Lecture Notes: Best-first Algorithm

Yufei Tao
Chinese University of Hong Kong
taoyf@cse.cuhk.edu.hk

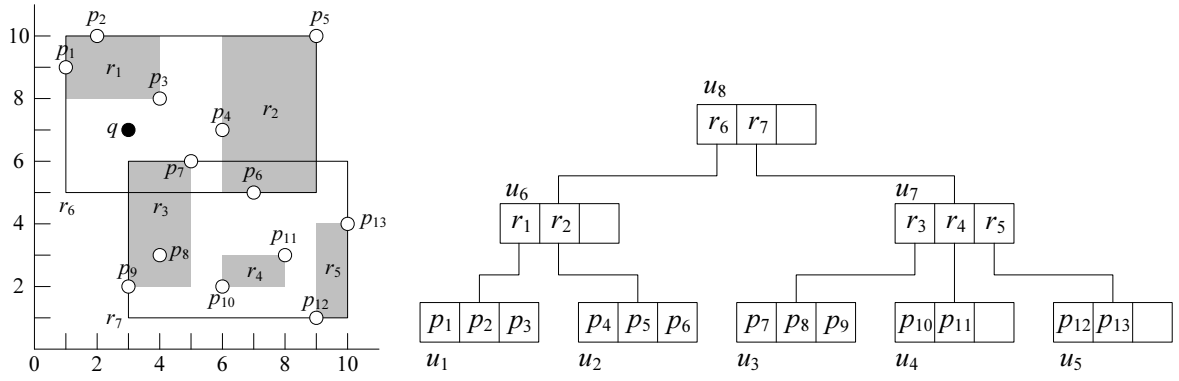
13 May, 2012

The R-tree is one of the (very) few multi-dimensional indexes that have been incorporated in a commercial database system (e.g., Oracle, DB2, Informix, etc.). Therefore, it is an important research topic to design efficient algorithms for R-trees that can be easily implemented. Such algorithms enhance the power of a commercial system, thus having immediate impacts in practice.

In this lecture, we will study a well-known method called *best-first (BF) search*, which has been applied extensively to solve a large number of problems with R-trees. A crucial feature of the method is that it can be used to design *optimal* algorithms (we will clarify the notion of optimality later). We will discuss it in the context of nearest neighbor (NN) search.

1 Best-first algorithm

Let P be a set of 2d points. The NN problem is to find the point $p \in P$ that is the closest to a query point q. For example, assume that P consists of p_1, ..., p_{13}, as in Figure 1a. Then, the result of the NN query q is p_3. Next, we will explain how a BF algorithm [1] uses the R-tree in Figure 1b to answer the query.



(a) Data and MBRs (b) The structure

Figure 1: An R-tree

BF uses a memory-resident heap $\mathcal{H}$ to manage all the (non-leaf or leaf) elements in an R-tree that have been accessed. This heap makes sure that the most *promising* element is always deheaped first, so that it can be processed as early as possible. In NN search, an element is promising if its MBR is close to q (the MBR of a leaf element is the point it represents). Formally, given a rectangle r and a point p, we define $mindist(p, r)$ as the minimum distance (MINDIST) from p to any point (on the boundary or in the interior) of r. See Figure 2. Clearly, $mindist(p, r)$ is a *lower bound* of the distance from p to any data point that lies in r.

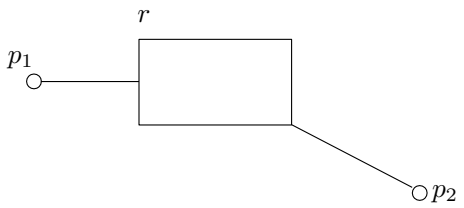


Figure 2: Illustration of *mindist*; $mindist(p_1, r)$ and $mindist(p_2, r)$ equal the lengths of the segments connecting p_1 and p_2 to r, respectively.

BF accesses the elements of an R-tree in ascending order of their MINDIST to q. For this purpose, it uses a min-heap $\mathcal{H}$ to organize the MINDIST of the elements that have been accessed. At each step, it deheaps an element from $\mathcal{H}$, and processes it. In case it is a non-leaf element, its child node is visited such that all of its elements are inserted in $\mathcal{H}$. This continues until the next element deheaped from $\mathcal{H}$ is a data point, which is guaranteed to be the NN of q.

Next, we illustrate the algorithm using Figure 1. The first node visited is the root. Its elements are inserted in $\mathcal{H}$, after which its content is $\{(r_6, 0), (r_7, 1)\}$[1] where each pair is in the form of $(r, mindist(q, r))$. Then, BF removes r_6 from $\mathcal{H}$, accesses the child u_6 referenced by r_6, and inserts the elements of u_6 in $\mathcal{H}$. At a result, $\mathcal{H}$ becomes $\{(r_7, 1), (r_1, 1), (r_2, 3)\}$. Note that the first two pairs of $\mathcal{H}$ have the same MINDIST; hence, their ordering is random. Similarly, next BF accesses u_7, after which $\mathcal{H}$ changes to $\{(r_1, 1), (r_3, 1), (r_2, 3), (r_4, 5), (r_5, \sqrt{45})\}$. In the same fashion, the next two nodes visited are r_1 and r_3, respectively. At this point, $\mathcal{H}$ is $\{(p_3, \sqrt{2}), (p_7, \sqrt{5}), (p_1, \sqrt{8}), (r_2, 3), (p_2, \sqrt{10}), (p_8, \sqrt{17}), (r_4, 5), (p_9, 5), (r_5, \sqrt{45})\}$. Since the next element deheaped is a data point p_3, BF terminates by returning it as the NN.

2 Optimality

Given an R-tree $\mathcal{T}$ (such as the one in Figure 1b), let $\mathcal{A}_\mathcal{T}$ be the class of all algorithms that uses only $\mathcal{T}$ to perform NN search, without any other knowledge about the underlying dataset P. In other words, any information that an algorithm $A \in \mathcal{A}_\mathcal{T}$ has about P must come from the nodes of the R-tree already accessed by A. Define $cost(A, q)$ to be the number of nodes of $\mathcal{T}$ that A accesses in order to answer a NN query q. Then, an algorithm $A^* \in \mathcal{A}_\mathcal{T}$ is *optimal on* q if

$$cost(A^*, q) \leq cost(A, q) \text{ for any } A \in \mathcal{A}_\mathcal{T}.$$

We consider a *general situation* where

1. q has a distinct MINDIST to every MBR in $\mathcal{T}$, except those MBRs covering q, and
2. q does not coincide with its NN.

Theorem 1. *BF is optimal on any query q in the general situation.*

Proof. Let p^* be the NN of q. Let O be a circle that centers at q and crosses p^*. First, notice that any algorithm $A \in \mathcal{A}_\mathcal{T}$ has the following property: if the MBR r of a node u intersects O, then A must definitely access u. This is because since all the knowledge of A about the dataset P comes

[1]For convenience, we demonstrate the content of $\mathcal{H}$ with a sorted list, but the sorted order does not need to be maintained in practice. BF only requires that the smallest element in $\mathcal{H}$ be obtained efficiently.

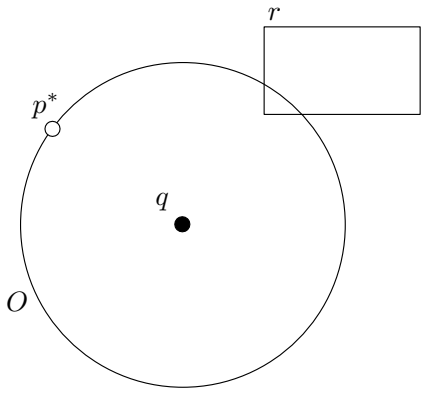


Figure 3: Search region: p is the NN of q; then any algorithm in $\mathcal{A}_{\mathcal{T}}$ must access a node whose MBR r intersects circle O.

from the nodes that A has accessed, without accessing u A will not be able to tell whether u has a point closer to q than p^*.

Hence, to establish the optimality of BF, it suffices to prove that BF accesses *only* those nodes whose MBRs intersect O. Note that, in the general situation, O has a positive radius (Requirement 1); furthermore, since p itself is a degenerated MBR in $\mathcal{T}$, no MBR can have MINDIST $\|p, q\|$ to q (Requirement 2). The optimality of BF thus follows directly from the fact that it accesses the MBRs of $\mathcal{T}$ in ascending order of their MINDIST. □

3 Remarks

BF can be easily extended to perform k nearest neighbor search. Specifically, instead of terminating immediately upon deheaping the *first* data point from $\mathcal{H}$, it terminates after deheaping the k-th.

A drawback of BF is that the size of $\mathcal{H}$ is unbounded. However, this is not a serious problem in practice when k and the dimensionality are low, because in this case $\mathcal{H}$ is usually by far smaller than the memory of a modern computer. Another famous NN algorithm on R-trees can be found in [2]. This algorithm requires much less memory than BF, but incurs higher query cost if memory is not a concern.

References

[1] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318, 1999.

[2] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 71–79, 1995.