Revisiting R-tree Construction Principles

Sotiris Brakatsoulas, Dieter Pfoser, and Yannis Theodoridis

Computer Technology Institute,
P.O. Box 1122, GR-26110 Patras, Hellas.
{sbrakats|pfoser|ytheod}@cti.gr

Abstract. Spatial indexing is a well researched field that benefited computer science with many outstanding results. Our effort in this paper can be seen as revisiting some outstanding contributions to spatial indexing, questioning some paradigms, and designing an access method with globally improved performance characteristics. In particular, we argue that dynamic R-tree construction is a typical clustering problem which can be addressed by incorporating existing clustering algorithms. As a working example, we adopt the well-known k-means algorithm. Further, we study the effect of relaxing the "two-way split procedure and propose a "multi-way" split, which inherently is supported by clustering techniques. We compare our clustering approach to two prominent examples of spatial access methods, the R- and the R*-tree.

1 Introduction

Classically, the term "Spatial Database" refers to a database that stores various kinds of multidimensional data represented by points, line segments, polygons, volumes and other kinds of 2-d/3-d geometric entities. Spatial databases include specialized systems like Geographical Information Systems, CAD, Multimedia and Image databases, etc.

However, the role of spatial databases is continuously changing and its importance increasing over the last years. Besides emerging new "classical" applications such as urban and transportation planning, resource management, geomarketing, archeology and environmental modeling, new types of data such as spatiotemporal seem to fall as well within the realm of spatial data handling. In expanding the scope of what defines spatial databases, the demands to the methods supporting such databases are altered. For example, traditionally in indexing the scope is on improving query response time, however by facing a more dynamic environment in which data is continuously updated/added, e.g., in a spatiotemporal context, other parameters such as insertion time gain in importance, e.g.,[18].

The key characteristic that makes a spatial database a powerful tool is its ability to manipulate spatial data, rather than to simply store and represent them. The basic form of such a manipulation is answering queries related to the spatial properties of data. Some typical queries are *range queries* (searching for the spatial objects that are contained within a given region), *point location* *queries* (a special case of a range query in which the search region is reduced to a point), and *nearest neighbor queries* (searching for the spatial objects that reside more closely to a given object). To support such queries efficiently, specialized data structures are necessary, since traditional data structures for alphanumeric data (B-trees, Hashing methods) are not appropriate for spatial indexing due to the inherent lack of ordering in multi-dimensional space. Multi-dimensional extensions of B-trees, such as the R-tree structure and variants [9, 3] are among the most popular indexing methods for spatial query processing purposes.

The vast majority of the existing proposals, including the original Guttman's R-tree that has been integrated into commercial database systems (Informix, Oracle, etc.), use heuristics to organize the entries in the tree structure. These heuristics address geometric properties of the enclosing node rectangles (minimization of area enlargement in the R-tree, minimization of area enlargement or perimeter enlargement combined with overlap increment in the R*-tree, etc.)[17, 8].

In this paper, we argue that the most crucial part of the R-tree construction, namely the node splitting procedure, is not more than a problem of finding some clusters (e.g. 2) in a set of entries (of the node that overflows). We investigate this idea and then go one step beyond by relaxing the "two-way" property of node splitting. By adopting a "multi-way" split procedure, we permit clustering to find real clusters, not just two groupings. We term the resulting R-tree variant that adopts clustering in its splitting procedure cR-tree.

The paper is organized as follows. Section 2 provides the necessary background on R-trees and, particularly, the R-tree node splitting procedure. Section 3 proposes an algorithm that incorporates a well-known clustering technique, namely the k-means, into this node splitting procedure. KMS (for k-means split), in general, finds k clusters. The choice between 2, 3 or … k clusters is based on the *silhouette coefficient* measure, proposed in [13]. Section 4 provides the experimental results, in terms of performance, speed and tree quality obtained. Section 5 briefly discusses the related work. Finally, Section 6 gives conclusions and directions for future work.

2 Spatial Indexing

R-trees [9] are extensions of B-trees [6] in multi-dimensional space. Like B-trees, they are balanced (all leaf nodes appear at the same level, which is a desirable feature) and guarantee that the space utilization is at least 50%. The MBR approximations of data objects are stored in leaf nodes and intermediate nodes are built by grouping rectangles at the lower level (up to a maximum node capacity M). Rectangles at each level can be overlapping, covering each other, or completely disjoint; no assumption is made about their properties.

2.1 Performance and Index Characteristics

R-tree performance is usually measured with respect to the retrieval cost (in terms of page or disk accesses) of queries. The majority of performance studies concerns point, range and nearest neighbor queries. Considering the R-tree performance, the concepts of node coverage and overlap between nodes are important. Obviously, efficient R-tree search requires that both overlap and coverage to be minimized. Minimal coverage reduces the amount of dead area (i.e., empty space) covered by R-tree nodes. Minimal overlap is even more critical than minimal coverage; for a search window falling in the area of n overlapping nodes, up to n paths to the leaf nodes may have to be followed (i.e., one from each of the overlapping nodes), therefore slowing down the search.

With the advent of new types of data, e.g., moving object trajectories, other index characteristics such as insertion time, i.e., the time it takes to insert a tuple into the index, gain in importance. A similar argument can be made about the actual size of the data structure comprising the index. With emerging small scale computing devices such as palmtops, the resources available to databases are tightened and large index structures might be unusable. Overall, the performance of an index should not only be measured in terms of its query performance but rather in terms of a combined measures that incorporates all the above characteristics.

2.2 On Splitting

Previous work on R-trees [3, 21, 8] has shown that the split procedure is perhaps the most critical part during the dynamic R-tree construction and it significantly affects the index performance. In the following paragraphs, we briefly present the heuristic techniques to split nodes that overflow. Especially for the R-tree, among the three split techniques (Linear, Quadratic and Exponential) proposed by Guttman in the original paper [9], we focus on the Quadratic algorithm, which has turned out to be the most effective in [9] and other studies.

R-tree (quadratic algorithm) : Each entry is assigned to one of the two produced nodes according to the criterion of minimum area, i.e., the selected node is the one that will be least enlarged in order to include the new entry.

R*-tree : According to the R*-tree split algorithm, the split axis is the one that minimizes a cost value S (S being equal to the sum of all margin-values of the different distributions). At a second step, the distribution that achieves minimum overlap-value is selected to be the final one along the chosen split axis.

On the one hand, the R-tree split algorithm tends to prefer the group with the largest size and higher population. It is obvious that this group will be least enlarged, in most cases [21]. A minimum node capacity constraint also exists; thus a number of entries are assigned to the least populated node without any control at the end of the split procedure. This fact usually causes high overlap between the two nodes.

On the other hand, the distinction between the “minimum margin” criterion to select a split axis and the “minimum overlap” criterion to select a distribution along the split axis, followed by the R*-tree split algorithm, could cause the loss of a "good" distribution if, for example, that distribution belongs to the rejected axis.

3 Clustering Algorithms and Node Splitting

As already mentioned, the split procedure plays a fundamental role in the R-tree performance. As we described in Section 2, R-trees and R*-trees use heuristic techniques to provide an efficient splitting of $M + 1$ entries of a node that overflows into two groups: minimization of area enlargement, minimization of overlap enlargement, combinations, etc. This is also the rule for the vast majority of R-tree variations.

Node splitting is an optimization problem which takes a local decision according to the objective that the probability of simultaneous access to the resulting nodes after split is minimized during a query operation. Clustering maximizes the similarity of spatial objects within each cluster (intra-cluster similarity) and minimizes the similarity of spatial objects across clusters (inter-cluster similarity). The probability of accessing two node rectangles during a query operation (hence, the probability of traversing two subtrees) is proportional to their similarity (for the queries we study in this paper). Therefore, node splitting should:

- assign objects with high probability of simultaneous access to the same node, and
- assign objects with low probability of simultaneous access to different nodes.

Taking this into account, we consider R-tree node splitting as a typical $Cluster(N, k)$ problem, i.e., a problem of finding the "optimal" k clusters of N data objects, with $k = 2$ and $N = M + 1$ parameter values (Figure 1(b)). According to this consideration, we suggest that the heuristic methods of the before mentioned split algorithms could be easily replaced by a clustering technique chosen from the extensive related literature [20, 12, 13, 10].

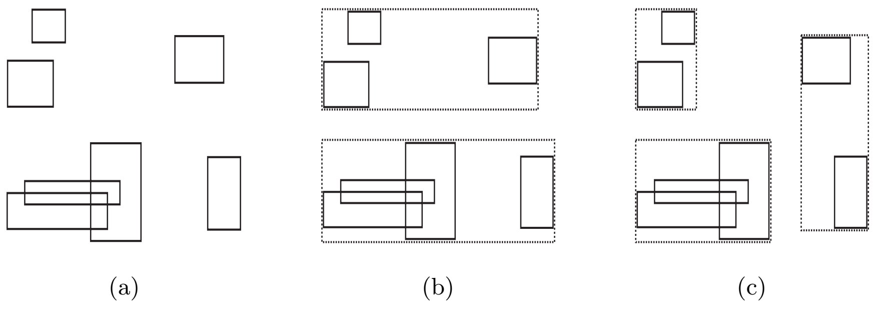

Fig. 1. Splitting an overflowing node into (b) two and (c) three groups.

Several clustering algorithms have been proposed, each of them classified in one of three classes: partitioning, hierarchical and density-based. Partitioning algorithms partition the data in a way that optimizes a specified criterion. Hierarchical algorithms produce a nested partitioning of the data by iteratively merging (agglomerative) or splitting (divisive) clusters according to their distance. Density-based algorithms identify as clusters dense regions in the data.

3.1 k-means Clustering Algorithm

Since we consider R-tree node splitting as a problem of finding an optimal bipartition of a (point or rectangle) set, we choose to work with partitioning algorithms. Among several existing techniques, we have selected the simple and popular k-means algorithm. The selection of k-means is due to the following reasons.

- The k-means clustering algorithm is very efficient with respect to execution time. The time complexity is $O(k \cdot n)$ and the space complexity is $O(n + k)$, thus it is analogous to the R-tree Linear split algorithm.
- K-means is order independent, unlike Guttman's linear-split heuristic.

Moreover, the page split is a local decision. Thus, the simplicity of k-means suits to the objective of the problem. Clustering algorithms that have been recently reported [24, 19] focus on handling large volumes of datasets, which is not our case.

```
Algorithm k-means  Divide a set of $N$ objects into $k$ clusters.
KM1 [Initialization]
        Arbitrary choose $k$ objects as the initial cluster centers.
KM2 [(Re) Assign objects to clusters]
        Assign each object to the cluster to which the object is the most
        similar, based on the mean value of the objects in the cluster.
KM3 [Update cluster centers]
        Update the cluster means, i.e., calculate the mean value of the objects
        in the cluster.
KM4 [Repeat]
        Repeat steps KM2 and KM3 until no change.
End k-means
```

As formally described above, to find k clusters, k-means initially selects k objects arbitrarily from the N-size data set as the centers of the clusters. Afterwards, in an iterative manner, assigns each object to its closest cluster, updates the cluster centers as the mean of the objects that have been assigned to the corresponding cluster and starts over. The iteration stops when there is no change in the cluster centers.

Before we proceed with the discussion of how to incorporate k-means into the R-tree construction procedure, we give give some details of the algorithm. We intend to apply k-means to form clusters of points or rectangles when a leaf node overflows and to form clusters of rectangles when an internal node overflows. For the purpose of showing dissimilarity, we define the *Euclidean distance* for any two shapes (this includes points and rectangles) to be the diagonal of the respective minimum bounding rectangle containing the respective shapes.

The *mean* of a set of objects is also a key parameter in k-means. Although the definition of the mean of a set of points may be straightforward, it is not true for the mean of a set of rectangles (e.g., during internal node splitting). We have adopted the following definitions.

The mean of N d-dimensional points $x_i(p_{i_1} \ldots, p_{i_d})$, $i = 1, \ldots, N$ is defined to be the following point.

$$\hat{x}\left(\frac{\sum_{i=1}^{N} p_{i_1}}{N}, \ldots, \frac{\sum_{i=1}^{N} p_{i_d}}{N}\right)$$

The mean of N d-dimensional rectangles $r_i(l_{i_1}, \ldots, l_{i_d}, u_{i_1}, \ldots, u_{i_d})$, $i = 1, \ldots, N$, where $l_{i_1}, \ldots, l_{i_d}$ the coordinates of the bottom-left corner and $u_{i_1}, \ldots, u_{i_d}$ the coordinates of the upper-right corner that define a rectangle, is defined to be the following point which corresponds to the center of gravity [16].

$$\hat{x}\left(\frac{\sum_{i=1}^{N} \frac{l_{i_1}+u_{i_1}}{2} area(r_i)}{\sum_{i=1}^{N} area(r_i)}, \ldots, \frac{\sum_{i=1}^{N} \frac{l_{i_d}+u_{i_d}}{2} area(r_i)}{\sum_{i=1}^{N} area(r_i)}\right)$$

3.2 Multi-Way Node Splitting

It is a rule for all existing R-tree based access methods to split a node that overflows into two new nodes. This number (i.e., two) origins from the B-tree split technique. While for B-trees, it is an obvious choice to split a node that overflows into *two* new ones, it cannot be considered as the single and universal choice when handling spatial data. To illustrate this point, an alternative splitting to that of Figure 1(b) could be the one in Figure 1(c).

By relaxing this constraint and by adopting the novel "multi-way" split procedure, we may reveal even more efficient R-tree structures. To our knowledge, it is the first time in the literature to overcome the "two-way" split property of multidimensional access methods [7] and this idea is implemented in the KMS algorithm (for k-means split) that we present next.

```
Algorithm KMS  Divide a set of $M+1$ entries into $k$ nodes
($2 \leq k \leq k_{max}$) by using k-means.
KMS1 [Initial Clustering]
        $k = 2$.
        Apply k-means on the $M+1$ entries to find $k$ clusters.
        Compute $\bar{s}(k)$. /* average silhouette width */
        $max = \bar{s}(k)$, $k_{opt} = k$
KMS2 [Repeating step]
        For k=3:$k_{max}$
                Apply k-means on the $M+1$ entries to find $k$ clusters.
                Compute $\bar{s}(k)$.
                If $\bar{s}(k) > max$ then
                        $max = \bar{s}(k)$, $k_{opt} = k$.
                end
        end
KMS3 [Assign entries to nodes]
        For $k$ = 1:$k_{opt}$
                Assign the entries of the $k$th cluster to the $k$th node.
        end
End KMS
```

KMS takes advantage of the k-means capability to find, in general, k clusters within a set of N points in space. In other words, KMS addresses the general Cluster(M, k) problem, thus it can be used to split a node that overflows into two, three, or k groups. This "multi-way" split algorithm is a fundamental revision of the classic split approach. In the rest of the section, we focus on algorithmical issues while in [4], we describe implementational details with respect to GiST (relaxing the "two-way" splitting of GiST is not straightforward at all).

Finding the Optimal Number of Clusters K-means requires the number k of clusters to be given as input. As described in literature, no a-priori knowledge of the optimal number k_{opt} of clusters is possible. In fact, comparing the compactness of two different clusterings of a set of objects and, hence, finding k_{opt}, is one of the most difficult problems in cluster analysis, with no unique solution [15].

To compare the quality of two different clusterings Cluster(M, k) and Cluster (M, $k+1$) of a point data set and, recursively, find k_{opt}, we use a measure, called *average silhouette width*, $\bar{s}(k)$, proposed in [13]. I.e., for a given $k \geq 2$ number of clusters, the average silhouette width for k is the average value of the silhouette widths, where the silhouette width of a cluster is the average silhouette of all objects in the cluster. In turn, the silhouette of an object is a number that indicates the closeness of an object to its cluster and varies in the range [-1, 1]:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where $a(i)$ and $b(i)$ are equal to the mean dissimilarity of object i to the rest of the cluster objects where it belongs and to the next closest cluster, respectively. The closer this value is to 1, the higher the object belongs to its cluster, compared to the rest of the clusters.

Having defined silhouettes $s(i)$ of objects and average silhouette widths $\bar{s}(k)$ of clusters, we now define k_{opt} to be the number k that gives the maximum average silhouette width, called *silhouette coefficient*, SC [13]:

$$SC = \bar{s}(k_{opt}) = \max_{2 \leq k \leq M} \bar{s}(k)$$

Hence, the clustering k_{opt} we select is the one that corresponds to average silhouette width equal to SC.

Restricting the Maximum Number of Clusters The silhouette coefficient is considered as a good measure to find the optimal number of clusters in [13]. However, in practice, it is expensive to set $k_{max} = M + 1$ in order to apply k-means for all possible k values. Instead of that, we considered k_{max} to be a parameter to be tuned and found that $k_{max} = 5$ was a "safe" choice. This choice is discussed in more detail in the extended version of this paper [4].

4 Experimental Results

This section presents the methodology used for the evaluation of our proposals and the obtained results in terms of speed of index construction, query performance, and index quality.

For a common implementation platform and for a fair comparison, we selected the GiST framework [11]. We used the original R-tree implementation included in GiST software package, but modified versions of the GiST framework were used allowing for an R*-tree implementation with forced reinsertion support and the realization of the cR-tree. The cR-tree differs from the R-tree only by its splitting routine, the rest of the R-tree construction procedure remains unchanged (e.g., the ChooseSubtree routine that traverses the tree and finds a suitable leaf node to insert a new entry). More details on GiST and our implementation can be found in [4].

In this study, we considered the following data sets (illustrated in Figure 2).

Random A synthetic data set of 80,000 points, generated by a random number generator that produced x- and y- coordinates.

Clustered A synthetic data set of 80,000 points, generated using the algorithm RecursiveClusters as introduced in [4].

Sierpinsky A fractal data set of 236,196 points generated by outputting the center points of the line segments of a fractal Sierpinsky data set. The generator used can be found in [23].

Quakes A real data set of 38,313 points, representing all epicenters of earthquakes in Greece during the period 1964-2000. It is publicly available through the web site of the Institute of Geodynamics, National Observatory of Athens, Greece [http://www.gein.noa.gr].

The construction of the indices was realized in the two step fashion followed by the GiST framework: bulk loading using the STR algorithm [14], as a first step, and successive insertions, as a second step. The following settings apply to all experiments we conducted.

- In each test case, 25% of the available data were used for bulk loading and the remaining 75% were used for dynamic insertions.
- The pagesize is set to 2Kbytes, thus corresponding to 55 two-dimensional points (leaf level) or 45 two-dimensional rectangles (intermediate level).
- For each test, we run 1,000 queries and present the *average* result.
- For each data set, the queries we exercise follow its distribution.

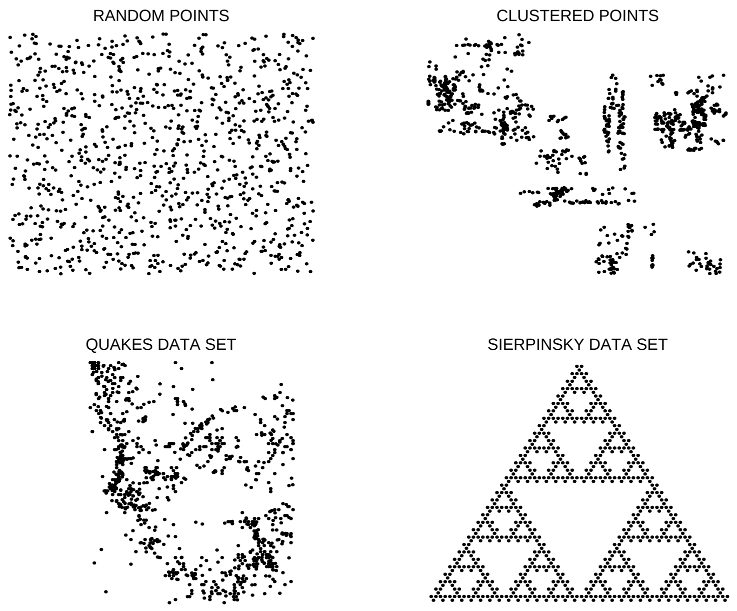

Fig. 2. 2-dimensional test data sets.

The following performance studies compares the cR-, R-, and R*-tree in terms of

Insertion Time The time needed for the second construction step of the indices (i.e., dynamic insertions of the 75% of the data sets) using an Intel 900MHz - 256MB RAM system.

Query Performance The number of I/O operations for range and nearest neighbor query loads.

Index Quality The quality of the indices, measured in terms of leaf utilization, sum of leaf node rectangles' perimeters, areas, and overlap.

4.1 Insertion Time

Besides the query performance of an index, the insertion time is equally important since it is a measure of robustness and scalability, while it is a critical factor for emerging applications that are required to manage massive amounts of data in a highly dynamic environment efficiently. We compare the three structures by measuring the time required for the second step of the tree construction (the dynamic insertions). The results appear in Figure 3. What can be observed is that insertions in the cR-tree can be done as fast as in the original R-tree and up to *six* times faster than in the R*-tree.

4.2 Query Performance

It is common practice in the spatial database literature to compare access methods in terms of node (or page) accesses for various query loads. We compare the performance of the R-tree variants for range and nearest-neighbor queries.

Figure 5 shows the experimental results for various range query sizes. Overall can be stated that the cR-tree performance is at the level of the R*-tree and, thus, also outperforms the R-tree. In particular, the performance of the cR-tree is almost identical to that of the R*-tree for the random, the clustered, and the sierpinsky data sets. For clustered data and small range queries, it even outperforms the R*-tree.

Similar results are also obtained for nearest neighbor queries (Figure 4). The cR-tree performs at the level of the R*-tree and clearly outperforms the R-tree.

4.3 Index Quality

The quality of an index, i.e., the resulting tree data structure, cannot be easily quantified and remains an open issue in the theory of indexability. Nevertheless, we have selected two factors as indicators of the quality of an index: *space utilization* and *sum of node rectangles perimeters* at the leaf level. The higher the space utilization the more compact the index, thus the less expensive its maintenance in terms of storage. The effect of this parameter in the R-tree performance also has been shown in [22] and other studies. The same is true for the perimeter [17]. The R*-tree has also revealed the effect of the perimeter (or, margin in [3]) as already discussed in Section 2.

Although the cR-tree does not impose any restrictions regarding the utilization of nodes resulting from a split, contrary to the R-tree (50% minimum utilization) and the R*-tree (40% minimum utilization), it achieves competitive space utilization as illustrated in Figure 6. The lowest value achieved is 66% for the Quakes data set, while the highest is 69% for the random data set.

The perimeter measure is better for the cR-tree as compared to the R-tree (see Figure 6). The improvement in the R-tree quality by incorporating clustering is a fact(40% - 60%). The quality of the cR-tree is in general close to that of the R*-tree (for clustered data even better).

The parameters related to the tree quality support the results of the query performance reported in the previous sections. Overall, the cR-tree data structure appears to be similar to the R*-tree. Thus, also its query performance is more similar to the R*-tree than it is to the R-tree.

4.4 Summary

The cR-tree query performance is competitive with the R*-tree and by far better than that of the R-tree. This query performance is achieved by not having to comprise on the insertion time. Here, the cR-tree is at the level of the R-tree and thus much faster than the R*-tree. The statistics collected on the index quality support the fact that the resulting tree data structure of the cR-tree is more similar to the R*- than to the R-tree.

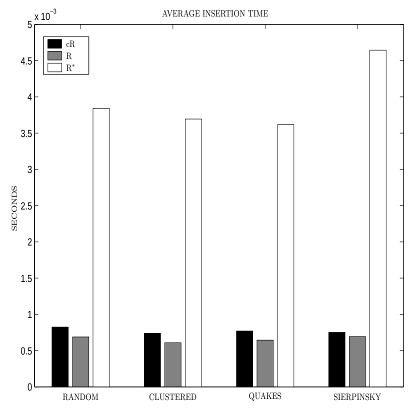

Fig. 3. Average time for one insertion

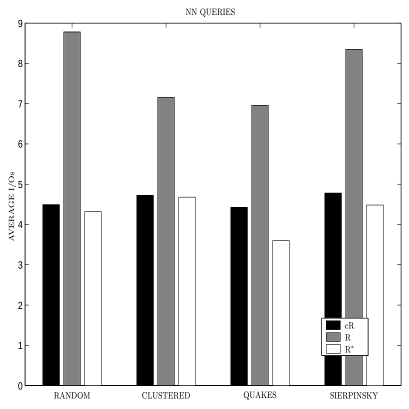

Fig. 4. Average I/Os for NN queries.

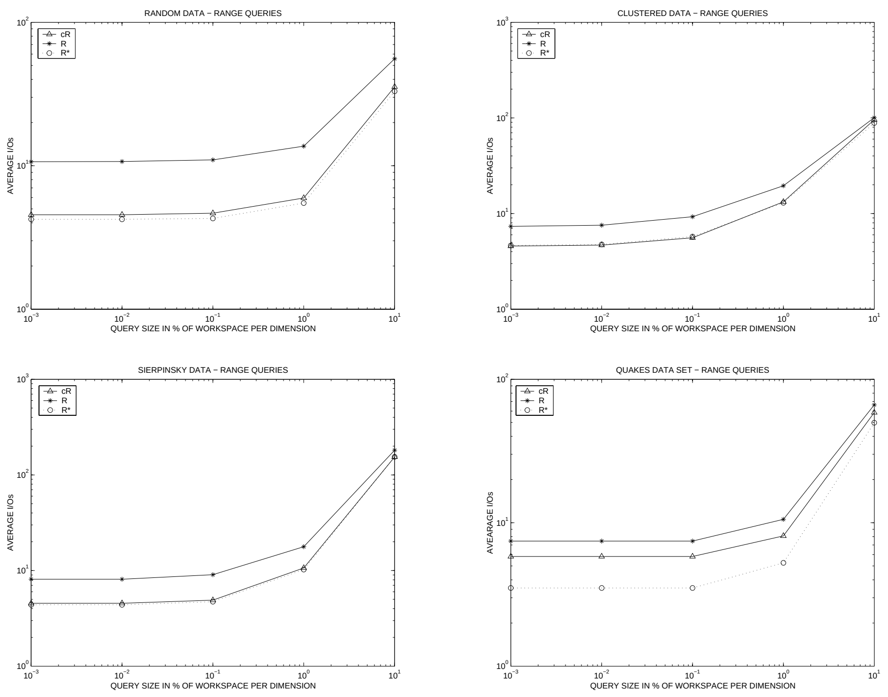

Fig. 5. I/O operations for range queries.

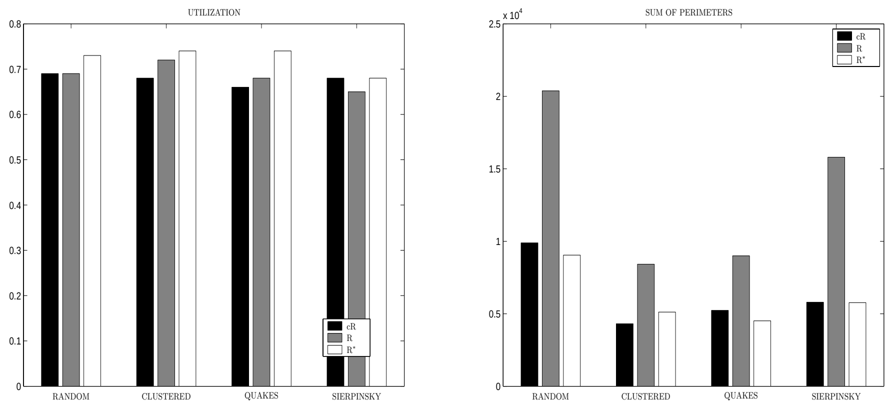

Fig. 6. Quality metrics for the leaf level.

5 Related Work

As already discussed, the vast majority of the R-tree based access methods uses heuristics to organize data (and to split nodes). An exception worth mentioning is [8]. This work proposes a locally optimal split algorithm of polynomial time and a more global restructuring heuristic, whose combined effects outperform R-trees and Hilbert R-trees. However, the extra time for local optimality does not always pay off since the index is dynamic and new insertions may retract previous decisions.

Related to the above, from a theoretical point of view, several researchers have addressed the following problem. Given a set of axis-parallel rectangles in the plane, the problem is to find a pair of rectangles R and S, such that (i) each member of the set is enclosed by R or S and (ii) R and S together minimize some measure, e.g., the sum of the areas. Algorithms that solve this problem in $O(n \cdot \log n)$ time have been proposed in [2].

In the field of coupling clustering and spatial indexing, to our knowledge, it is the first time to incorporate a clustering algorithm into a dynamic spatial access method. Related work includes [5], which proposed GBI (for Generalized Bulk Insertion), a bulk loading technique that partitions new data to be loaded into sets of clusters and outliers and then integrates them into an existing R-tree. That work is not directly comparable with ours, since it considers bulk insertions.

6 Conclusion

Spatial data are organized in indices by using several heuristic techniques (minimization of area or perimeter enlargement, minimization of overlap increment, combinations, etc.). In this paper, we investigated the idea to examine data organization, especially node splitting, as a typical clustering problem and replace those heuristics by a clustering algorithm, such as the simple and well-known k-means. We proposed a new R-tree variant, the cR-tree that incorporates clustering as a node splitting technique and, thus, relaxes the "two way" split property, allowing for "multi-way" splits. The main result of our study is the improved overall performance of the cR-tree. It combines the "best of both worlds," in that the insertion time is at the level of the R-tree and the query performance is at the level of the R*-tree. The fast insertion time makes the cR-tree preferable for data intensive environments. At the same time it does not rely on complex techniques such as forced-reinsertion that would reduce the degree of concurrency achieved due to the necessary locking of many disk pages. This makes the cR-tree also a suitable candidate for a multi-user environment. It can be seen that a simple clustering algorithm, which is easily implementable in practice, creates an access method that is fast in the tree construction phase without compromising in query performance. Consequently, the "multi-way" split deserves some more attention since it may result in very efficient indices. Additional improvements may be achieved by working towards the following directions.

- Tuning of k-means. A weakness of k-means is that it is sensitive to the selection of the initial seeds and may converge to a local minimum. Several variants have been proposed to address that issue [1].
- Further experimentation with other clustering algorithms, especially hierarchical algorithms, instead of k-means. Using a divisive (or agglomerative) hierarchical algorithm, KMS would not have to restart clustering for each k up to k_{max} (or down to 1, respectively).
- Apart from the silhouette coefficient measure proposed in [13], investigation of other criteria to find the optimal number of clusters.

References

[1] M. R. Anderberg. *Cluster Analysis for Applications.* Academic Press, Inc., 1973.

[2] B. Becker, P.G. Franciosa, S. Gschwind, S. Leonardi, T. Ohler, and P. Widmayer. Enclosing a set of objects by two minimum area rectangles. *Journal of Algorithms*, 21:520–541, 1996.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 322–331, 1990.

[4] S. Brakatsoulas, D. Pfoser, and Y. Theodoridis. Revisiting R-tree construction principles. Technical report, Computer Technology Institute, Patras, Greece, 2002. http://dias.cti.gr/~pfoser/clustering.pdf.

[5] R. Choubey, L. Chen, and E. A. Rundensteiner. GBI: a generalized R-tree bulk-insertion strategy. In *Proceedings of International Symposium on Spatial Databases*, pages 91–108, 1999.

[6] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–127, 1979.

[7] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, Volume 30(2):381–399, 1998.

[8] Y. J. Garcia, M. A. Lopez, and S. T. Leutenegger. On optimal node splitting for R-trees. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 334–344, 1998.

[9] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 47–57, 1984.

[10] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[11] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the 21st International Conference on Very Large Databases*, pages 562–573, 1995.

[12] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, Volume 31(3):264–323, 1999.

[13] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, 1990.

[14] S. Leutenegger, M. Lopez, and J. Edgington. STR: A simple and efficient algorithm for R-tree packing. In *Proceedings of the 12th International Conference on Data Enginnering*, pages 497–506, 1997.

[15] G. Milligan and M. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrica*, Volume 50(2):159–179, 1985.

[16] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, second edition, 1998.

[17] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance. In *Proceedings of the 12th ACM Symposium on Principles of Database Systems*, 1993.

[18] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. In *Proceedings of the 26th International Conference on Very Large Databases*, pages 395–406, 2000.

[19] S.Guha, R.Rastogi, and K.Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 73 – 84, 1998.

[20] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 1999.

[21] Y. Theodoridis and T. Sellis. Optimization issues in R-tree construction. In *Proceedings of the International Workshop on Geographic Information Systems*, pages 270–273, 1994.

[22] Y. Theodoridis and T. Sellis. A model for the prediction of r-tree performance. In *Proceedings of the 15th ACM Symposium on Principles of Database Systems*, pages 161–171, 1996.

[23] Leejay Wu and C. Faloutsos. Fracdim. Web site, 2001. URL: http://www.andrew.cmu.edu/~lw2j/downloads.html current as of Sept. 30, 2001.

[24] T. Zhang, R. Ramakrishnan, and M. Linvy. An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 103–11, 1996.