A Greedy Algorithm for Bulk Loading R-trees *

Yván J. García R. Mario A. López Scott T. Leutenegger
Mathematics and Computer Science Department
University of Denver, Denver, CO 80208-0189
{ygarcia,mlopez,leut}@cs.du.edu

Abstract

Off-line loading of R-trees is useful to improve node utilization and query performance. We present an algorithm for bulk loading R-trees which differs from previous ones in two aspects: (a) it partitions input data into subtrees in a top-down fashion (based on the fact that splits close to the root are likely to have a greater impact on performance), (b) at each tree level, it considers all cuts orthogonal to the coordinate axes that result in packed trees and greedily picks those optimizing an arbitrary cost function. Extensive experimentation with both real and synthetic data indicate that for region data our algorithm requires up to three times fewer disk accesses than other algorithms. It is the method of choice for data with skew in locations, areas, or aspect ratios. Such data is common in practice.

1 Introduction

A search for spatial objects to satisfy a given multidimensional query is one of the most frequent operations in areas related to spatial databases. R-trees [3] are index structures for the efficient execution of multidimensional queries. Although R-trees are based on axis-aligned rectangles, arbitrarily shaped objects can be handled by storing their minimum bounding rectangles (MBRs).

R-trees can be updated incrementally [1, 5], however, incremental construction often produces trees with poor space utilization and structure. For static data, a preprocessing step may be used to build more query efficient R-trees. Examples of such pre-processing methods include [4, 6, 8].

In this paper we propose a new pre-processing algorithm, *Top-down Greedy Split* (TGS), that builds the tree top-down using a data adaptive step which rearranges the database objects under a node in construction. Trees built by TGS achieve up to three times improvement in query performance.

*This work has been partially supported by NSF grant IRI-9610240 and Colorado Advanced Software Institute grants TT-97-05 and TT-97-06.

```
Let n = number of input rectangles
Let S = maximum number of rectangles per subtree
Let M = maximum number of entries per node
Let f(r1,r2) be the "user-supplied" cost function
If n ≤ S return {stop condition}
For each dimension d
    For each ordering considered in this dimension d
        For i from 1 to ⌈n/M⌉ − 1
            Let B0 = MBR of first i·S rectangles
            Let B1 = MBR of the other rectangles
            Remember i if f(B0,B1) is better valued
Split input set and orderings at best position.
```

Figure 1: The basic split step

2 The Top-down Greedy Splitting Algorithm (TGS)

Unlike previous packing algorithms, TGS builds the R-tree top-down while using an aggressive approach that greedily constructs the various subtrees of the R-tree. The algorithm was motivated by two key ideas:

1. Minimize first the top levels since the potential for cost reduction is higher.

2. Consider all partitions induced by guillotine cuts such that resulting sub-trees are fully packed.

TGS recursively applies a *basic split step* which partitions a set of n rectangles into two subsets by a cut orthogonal to an axis. Such a cut must meet two conditions: (a) the cost of some objective function $f(r_1, r_2)$, where r_1 and r_2 are the MBRs of the two ensuing parts, is minimum, and (b) one subset has a cardinality $i \cdot S$ for some i where S is fixed per level so that the resulting subtrees are packed. The recursion is applied to both subsets until there is one subset per child.

A description of the basic split process is shown in Figure 1. Based on the extreme values defining the rectangle, we considered orderings on: min., max., both, center. A more detailed description of the algorithm, load time analysis, and results can be found in [2].

	Performance ratio HS/TGS				
data	10	25	50	100	250
VLSI	2.09	1.98	1.90	1.84	1.74
TILE	2.79	3.03	2.85	2.72	2.32
UNIF	1.32	1.32	1.33	1.33	1.34

Table 1: Point query, Hilbert vs TGS

	Performance ratio STR/TGS				
data	10	25	50	100	250
VLSI	2.16	2.14	2.11	2.10	1.98
TILE	2.29	2.44	2.32	2.23	1.93
UNIF	0.99	0.98	0.99	0.99	1.02

Table 2: Point query, STR vs TGS

3 Experimental Results

We now discuss the methodology used to provide a sound comparison of the packing algorithms and the results obtained from this comparison.

3.1 Methodology

For each data set, R-trees are built with each of the three packing algorithms Hilbert Sort (HS) [4], Sort-Tile-Recursive (STR) [6], and Top-down-Greedy-Split (TGS). Resulting trees are searched with the same sets of random queries. A query set has 10,000 equal size squares. Lower left corner of queries are generated as uniformly distributed points within the unit square. Queries were truncated to fit into the unit square.

We use LRU buffering and consider various buffer sizes. Our main metric is the average number of disk accesses for the 10,000 searches. We use the following data sets:

VLSI: A CIF data set of 453,994 rectangles provided by Bell Labs from the design of a chip [7].

UNIF: This synthetic set of 50,000 squares has a uniform distribution of both lower left corner locations and areas with area sum about 1 and truncation to fit unit square.

TILE: Tiles are created by randomly choosing a tile from the current set and splitting one dimension at a random point until obtaining 50,000 tiles.

These data sets were normalized to the unit square. We also tested other data sets (not shown) which resulted in similar conclusions.

3.2 Results

Our comparison metric is the ratio of the number of disk accesses needed for HS or STR over the number of disk accesses needed for TGS.

The maximum number of entries per node M is 100. We present results for buffer sizes of 10, 25, 50, 100, and 250 pages, one table column per each value.

TGS can consider any cost function and several orderings. We present the results for area as cost function since there was no significant difference when using the weighted perimeter as an objective function [4]. Only center point value ordering is presented since results do not significantly differ from using minimum and maximum ordering.

Two general trends were observed. First, TGS outperforms STR or HS under significant skew in location, area, or aspect ratio. As seen in Tables 1 and 2, TGS achieves an improvement of up to 3 times when dealing with data with skew. However there is little gain when dealing with uniform data. Modifications on data set distributions for locations, areas, aspect ratios lead to similar conclusions.

Second, the larger the region query, the smaller the performance difference as shown in Table 3. This performance difference decrease for larger queries is due to two factors. First, more of the data set is returned and thus allow for a more naive search without much degradation in performance. Second, as indicated by Kamel and Faloutsos model [4], the expanded queries have the effect of decreasing the skew TGS takes advantages of.

	10	25	50	100	250
	Query dimensions = 0.1 x 0.1				
HS/TGS	1.21	1.23	1.22	1.21	1.21
STR/TGS	1.12	1.13	1.12	1.11	1.11
	Query dimensions = 0.3 x 0.3				
HS/TGS	1.06	1.07	1.07	1.07	1.08
STR/TGS	1.02	1.03	1.03	1.03	1.03

Table 3: Region query, TILE

4 Conclusions

We found through extensive experimentation on actual implementations that R-trees constructed with our proposed bulk loading algorithm are up to three times more efficient than those built by STR and Hilbert. TGS is the method of choice for data sets with skew in location, area, or aspect ratio.

TGS outperforms other approaches due to two main factors: (a) TGS takes advantage of choices (closer to the root) likely to have a greater impact on query performance and, (b) by considering a larger collection of cuts, TGS is better able to adapt to the geometry of the input.

References

[1] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B., "The R^{*}-tree: an Efficient and robust Access Method for Points and Rectangles," Proc. ACM SIGMOD, 323-331, 1990.

[2] Garcia, Y.J, Lopez, M.A., Leutenegger, S.T., "A Greedy Algorithm for Bulk Loading R-Trees", University of Denver Technical Report # 97-02, www.cs.du.edu/users/leut/TGS.ps.

[3] Guttman, A., "R-trees: a Dynamic Index Structure for Spatial Searching," Proc. ACM SIGMOD, 47-57, 1984.

[4] Kamel, I., Faloutsos, C., "On Packing R-trees", Proc. 2nd International Conference on Information and Knowledge Management, 490-499, 1993.

[5] Kamel, I., Faloutsos, C., "Hilbert R-tree: An improved R-tree Using Fractals", Proc. International Conference on Very Large Databases, 500-509, 1995.

[6] Leutenegger, S.T., Lopez, M.A., Edgington, J., "STR: A Simple and Efficient Algorithm for R-Tree Packing ", Proc. 13th International Conference on Data Engineering, 497-506, 1997.

[7] Lopez, M.A., Janardan, R., Sahni S., "Efficient Net Extraction for Restricted Orientation Designs", IEEE Transactions on Computer-Aided Design, Vol. 15, No 9, 1151-1159, 1996.

[8] Roussopoulos, N, Leifker, D., "Direct Spatial Search on Pictorial Databases Using Packed R-trees,", Proc. ACM SIGMOD, 17-31, 1985.