a point from the trees. The performance is measured on a SGI machine with 180 MHz RS10000 CPU and 128 MB Ram. The column labelled with 6 is the HG-tree with HilbertPointSearch function.

The S-tree achieves the best performance in point search overall, since it calculates the optimized separation of the entire database. Although this may not be a fair comparison as the S-tree is designed for static databases only, this shows how large the performance difference is between static and dynamic models. After the 4 dimensional case, our point search method outperforms all the other methods except the S-tree model with less nodes accessed. The S-tree generally has fewer nodes touched than any method but with low storage utilization. From 6D case onwards, our method has comparatively fewer nodes touched than all other R-trees variants and has a very close performance to the S-tree.

7 Conclusion

A point search algorithm is introduced for the HG-tree which uses Hilbert curve values as the main matching target. This is different to the conventional approach of point search methods of R-tree variants. We compared the new approach with several other techniques, namely R*-tree, Hilbert R-tree, S-tree, R*-tree with the new linear split, and HG-tree with normal point search on a set of large databases (120,127 data) with various dimensions. The properties of these R-tree variants are also analysed. Since the S-tree is designed for indexing fixed size databases, it generally has the best performance due to its searches for optimal split of the entire database. However, the alternative point search technique with the HG-tree has very close node access rates to the S-tree in the higher dimensional cases. Compared to other methods, the results for our alternative point search technique show that it has fewer nodes touched. The main disadvantage of the HG-tree is the extra computation of minimum bounding rectangle from the range of Hilbert values required for point search, range search and nearest neighbour search.

References

[1] D. J. Abel and J. L. Smith. A data structure and algorithm based on a linear key for a rectangle retrieval problem. *Computer Vision, Graphics and Image Processing*, 20:1 – 24, 1983.

[2] Charu. C. Aggarwal, Joel L. Wolf, Philip S. Yu, and Marina Epelman. The S-tree: An efficient index for multidimensional objects. In *Fifth International Symposium on Spatial Databases*, pages 350 – 373, Berlin, Germany, July 1997.

[3] C. H. Ang and C. H. Tan. New linear node splitting algorithm for R-trees. In *5th International Symposium on Large Spatial Databases*, pages 339 – 349, Berlin, Germany, July 1997.

[4] N. Beckmann, H.P.Kriegel, R.Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD*, pages 322 – 331, May 1990.

[5] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd Very Large DataBase Conference*, pages 28 – 39, Mumbai (Bombay), India, 1996.

[6] T. Bially. Space-filling curves: Their generation and their application to bandwidth reduction. *IEEE Trans. on Information Theory*, IT-15(6):658 – 664, November 1969.

[7] P. Brodatz. *Textures: A Photographic Album for Artists & Designers*. New York: Dover, 1966.

[8] Arthur R. Butz. Alternative algorithm for Hilbert's space-filling curve. *IEEE Trans on Computers*, C-20(4):424 – 426, April 1971.

[9] Guang-Ho Cha and Chin-Wan Chung. HG-tree: An index structure for multimedia databases. In *Proceedings on IEEE Multimedia*, pages 449 – 452, Hiroshima, Japan, 1996.

[10] B. G. Cook. The structural and algorithmic basis of a geographic data base. In *Int. Advanced Studies Symposium on Toplogical Data Structures for Geographical Information Systems*, pages COOK/1–COOK/30, Dedham, Mass., October 1977.

[11] Myron Flickner, Harpreet Sawhney, Watne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *IEEE Computer*, 28(9):23 – 32, September 1995.

[12] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 45 – 57, 1984.

[13] Gísli R. Hjaltason and Hanan Samet. Ranking in spatial databases. In *Proceedings of the 4th Symposium on Spatial Databases*, pages 83 – 95, Portland, Maine, August 1995.

[14] Kamel. I and Faloutsos. C. Hilbert R-tree: an improved R-tree using fractals. In *Proc. of Int. Conf. of Information and Knowledge Management*, pages 490 – 499, 1993.

[15] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *ACM SIGMOD Conf.*, pages 332 – 342, May 1990.

[16] Joseph Kuan and Paul Lewis. Complex texture classification with edge information. In *Proc. of 2nd International Conference on Visual Information System*, pages 157 – 162, San Diego, CA, December 1997.

[17] Joseph Kuan and Paul Lewis. Fast k nearest neighbour search for the R-tree family. In *Proc. of First International Conference on Informations, Communications, and Signal Processing*, pages 924 – 928, Singapore, September 1997.

[18] Rosalind Picard, Chris Graczyk, Steve Mann, Josh Wachman Len Picard, and Lee Campbell. Vision texture 1.0 a, 1995. http://www-white.media.mit.edu/vismod/imagery/VisionTexture/vistex.html.

[19] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM-SIGMOD Intl.Conf. on Management of Data*, pages 71 – 79, San Jose, CA, June 1995.

[20] Hans Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.