EXTERNAL BALANCED REGULAR (x-BR) TREES: NEW STRUCTURES FOR VERY LARGE SPATIAL DATABASES

MICHAEL VASSILAKOPOULOS AND YANNIS MANOLOPOULOS
mvass@computer.org *manolopo@csd.auth.gr*
Data Engineering Lab, Department of Informatics
Aristotle University, Thessaloniki 54006, Greece

The External Balanced Regular (x-BR) Trees constitute a family of new secondary memory structures which are suitable for storing and indexing multi-dimensional points and line segments. In 2 dimensions, the resulting structure is an External Balanced Quadtree, in 3 dimensions an External Balances Octtree, and in higher dimensions an External Balanced Hyper-quadtree. The main characteristic of all these structures is that they subdivide space (in an hierarchical and regular fashion) into disjoint regions. These spatial access methods are fully dynamic, while insertions are not complicated to program and affect only one path in the tree. Moreover, x-BR trees are variable resolution structures. That is, the number of space subdivisions is not predefined, making these structures suitable for very large amounts of data. Due to the balanced nature of these structures and the disjointness of the resulting regions, searches and other queries in these trees are processed very efficiently.

1 Introduction

Several spatial access methods have been proposed in the literature, for storing multi-dimensional objects (e.g. points, line segments, areas, volumes, and hyper-volumes). These methods are classified in one of the following two categories according to the principle guiding the hierarchical decomposition of data regions in each method.

- Data space hierarchy: a region containing data is split (when, for example, a maximum capacity is exceeded) to sub-regions which depend on these data only (for example, each of two sub-regions contains half of the data)
- Embedding space hierarchy: a region containing data is split (when a certain criterion holds) to sub-regions in a regular fashion (for example, a square region is always split in four quadrant sub-regions)

A widely used representative of the first principle is the family of R-trees (secondary memory structures). All the variations of this structure organize multidimensional data objects by making use of the Minimum Bounding Rectangles (MBRs) of the objects. This is an expression of the "conservative approximation principle". This family of structures is considered an excellent choice for indexing various kinds of data in spatial databases and Geographical Information Systems.

A famous representative of the second principle is the family of Quadtrees (in 3 dimensions Octtrees). Many variations suitable for storing various kinds of spatial data (points, line segments, images, etc) have appeared in the literature. As a logical structure the Quadtree is a four way tree and is thus most suitable for main memory (in contrast to an R-tree). The volume of spatial data makes the use of disk space unavoidable. Many implementations of Quadtrees have been developed for secondary memory.

The book by Samet [9] and the recent survey by Gaede and Guenther [3] provide excellent information sources on both families of structures for the interest reader.

A wish-list in the design of a spatial access method would describe an external structure with the following properties: balanced, simple, fast in answering queries, able to handle large amounts of data and with guaranteed space occupancy. In the present report, we present a new kind of tree structures which are based on Hierarchical Regular Decomposition of space (like Quadtrees) and are suitable for indexing multi-dimensional points and line segments. The nodes of these structures coincide to disk pages and the leaf nodes all appear at the same level (like R-trees [4]). The regions to which space is partitioned are disjoint (like R^+trees [8]). These structures are called External Balanced Regular (x-BR) Trees and are fully dynamic, while insertions are not complicated to program and affect only one path in the tree (in contrast to R^+trees).

Moreover, x-BR trees are variable resolution structures. That is, the number of space subdivisions is not predefined. Besides, the description of the region represented by a node is very compact, since it is formed incrementally, while descending the path to this node. These characteristics make x-BR trees suitable for very large amounts of data. The form of nodes in these structures has similarities to the form of nodes of Generalized BD trees [7]. GBD trees are based on k-d tree like decomposition of space, while x-BR trees on Quadtree (and variants of higher dimensions) like decomposition (which gives more regular shapes of regions, especially for higher dimensions). Moreover, splitting of internal nodes is handled in a more sophisticated way in x-BR trees and the encoding of region addresses is of variable size and the description of a region depends on all the levels.

The regions to which space is partitioned do not have necessarily quadrangular shape, but correspond to quadrangles from which smaller quadrangles have been excised. This is similar to the hB-tree [5], where space is partitioned according to k-d trees and "holey brick"-like regions are created. However, in contrast to the hB-tree, the partitioning in the x-BR tree follows the Data Space Hierarchy Principle (and specially partitioning in d equal hyper-rectangular subregions, for d-dimensional space). Moreover, unlike the hB-tree, each internal node has only one pointer to a child node and the entries of an internal node are address-pointer pairs and not tree structures (k-d trees, in the case of the hB-tree).

In case that x-BR trees are used for line segments, a segment is stored in a leaf, if it intersects the region of the leaf. This is analogous to PMR-Quadtrees [2,6,9]. However, unlike PMR-Quadtrees, in the usual implementation of which the indexing part of the structure is a quadtree (degree-four tree) in main memory, the indexing part of x-BR trees is a multiway disk based tree.

All the above characteristics, similarities with and differences from other structures make x-BR trees easy to program and efficient to use in answering queries about a very large number of stored points or line segments. Nevertheless, all these are achieved on the absence of a desired property: x-BR trees do not provide guarantee for space usage on internal nodes. However, their design aims at a rather good space usage, on the average case.

The rest of the paper is organized as follows. Section 2 describes the new structure (its leaf and internal nodes). Section 3 describes the subdivision of leaf nodes due to overflows during insertions, while Section 4 describes the subdivision of internal nodes. Sections 5 refers to searches (and queries in general) and deletions. We conclude in Section 5, suggesting also directions for future work.

2 x-BR Trees

Although x-BR trees can be defined for various dimensions, for the ease of exposition in the rest of the paper, we assume 2 dimensions. For 2 dimensions the hierarchical decomposition of space is that of Quadtrees (the space is subdivided in 4 equal subquadrants, any of which may be further subdivided recursively in 4 subquadrants).

The space indexed by an x-BR tree is a rectangle, expressed in a coordinate system of real numbers (not in a digitized space). The nodes of x-BR trees are disk pages and are distinguished in two kinds: leaves, which store the actual multidimensional data themselves and internal nodes, which provide a multiway indexing mechanism for these data.

2.1 Internal Nodes

Internal nodes contain pairs of the form (address, pointer). An address is used to distinguish the region of a child and is accompanied by the pointer to this child. Since addresses are of variable size, the number of pairs fitting in each node is not predefined. Apparently, the space occupied by all pairs within a node must not exceed the size of this node. The maximum size of an address is only limited by the node size and in practice it never reaches this limit. Each address represents a subquadrant which has been produced by quadtree-like hierarchical subdivision of the current space. It consists of a number of directional digits that make up this subdivision. The NW, NE, SW and SE subquadrants of a quadrant are distinguished by the directional digits 0, 1, 2 and 3, respectively. For example, the address 1 represents the NE quadrant of the current space, while the address 10 the NW subquadrant of the NE quadrant of the current space.

However, the region of a child is, in general, the subquadrant of the related address minus a number of smaller subquadrants. The region of this child is the subquadrant determined by the address in its pair, minus the subquadrants corresponding to the previous pairs of the internal node.

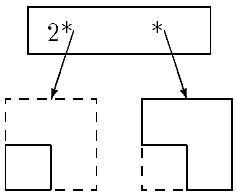

Figure 1: An x-BR tree with one internal node and two leaves

For example, in Figure 1 an internal node (in fact a root) that points to two leaves is depicted. The region of the root is the original space, which is assumed to have a quadrangular shape. The region of the left child is the SW quadrant of the original space. The region of the right child is the whole space minus the region of the first quadrant. The * symbols are used to denote the end of a variable size address. The address of the first child is 2*, since the region of this child is the SW quadrant of the original space. The address of the second child is * (has zero directional digits), since the region of the second child is the whole space minus the region of the first child. Each of these addresses is expressed relatively to the minimal quadrant that covers the internal node (each address determines a subquadrant of this minimal quadrant). In the example of Figure 1 the minimal quadrant for the depicted internal node is the whole space, since this node is the root.

During a search, or an insertion of a data element with specified coordinates, the appropriate leaf and its region is determined by descending the tree from the root. Initially, the region under consideration is the whole space (the region of the root). At each internal node that we visit, we examine its pairs in sequence. The address in each pair determines a subquadrant of the region under consideration. We choose the first pair with a subquadrant, that contains the given coordinates and follow the pointer to the next level. This sequential examination of pairs within each node in the path refines the region under consideration by intersecting it with the subquadrant of the chosen pair and by subtracting the subquadrants of the pairs appearing to the left of this pair.

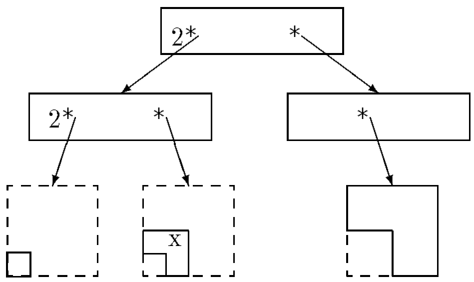

Figure 2: An x-BR tree with two levels of internal nodes.

In Figure 2 a situation where insertions caused the splitting of the left child of Figure 1 is depicted. This splitting caused a splitting of the internal node too and the creation of a new root. For example, a search for the data element marked with "x" proceeds as follows. We visit the root. We examine, in sequence, the pairs appearing in the root. The address of the first pair (2*) determines the SW quadrant of the whole space, which contains the coordinates of "x". We follow the pointer of the first pair and reach the leftmost child of the root. We examine, in sequence, the pairs appearing in this node. The address of the first pair (2*) determines the SW subquadrant of the SW quadrant of the whole space, which does not contain the coordinates of "x". The address of the second pair (*) determines the rest of the SW quadrant of the whole space, which contains the coordinates of "x". We follow the pointer of this pair and reach the leaf containing "x".

Although, for the sake of presentation, Figures 1 and 2 depict trees with nodes having one or two children, note that nodes are disk pages and they are likely to have a significant number of children (x-BR trees are multiway trees).

2.2 Leaf Nodes

External nodes (leaves) simply contain the data elements and have a predetermined capacity C. When C is exceeded, due to an insertion, the leaf is partitioned according to hierarchical decomposition (quadtree like decomposition), until the resulting two regions all contain data elements that are more than xC and less than $(1-x)C$, $0.5 < x < 1$. The choice of x affects the number of necessary subdivisions of an overflowed node and the size of addresses that result from a node split. A value closer to 0.5, in general, results in more subdivisions and larger addresses, since it is more difficult to partition the region of the leaf in subregions with almost equal numbers of elements. Of course, such a choice provides a better guarantee for the space occupancy of leaves. An alternative (adopted in PMR quadtrees) is to partition the leaf once and only once. Of course, overflow pages may be needed. In this case, a leaf is guaranteed to contain at most C plus the number of directional digits needed to reach this leaf and there is no minimum occupancy of leaf nodes.

For example, for $x = 0.75$, if many data elements are inserted near the NW corner of the region of the right leaf of Figure 2, the region of this leaf is split in four. If none of the subregions formed contains less than 3/4 C and more than 1/4 C data elements, the subregion containing the larger number of data elements is split in four. This subdivision is repeated recursively until a subregion that contains less than 3/4 C and more than 1/4 C data elements is created. Then the original leaf will split in two leaves: this subregion will represent the region of the left of the two resulting leaves. The rest of the original region is the region of the new right leaf. Following this policy, both leaves created will be at least 1/4 full. This situation is depicted in Figure 3.

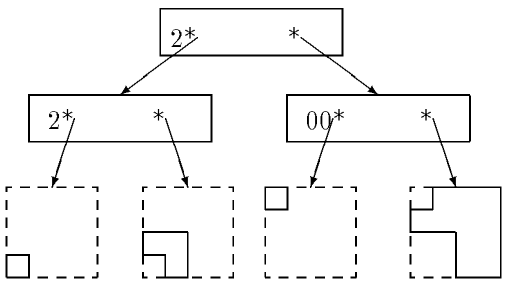

Figure 3: The x-BR tree after splitting the rightmost leaf of the tree in Figure 2.

2.3 Splitting of Internal Nodes

When an internal node overflows, it is split in two. The goal of this split is to achieve the best possible balance between the space use in the two nodes. The split in x-BR trees is either based on existing quadrants or in ancestors of existing quadrants.

First, a quadtree is built that has as nodes the quadrants specified in the x-BR internal node. For example, in Figure 4.a an x-BR internal node is depicted (in fact, only the address of each pair is depicted). The addresses appearing in the node subdivide the region (quadrant) of the node. This subdivision is depicted in Figure 4.b. For each address appearing in this node we create a quadtree node (a square in Figure 4.c). The position and level of each node is determined by the respective address (each digit specifies a direction in the path from the root of the quadtree down to this node). In order to create some of these squares, the creation of quadtree internal nodes (represented by circles) may be needed (for example, the creation of squares for addresses 100 and 101 requires the creation of two circles). Besides, some squares may appear as ancestors of other squares (for example, the square of address 0* is a parent for the squares of addresses 00*, 01* and 02*). The address * specifies the root of the quadtree. The tree created is a variation of the classic quadtree (see discussion about non-condensed quadtrees in [10]).

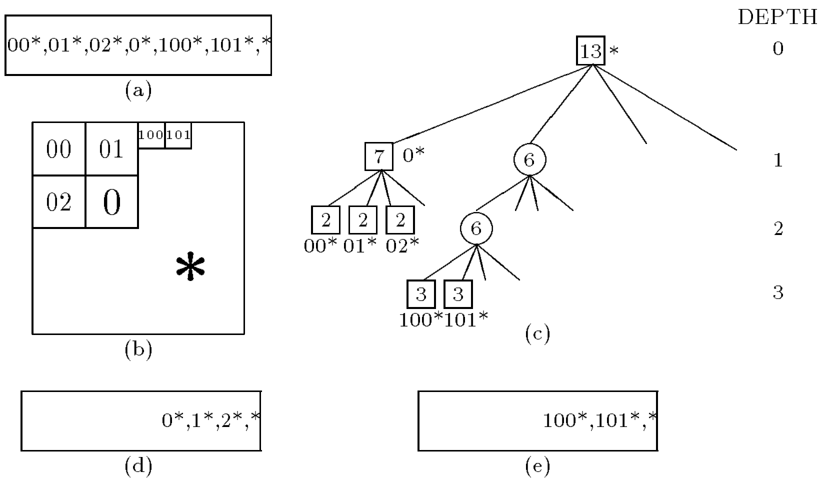

Figure 4: Splitting of an internal x-BR tree node.

To each square, as well as to each gray node, we assign the number of directional digits that will be freed when we eliminate the subtree rooted at this node. This number can be easily calculated with a bottom-up procedure. For example, in Figure 4.c each external square is assigned the number of digits that make up its address: the squares of 100*, 101*, 00*, 01* and 02* are assigned the values 3, 3, 2, 2 and 2, respectively. Each internal square is assigned the sum of values of its children plus the number of digits of its address. For example, the square of 0* is assigned the value 7=2+2+2+1. Finally, a circle is assigned the sum of values of its children only. For example, the second child of the root is assigned 6, since it has only one child (another circle) with value 6.

Then, we try to find a subtree that when eliminated we are left (approximately) with half of the total number of digits. This is accomplished with the following strategy.

- We start at the root.
- If we find a node that is assigned half of the total digits, the subtree we seek is rooted at this node.
- If this node is assigned a larger number of digits, we check its children. If none of them is assigned the requested amount and the largest assignment exceeds the requested amount, we check the related child.
- Otherwise, we choose the child with the largest assignment, or the node itself, as the best approximation.

For example, in the tree of Figure 4.c, the root is assigned number 13. This means that the approximate half of the number of digits is 6. The algorithm above goes though the following steps. We check the root. Since it is assigned a number larger than 6, we check its children. We discover that the leftmost child is assigned 7. The subtree we seek is rooted at this node, since by eliminating this subtree we are left with 13-7=6 digits. The resulting two nodes are depicted in Figures 4.d and 4.e. Of course, in the father node of the original internal node, the entry 0*, which corresponds to the minimal quadblock of the left of the resulting nodes, should be inserted.

2.4 Encoding of Region Addresses

The variable length coding of addresses can be done in various ways. In the following we present a simple, but quit effective encoding method (for more complicated methods, see [1,11]). For one binary integer x initially we form code γ that consists of two parts. The first has $\lfloor \log_2 x \rfloor$ 0s and one 1, while the second is the number $x - 2^{\lfloor \log_2 x \rfloor}$ in binary form, expressed with $\lfloor \log_2 x \rfloor$ bits. In Table 1, in the second column the γ encoding of the numbers of the first column are depicted. The code we finally use is δ that encodes the number $\lfloor \log_2 x \rfloor + 1$ with first part code γ (with the two parts of γ concatenated) and with second part the same to that of code γ (in binary form the number $x - 2^{\lfloor \log_2 x \rfloor}$). In Table 1, in the third column the δ encoding of the numbers of the first column are depicted. Code δ is larger than γ for most values $x < 15$, but beyond that, it is never worse.

x	code γ	code δ
1	1,	1,
2	01,0	010,0
3	01,1	010,1
4	001,00	011,00
5	001,01	011,01
6	001,10	011,10
7	001,11	011,11
8	0001,000	00100,000

Table 1: Examples of encoding.

3 Conclusions and Future Plans

In this report, a new family of spatial access methods was presented: External Balanced Regular (x-BR) Trees. These structures are fully dynamic, they subdivide space into disjoint regions and insertions are easy to program. Moreover, x-BR trees are suitable for very large amounts of data. Due to their balanced nature and the disjointness of the resulting regions, searches and other queries are expected to be processed very efficiently. x-BR trees do not provide a guarantee for space usage on internal nodes, although, their design aims at a rather good space usage, on the average case.

Future plans include the implementation of these structures and an extended experimentation for studying insertions and searches in comparison to other spatial access methods. The data stored could be collections of multidimensional points or line segments. Another possible research direction would be to analyze the storage and query answering performance of these structures by using probabilistic models (for example, extensions of the branching model presented in [10]).

Acknowledgments

This research was performed under the European Union's TMR Chorochronos project, contract number ERBFMRX-CT96-0056 (DG12-BDCN). The first author, who is a Post-doctoral Scholar of the State Scholarship Foundation of Greece, would like to thank this foundation for its financial assistance.

References

1. P. Elias: "Universal Codeword Sets and Representations of the Integers", *IEEE Transactions on Information Theory*, Vol.21, pp.194-203, 1975.
2. E.G. Hoel and H. Samet: "Efficient Processing of Spatial Queries in Line Segment Databases, Proceedings SSD'91 Conference, pp.237-256, 1991.
3. V. Gaede and O. Guenther: "Multidimensional Access Methods", *ACM Computer Surveys*, Vol.30, No.2, pp.170-231, 1998.
4. A. Guttman: "R-trees - a Dynamic Index Structure for Spatial Searching", *Proceedings ACM SIGMOD'84 Conference*, pp.47-57, 1984.
5. D. Lomet and B. Saltzberg: "The hB-tree: a Multiattribute Indexing Method with Good Guaranteed Performance", *ACM Transactions on Database Systems*, Vol.15, No.4, pp.625-658, 1990.
6. R.C. Nelson and H. Samet: "A Consistent Hierarchical Representation for Vector Data", *Computer Graphics*, Vol.20, No.4, pp.197-206, 1986.
7. Y. Ohsawa and M. Sakauchi: "A New Tree Type Data Structure with Homogeneous Node Suitable for a Very Large Database, *Proceedings 6th IEEE ICDE Conference*, pp.296-303, 1990.
8. T. Sellis, N. Roussopoulos and C. Faloutsos: "The R+tree - a Dynamic Index for Multi-Dimensional Objects", *Proceedings 13th VLDB Conference*, pp.507-518, 1987.
9. H. Samet: "*The Design and Analysis of Spatial Data Structures*", Addison-Wesley, Reading MA, 1990.
10. M. Vassilakopoulos and Y. Manolopoulos: "A Random Model for Analyzing Region Quadtrees", *Pattern Recognition Letters*, Vol.16, pp.1137-1145, 1995.
11. J. Zobel, A. Moffat and R. Sacks-Davis: "An Efficient Indexing Technique for Full-text Database Systems, *Proceedings 18th VLDB Conference*, pp.352-362, 1992.