Towards Logarithmic Search Time Complexity for R-Trees

Richard Göbel
University of Applied Sciences Hof, Alfons-Goppel-Platz 1,
95028 Hof, Germany

Abstract-**Index structures are frequently used to reduce search times in large databases. With index structures like the B-Tree search time grows only logarithmic with the size of a database for several types of searches. This means that the search time is almost constant for very large database systems even if their size grows significantly. Conventional index structures however do not well support searches specifying lower and/or upper bounds for more than one attribute (multidimensional range searches). Therefore R-Trees are increasingly used in this application context. A typical application domain of R-Trees are spatial database systems with two dimensional search conditions specifying upper and lower bounds for longitude and latitude values. Unfortunately R-Tree efficiency does not meet the expectations in many cases. Theoretical analysis of this problem showed, that search time grows much faster than logarithmic for two and more dimensional range searches in contrary to the one dimensional case. In this paper we prove that a logarithmic search complexity can be achieved for two dimensions, if the form of nodes is optimized relative to the form of search conditions. Based on this result, the paper investigates the form of nodes generated by different existing tree packing methods. Since existing methods fail to ensure the required form, a new tree packing method is proposed which improves the chance to meet the identified requirements.**

I Introduction

Index structures help to reduce search time for database systems. Without an index structure search time grows at least linearly with the number of "relevant" entries in the database. With an index structure it is possible to navigate to the search result instead of checking every individual entry. A frequently used index structure is the B-Tree [9]. It ensures a time complexity growing logarithmic with the number n of relevant entries for some queries. As an example, queries specifying a lower and an upper bound for a single column can be efficiently supported by a B-Tree (e.g. find all persons with an income between 50 k and 100 k). So called multi-dimensional range searches, specifying upper and/or lower bounds for more than one column however cannot be directly supported by the B-Tree (e.g. persons with an age between 18 and 21 and with an income between 50 k and 100 k). Different index structures have been proposed since the seventies (e.g. [4], [5], [6], [15]) for this type of search conditions. An overview on "Multidimensional Access Methods" is given by [12].

Multidimensional access methods can be divided into replicating (e.g. [6]) and non-replicating index structures (e.g. [15]). Replicating index structures ensure a logarithmic time complexity but their space requirements make it difficult to use them in real applications. On the other hand non-replicating index structures have moderate space requirements facilitating their application even in very large databases. Unfortunately theoretical analysis identified $\Omega(n^{(d-1)/d})$ (d number of columns/dimensions) as a lower bound for the time complexity of this type of index structures ([20], [18]).

During the nineties interest in multidimensional range searches increased significantly due to database applications in the domains Multimedia, Bioinformatics and Geographical Information Systems (GIS). Of special interest were spatial database systems as the backend for GIS, because almost all major database providers offer a spatial extension for their database engine. Many of them use the R-Tree ([15]) as a multidimensional index structure for supporting spatial searches. As a consequence research has investigated the R-Tree in more details and proposed several extensions and improvements (e.g. [24], [8], [7], [2], [10]). Of particular interest are so called tree packaging methods which generate a tree from a given set of entries (e.g. [23], [17], [19] and [13]). These tree packing methods improve the chance to optimize a tree according to some selected criteria. Although all proposed methods cannot be better than the identified lower bound, these methods might provide better results in the average case than incrementally generated R-Trees. On the other hand most of the mentioned methods may have significant longer search times in several cases than the identified lower bound. Only most recently a new tree packing method was proposed which guarantees this lower bound also as the upper bound for all searches [1].

In comparison to a logarithmic performance the lower bound $\Omega(n^{(d-1)/d})$ is probably not acceptable for very large databases (see for example [14], [11]). Logarithmic performance means that response times are almost constant for very large databases, if some entries are added.

Often experiments show better search times than the identified lower bound. Some theoretical analysis is available as well ([25], [16]). Unfortunately this analysis does not clearly identify conditions which can be easily checked for an application.

In general the efficiency of a database application based on R-Trees is not clear. Currently an application programmer takes the risk that database efficiency decreases significantly if the amount of data grows or changes. This paper addresses this gap by providing conditions under which efficient R-Tree support is guaranteed. These conditions refer to the form of nodes and search conditions and can be easily checked for a database.

The paper starts with a theoretical analysis of R-Tree performance in section II. Section III introduces a new tree packing method considering the identified conditions. The new method is applied to test data in section IV and compared with other tree packing methods. Section V summarizes the results.

T. Sobh (ed.), Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 201–206.

II Analysis

This section investigates the dependency of the number of visited nodes from the number of entries *n* and the size of the result set *m* (number of entries satisfying the search condition). For this purpose the section investigates the number of overlaps between a search condition and the nodes of an arbitrary level (depth) in a tree (e.g. all leaf nodes). This number multiplied with the depth of the tree provides an upper bound for the total number of nodes which need to be visited.

In this section we focus on two dimensional R-Trees. All nodes of a two dimensional R-Tree can be considered as rectangles in two dimensional space, where the sorted value spaces of the involved columns represent the two dimensions. A rectangle defines lower and upper bounds for both dimensions. A rectangle of a parent node includes all rectangles of its child nodes and the leaf nodes contain entries as points. Note that for an R-Tree all rectangles are minimal (minimal bounding rectangle). A search condition is also a rectangle (search rectangle). A search starts at the root node and continues recursively at all child nodes which overlap the search rectangle. On the leaf level all points are checked for the search condition.

It is probably obvious that the number of visited nodes grows at least linearly with the size *m* of the result set. The dependency between the number of entries and the visited nodes is however not obvious. In fact a search rectangle may overlap a large number of nodes without retrieving any results. If however every node contributes at least one entry to the search result, then the size of the result set *m* is an upper bound for the number of visited nodes (not more than *m* nodes need to be searched). This also means that the number of visited nodes would not be dependent on the total number of entries *n*.

Accordingly we will start our analysis by identifying cases of overlaps which contribute at least one entry to a search condition. Since we cannot avoid other types of overlaps we will derive conditions which limit overlaps not contributing entries.

For this analysis we distinguish between strong and weak overlaps. Two nodes weakly overlap, if borders of these two nodes just meet (common points are on borders of nodes). A strong overlap is given, if two nodes share common inner points. Figure 1 shows examples of weak and strong overlaps.

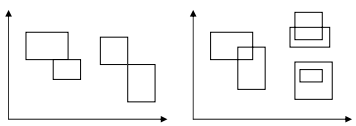

Figure 1: Weak (left) and strong overlaps (right)

Our analysis is based on the following general assumptions:

- For the rest of the paper we assume that nodes on a given level do not strongly overlap. This property can be guaranteed by tree packing methods for point data.
- For technical reasons we assume that lower bounds of ranges are always less (not equal) than upper bounds (arbitrarily small differences are still possible).

In the next section we will discuss a tree packing method satisfying these conditions facilitating efficient searching.

We start the analysis by identifying a condition under which a node contributes at least a single entry to a search region. It is probably obvious that a node contributes all its points if the node is fully included in the search region. In the two dimensional case it is already sufficient if one edge is included in the search region because an edge of a minimal bounding rectangle includes at least one point (otherwise the bounding rectangle would not be minimal). Figure 2 shows such an overlap between a search region (bounded by a dotted line) and a tree node (bounded by a solid line).

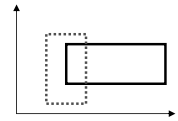

Figure2: Edge of node included in search region

Lemma 1 In a two dimensional R-Tree a node contributes a point to a search region if the search region contains at least an edge of the node.

All other cases of overlaps may not always contribute a point to a search result. There are however other cases for which the set of overlapping nodes can be limited to a constant number not depending on the number of entries in the database. One of these cases deals with the situation that nodes overlap a corner point of a search region.

The first question in this context is how many nodes may overlap a point if we have no strong overlaps between these nodes. We use Figure 3 to analyze the worst case. This diagram splits the neighborhood of a point into four different quadrants (**n**orth-**w**est, **n**orth-**e**ast, **s**outh-**w**est, **s**outh-**e**ast).

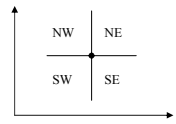

Figure3: Quadrants in neighborhood of point

A single node overlapping a point blocks:

- all quadrants if the point is inside the node (not on edge).
- two quadrants if the point is on an edge.
- a single quadrant if the point is the corner of a node

for further overlaps. As a consequence a point may be overlapped by at maximum 4 different nodes.

Lemma 2 Given is a set of two dimensional *NODES* without strong overlaps. Then a single point is overlapped by not more than 4 nodes.

A search rectangle has four corner points. This means that not more than 16 nodes overlap the corners of a search rectangle.

Lemma 3 Given is a set of two dimensional *NODES* without strong overlaps. Then the search region is overlapped by not more than 16 nodes which include corners of the search region.