Nearest Neighbor Search

Yufei Tao

ITEE
University of Queensland

In this lecture, we will study a new problem called nearest neighbor search, which plays an important role in a great variety of applications. Our discussion will also introduce two methods: the branch-and-bound and the best first techniques, both of which are generic algorithmic paradigms useful in many scenarios.

Nearest Neighbor Search

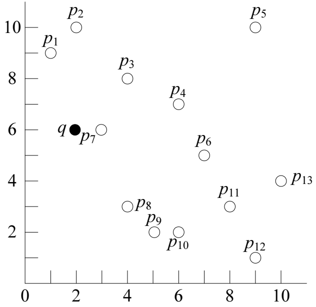
Let P be a set of d-dimensional points in $\mathbb{R}^d$. The (Euclidean) nearest neighbor (NN) of a query point $q \in \mathbb{R}^d$ is the point $p \in P$ that has the smallest Euclidean distance to q.

Given a query point q, an NN query returns the NN(s) of q. Note that multiple points can have the smallest distance to q, in which case they are all nearest neighbors and should be reported.

Note:

- The Euclidean distance between p and q is the length of the line segment connecting p and q.
- We denote the Euclidean distance between p and q as $\|p, q\|$.

Example

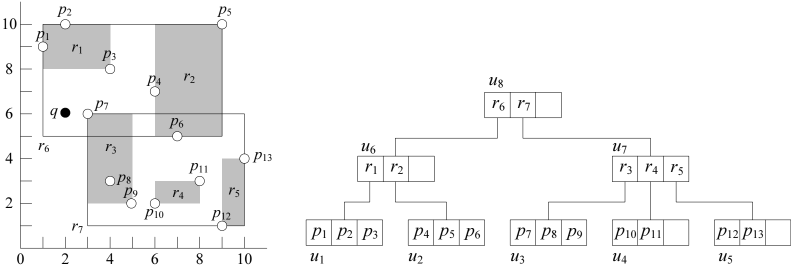


The NN of q is p_7.

Applications

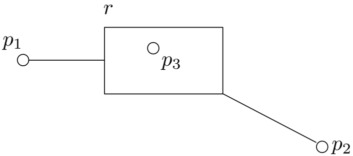
- “Find the McDonald that is nearest to me”.
- “Find the customer profile in the database that is most similar to the profile of the new customer”.
- “Retrieve the image from the database that is most similar to the one given by the user”.
- ...

If no pre-processing is allowed on P, we must scan the entire P to answer a NN query. Query efficiency can be significantly improved by using an R-tree on P.



Mindist

Given a point q and an axis-parallel rectangle r, the *mindist* of q and r, denoted as *mindist*(q, r), equals $\min_{p \in r} \|q, p\|$.



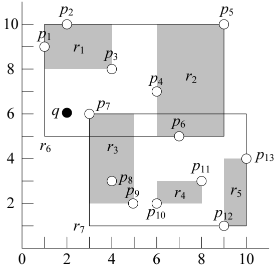
In the above example, with respect to r, the mindists of p_1 and p_2 are equal to the lengths of the two segments shown, while that of p_3 is 0.

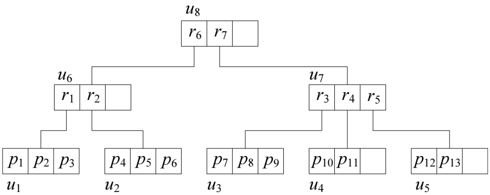
Think: how to compute *mindist*(q, r) in $O(d)$ time?

Algorithm 1: Branch-and-bound (BaB)

BaB performs a depth-first traversal of the R-tree but uses mindists to (i) prioritize the nodes for accessing, and (ii) prune the nodes that cannot contain the final answer.

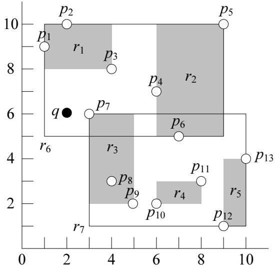
Let us illustrate the algorithm from an example. To find the NN of q (as shown in the figure), BaB starts from the root of the R-tree, where it sees two MBRs r_6 and r_7. The mindists from q to r_6 and r_7 are 0 and 1, respectively. Since $mindist(q, r_6)$ is smaller, algorithm visits u_6 next.

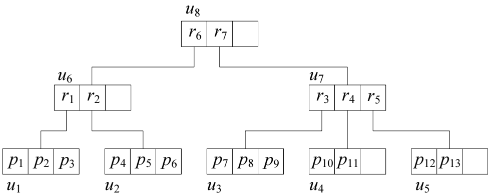




Branch-and-bound (BaB)

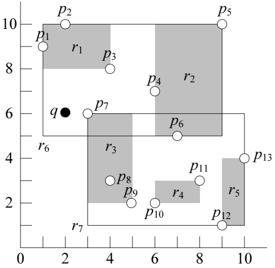
At node u_6, BaB chooses to descend into MBR r_1, because its mindist from q is smaller than that of r_2.

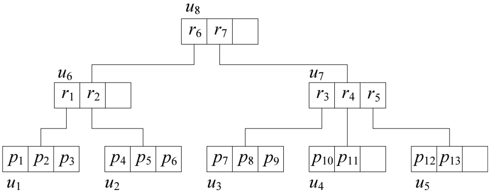




Branch-and-bound (BaB)

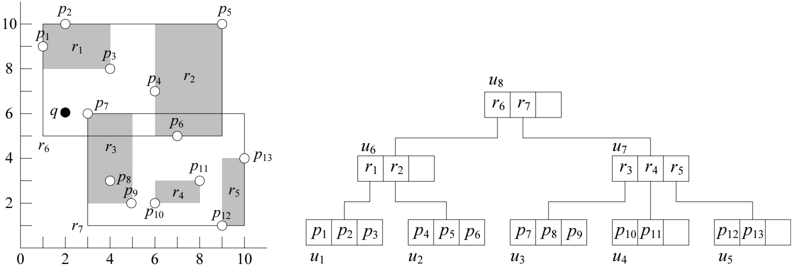
Now the algorithm is at the leaf node u_1. It simply computes the distance from q to each data point in u_1, and remembers the nearest one, i.e., p_3. This is the current NN of q found so far.





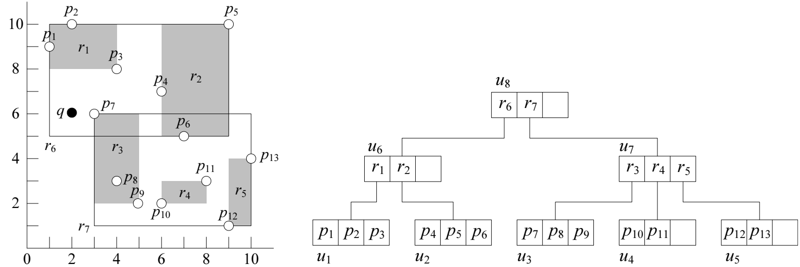
Branch-and-bound (BaB)

Now the algorithm backtracks to node u_6, where the subtree of MBR r_2 has not been explored yet. However, the fact that the $mindist(q, r_2) = 4$ is greater than the distance $2\sqrt{2}$ from q to the current NN p_3 rules out the possibility that the NN of q can be inside r_2. Therefore, the subtree of r_2 can be pruned.



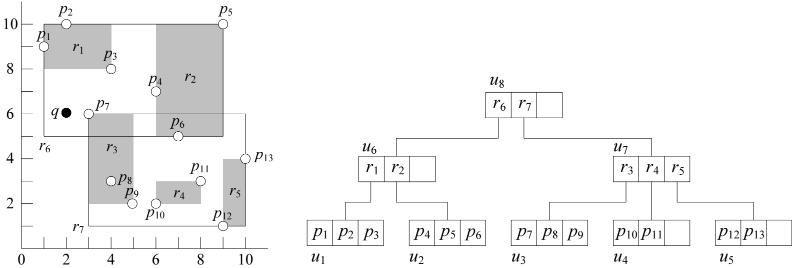
Branch-and-bound (BaB)

Now we backtrack to the root, where MBR r_7 has not been processed yet. The mindist 1 between q and r_7 is smaller than $\|q, p_3\| = 2\sqrt{2}$. Therefore, the child u_7 of r_7 must be visited.



Branch-and-bound (BaB)

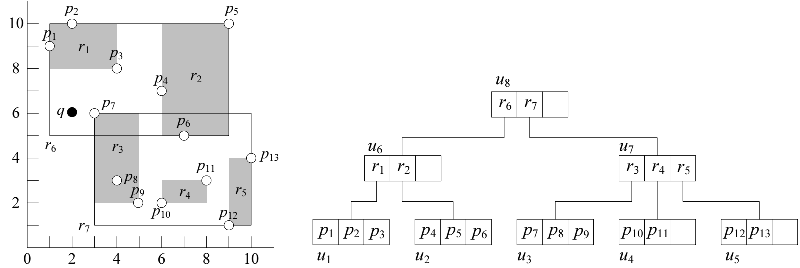
At node u_7, the algorithm accesses the child node u_3 of MBR r_3 which has the smallest mindist to q among r_3, r_4, r_5.



Branch-and-bound (BaB)

At node u_3, BaB finds p_7 which replaces p_3 as its current NN.

Then, it backtracks to node u_7 and prunes r_4 and r_5. After that, the algorithm backtracks one more level to the root. As all the MBRs of the root have been processed, it terminates with p_7 as the final result.



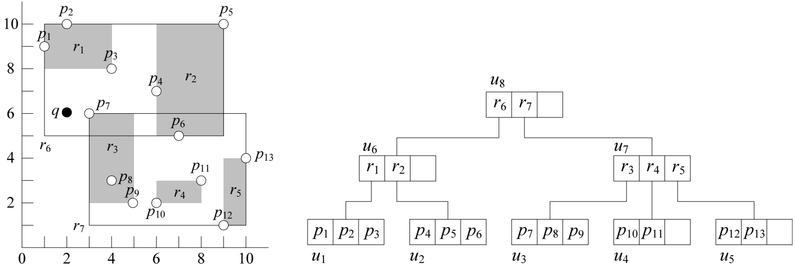
Pseudocode of BaB

```
algorithm BaB($u, q$)
/* $u$ is the node being accessed, $q$ is the query point;
   $p_{best}$ is a global variable that keeps the NN found so far;
   the algorithm should be invoked by setting $u$ to the root */
if $u$ is a leaf node then
    if the NN of $q$ in $u$ is closer to $q$ than $p_{best}$ then
        $p_{best}$ = the NN of $q$ in $u$
else
    sort the MBRs in $u$ in ascending order of their mindists to $q$
    /* let $r_1, ..., r_f$ be the sorted order */
    for $i = 1$ to $f$
        if $mindist(q, r_i) < \|q, p_{best}\|$ then
            Bab($u_i, q$)
            /* $u_i$ is child node of $r_i$ */
```

Note: the above description assumes that q has only one NN. It is easy to extend it to the scenario where multiple points have the smallest distance to q (think: how?)

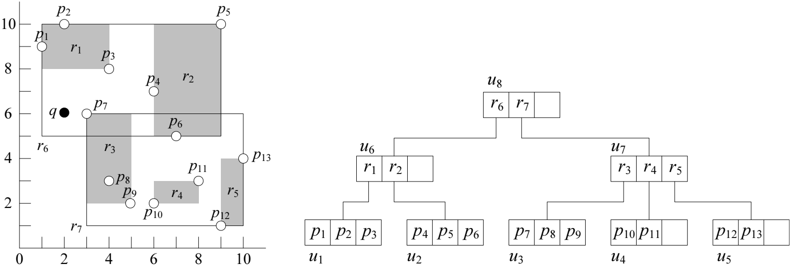
Algorithm 2: Best First (BF)

We have seen that BaB accessed u_8, u_6, u_1, u_7, u_3. Next, we will learn a better algorithm called best first (BF) that can avoid accessing u_1.



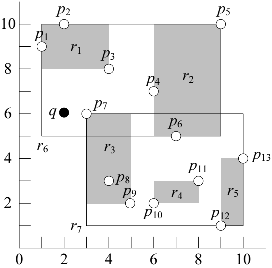
Algorithm 2: Best First (BF)

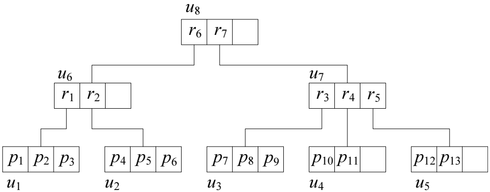
Again, we illustrate the BF algorithm with an example. As with BaB, BF also starts from the root. At any moment, the algorithm keeps in memory all the intermediate MBRs that have been seen but not yet accessed in a sorted list H, using their mindists to q as the sorting keys. In our example, so far we have seen only two MBRs r_6, r_7, so H has two entries $\{(r_6, 0), (r_7, 1)\}$.



Best First (BF)

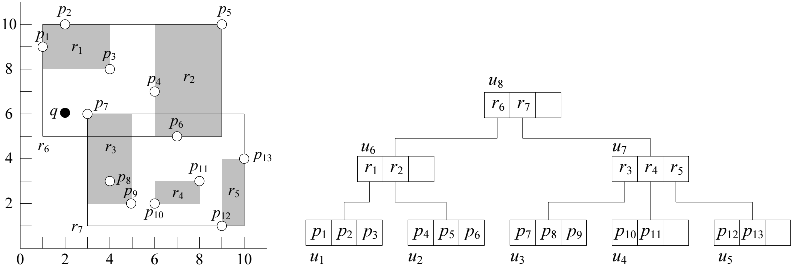
Each iteration of BF removes from H the MBR with the smallest mindist, and accesses its child node. Continuing the example, BF removes r_6 from H, visits its child node u_6, and adds to H the MBRs r_1, r_2 there. At this time, $H = \{(r_7, 1), (r_1, 2), (r_2, 4)\}$.





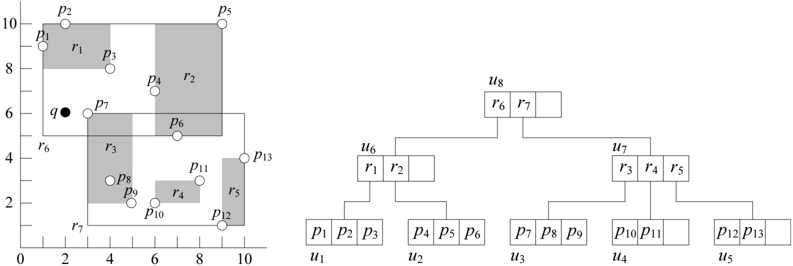
Best First (BF)

Similarly, as r_7 has the smallest key in H, BF accesses its child node u_7, after which $H = \{(r_3, 1), (r_1, 2), (r_2, 4), (r_4, 5), (r_5, \sqrt{53})\}$.



Best First (BF)

Next, the algorithm visits leaf node u_3, where p_7 is taken as the current NN. Then, BF terminates because $\|q, p_7\| = 1$ is smaller than the lowest mindist of the MBRs in $H = \{(r_1, 2), (r_2, 4), (r_4, 5), (r_5, \sqrt{53})\}$, implying that p_7 must be the final NN.



Pseudocode of BF

```
algorithm BF($q$)
/* in the following $H$ is a sorted list where each entry is an MBR
   whose sorting key in $H$ is its mindist to $q$;
   $p_{best}$ is a global variable that keeps the NN found so far. */
insert the MBR of the root in $H$
while $\|q, p_{best}\|$ is greater than the smallest mindist in $H$
/* if $p_{best} = \emptyset$, $\|q, p_{best}\| = \infty$ */
    remove from $H$ the MBR $r$ with the smallest mindist
    access the child node $u$ of $r$
    if $u$ is an intermediate node then
        insert all the MBRs in $u$ into $H$
    else
        if the NN of $q$ in $u$ is closer to $q$ than $p_{best}$ then
            $p_{best}$ = the NN of $q$ in $u$
```

Note: the above description assumes that q has only one NN. It is easy to extend it to the scenario where multiple points have the smallest distance to q (think: how?)

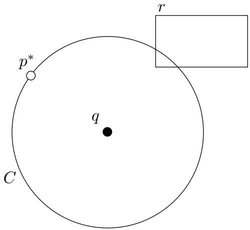
Think: what data structure would you use to manage H?

We have seen from the above examples that BF accesses less nodes than BaB. It is natural to wonder: can BF be further improved? The answer turns out to be no. As will proved next, BF is optimal, i.e., it is guaranteed to access the least number of nodes among all the algorithms that use the same R-tree to solve a given NN query.

Optimality of BF

Denote by C the circle that centers at q, and has radius $\|p^*, q\|$, where p^* is an arbitrary NN of q. Let S^* be all the nodes whose MBRs intersect C.

It is important to observe that all algorithms must access all the nodes in S^*. Assume, for example, that the node with MBR r in the figure below was not accessed. How could the algorithm assert that no point in r is closer to q than p^*?



Optimality of BF

It suffices to prove that BF accesses only those nodes whose MBRs intersect C. This can be shown in two steps:

1. BF accesses MBRs in non-descending order of their mindists to q.
    - Let r_1 and r_2 be two MBRs accessed consecutively. r_2 either already existed in H when r_1 was visited, or r_2 is an MBR inside r_1. In either case, it must hold that $mindist(q, r_2) \geq mindist(q, r_1)$.
2. Let r be the MBR of a leaf node containing an arbitrary NN of q. Let r' be an MBR that does not intersect C. By the first bullet, r is visited before r'. However, when r is found, BF must necessarily discover p^*, whose presence prevents the algorithm from accessing r' (Line 2 in Slide 21).

So far we have assumed that, if multiple data points have the smallest mindist to q, all of them must be reported.

There is an alternative version of NN search where it suffices to report one arbitrary NN in the aforementioned scenario. The BF algorithm (executed precisely as described in Slide 21) is not optimal in such a case. Can you construct a counter-example?

Extensions

BF can be adapted to solve more complicated forms of nearest neighbor search:

- Other distance metrics: So far we have assumed that the distance between two points are computed by Euclidean distance, which is known as the L_2 norm. In general, the distance between two points p and q under L_t norm—where t is an arbitrary positive value—is calculated as:

  $$\left(\sum_{i=1}^{d}\Big|p[i]-q[i]\Big|^{t}\right)^{1/t}.$$

  The NN problem extends in a straightforward manner to these distance metrics (and many others).
- k nearest neighbor search: Given a query point q, return the data points with the smallest, 2nd smallest, ..., k-th smallest distances to q.
- Distance browsing: This operation outputs the points of the dataset P in ascending order of their distances to q.