Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 101 (2007) 78–85
Information Processing Letters
www.elsevier.com/locate/ipl

Bottom-up nearest neighbor search for R-trees ☆

Moon Bae Song, Kwang Jin Park, Ki-Sik Kong, Sang Keun Lee *

Department of Computer Science and Engineering, Korea University, 5-1, Anam-dong, Seongbuk-Ku, Seoul 136-701, Republic of Korea

Received 10 October 2005; received in revised form 10 April 2006; accepted 14 August 2006

Available online 18 September 2006

Communicated by J. Chomicki

Keywords: R-trees; Nearest neighbor search; Bottom-up search; Databases

1. Introduction

Given query point q, finding the nearest neighbor object, a so-called nearest neighbor (NN) search, is one of the most important problems in computer science, particularly within the database community. As a general form of these problems, the formal definition of NN search is the following. A point set P is a set of points in a d-dimensional data space DS, $P = \{p_0, p_1, \ldots, p_{|P|-1}\}$, $p_i \in DS \subseteq \mathbb{R}^d$. Given query point q, the result of the NN search is $NN(q) = \{p \in P \mid \forall p' \in P\colon \operatorname{dist}(p, q) \leqslant \operatorname{dist}(p', q)\}$.

Since 1984, when Guttman proposed his work [2], R-trees have become the most popular data structure for indexing multidimensional data for various purposes. Two different approaches for processing NN queries on R-trees exist. The first was developed by Roussopoulos et al. [5]. Owing to its search behavior, it is referred to as depth-first search (DFS) algorithm in the following. The second, called best-first search (BFS), was proposed by Hjaltason and Samet [3]. The BFS algorithm is known as an optimal NN search algorithm for R-trees. This means that it visits a node only if necessary. Since the top–down approaches visit from the root node to leaf node, the minimum I/O cost per query cannot be smaller than the height of the R-tree.

In this paper, a new NN search algorithm for R-trees is proposed, which performs in a bottom-up manner. In contrast to the two existing algorithms that work in a top–down manner, the proposed approach starts at the leaf level and traverses to the *Root* node. In order to select a more appropriate leaf node, it exploits in-memory hash structure in which whole data space is disjointly partitioned into $n \times n$ cells. Each cell contains the pointers of leaf nodes that intersect with the cell. The experimental results demonstrate that the proposed NN search algorithm outperforms existing NN search algorithms which are based on the R-tree including the existing I/O optimal algorithm. This is the first work to overcome the performance limitation of the BFS algorithm.

2. Nearest neighbor search using R-trees

As a d-dimensional extension of B^+-tree, the R-tree was proposed by Guttman [2]. In the data structure, a geometric object is represented by its minimum bounding rectangle (MBR). An MBR is minimal approximation of a geometric object and multidimensional rectangle $R = [l_1, u_1] \times \cdots \times [l_d, u_d]$ in the data space.

☆ This work was supported in part by MIC & IITA through IT Leading R&D Support Project.

* Corresponding author.
E-mail address: yalphy@korea.ac.kr (S.K. Lee).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2006.08.005

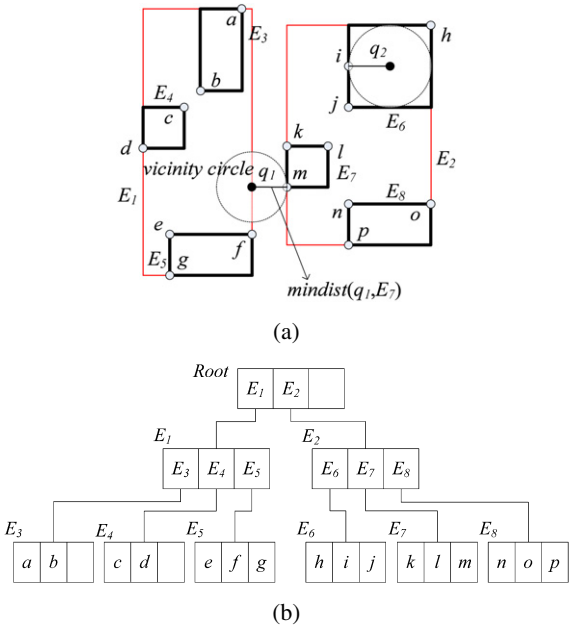

Fig. 1. An example of R-tree and NN query processing for q_1 and q_2. (a) Points and node extents, (b) the corresponding R-tree.

Every node has between m and M entries ($m \leqslant M/2$) unless it is the root node. Fig. 1 presents an R-tree for point set $P = \{a, b, \ldots, p\}$ in which the maximum node capacity M is 3. Points that are spatially close in space (e.g., k, l, and m) are clustered in the same leaf node (E_7). Nodes are then recursively grouped together using this same principle, except for the top level, which consists of a single root.

Roussopoulos et al. [5] proposed a branch-and-bound algorithm for NN search in a depth-first manner. This algorithm is referred to as the depth-first search (DFS). Three pruning heuristics are suggested, based on two distance metrics (*mindist* and *minmaxdist*) to discard the candidate nodes, so that the number of disk accesses can be minimized. *mindist* is the minimum possible distance between the query point and a node (or MBR R), while *minmaxdist* is the minimum of maximum possible distances from the query point to a face of the MBR. Conceptually speaking, *mindist* and *minmaxdist* provide a lower- and an upper-bound on the actual distance of object O from query point q, respectively. Based on these metrics, Roussopoulos et al. [5] proposed three pruning heuristics: PH1, PH2, and PH3. In [1], Cheung and Fu have observed that only the PH3 is sufficient to maintain the same number of pruned nodes. This implies that *minmaxdist* is not necessary for pruning. The DFS algorithm visits nodes with minimum *mindist* order from the *Root*. In the example of query point q_1 in Fig. 1, after visiting the root node, the minimum *mindist* node (e.g., E_1) from q_1 is visited. The process is repeated recursively until the leaf node E_5 contains the potential NN f. When backtracking to the previous level (node E_1), remaining entries (E_3 and E_4) are easily pruned by PH3. The actual NN m can be found, by backtracking again to the root level and following the search path E_2E_7. In summary, the order of nodes visited in the DFS algorithm for query point q_1 is *Root*, E_1, E_5, E_2, E_7. The DFS algorithm is proven to be sub-optimal [4]. This means that it visits more nodes than actually necessary.

Given a query point q, let $VC(q)$ be the vicinity circle of query point q that centers at the query point q and has radius equal to $mindist(q, NN(q))$. As proven in [4], an optimal NN search algorithm should only visit the nodes intersecting with the vicinity circle VC. In [3], Hjaltason and Samet proposed the best-first search (BFS) algorithm, which achieves optimal I/O performance. The algorithm maintains a priority queue $\mathcal{H}$ of the entries visited so far in ascending order by *mindist*. Similar to DFS, BFS begins from the root and insert all entries in the node into $\mathcal{H}$. In the example of Fig. 1, it inserts all entries (e.g., E_1 and E_2) from the *Root* node into $\mathcal{H}$. The priority queue is then $\mathcal{H} = \{\langle E_2, mindist(q_1, E_2)\rangle, \langle E_1, mindist(q_1, E_1)\rangle\}$. At each step, the first item in the priority queue is selected for visiting, and all its entries are inserted into $\mathcal{H}$. The algorithm follows the same procedure until a data object is visited. Therefore, the order of nodes visited in the BFS algorithm for query point q_1 is *Root*, E_1, E_2, E_7 (without visiting E_5). The BFS algorithm proposed by Hjaltason and Samet is known as the optimal NN search algorithm for R-trees.

3. Bottom-up search (BUS) algorithm

The performance limitation of conventional top–down algorithms is the height of the tree resulting from the characteristics of the algorithm. This limit should be overcome, in order to support high performance database applications. This increasing demand leads to the creation of a new search algorithm.

3.1. Hash structure

In order to make the bottom-up approach possible, the conventional R-trees are required to be modified slightly. In this paper, an in-memory hash data structure to support the proposed bottom-up search algorithm is presented. Definitions 1 and 2, which follow, describe the basic concept of the hash and cell. For the simplicity of presentation, it is assumed that $d = 2$ in the rest of