

Approximate Nearest Neighbor Searching in Fixed Dimensions

2/7/2000

Based on results by Arya et al. [AMN⁺98].

1 Introduction

Let P be a set of points \mathbb{R}^d , where d is a constant denoting the dimension. For a query point p , the point $q \in P$ is the nearest neighbor of p in P , if $\text{dist}(p, q) \leq \text{dist}(p, q'), \forall q' \in P$.

Given a point set P and constructing a data-structure for answering such queries can be achieved by constructing a Voronoi diagram of P , and performing a point-location queries in the diagram. For $d > 2$, one construct such diagram in $O(n^{\text{ceild}/2+\delta})$ time, and answer such queries in $O(\log n)$ time. One can achieve a slightly sublinear query time, using linear space [Aga97]. All the constants depend exponentially in d , and are thus of extremely restricted usefulness in practice (except for $d = 2$).

Given $\varepsilon > 0$, we say that a point $p \in S$ is a $(1 + \varepsilon)$ -approximate nearest neighbor of q if

$$\text{dist}(p, q) \leq (1 + \varepsilon)\text{dist}(p^*, q),$$

where p^* is the true nearest neighbor to q .

We will show the following result:

Theorem 1.1 *consider a set S of n data points in \mathbb{R}^d . There is a constant $c_{d,\varepsilon} \leq d \lceil 1 + 6d/\varepsilon \rceil^d$, such that in $O(dn \log n)$ time it is possible to construct a data structure of size $O(dn)$, such that for any metric:*

1. *Given any $\varepsilon > 0$ and $q \in \mathbb{R}^d$, a $(1 + \varepsilon)$ -approximate nearest neighbor of q in S can be reported in $O(c_{d,\varepsilon} \log n)$ time.*
2. *More generally, given $\varepsilon > 0, q \in \mathbb{R}^d$, and any $k, 1 \leq k \leq n$, a sequence of k $(1 + \varepsilon)$ -approximate nearest neighbors of q in S can be computed in $O((c_{d,\varepsilon} + kd) \log n)$ time.*

2 The BBD-Tree

Definition 2.1 A *rectangle* in \mathbb{R}^d is a d -fold product of closed intervals on the coordinate axes. For $1 \leq i \leq d$, the i -th length of a rectangle is the length of the i -interval. The *size* of a rectangle is the length of its longest side. We define a *box* to be a rectangle whose aspect ratio (the ratio between the longest and shortest sides) is bounded by some constant, which for concreteness we will assume to be 3.

A is either a box or the set theoretic differences of two boxes, one enclosed within the other (i.e., the “annulus” between two boxes). Thus, each cell is defined by an *outer box* and an optimal *inner box*. Each cell will be associated with the set of data points lying within the cell. Cells are considered to be closed and hence points which lie on the boundary between cells may be assigned to either cell. The size of a cell is the size of its outer box.

Definition 2.2 For two intervals $[x_I, y_I] \subseteq [x_O, y_O]$ and $w = y_I - x_I$. We say that $[x_I, y_I]$ is *sticky* for $[x_O, y_O]$ if each of the two distances between the inner interval and the other interval, are either 0 or at least w . The inner box b_I is *sticky* for the outer box b_O if each of the d intervals of b_I is sticky for the corresponding interval of b_O .

In the construction of the BBD-Tree we maintain the condition that the inner box is always sticky for the outer box.

BBD-Tree is constructed recursively by alternatively applying splits by planes, or alternatively, by shrinking (i.e., creating inner and outer node). The following condition will be maintained:

- All boxes satisfy the aspect ratio bound.
- If the parent has an inner box, then this box lies entirely within one of the two children., If the operation is a shrink, then this inner box lies within the inner child of the shrink.
- Inner boxes are sticky for their enclosing outer boxes.

In partitioning the points during the recursive construction, we will maintain d -connected (sorted) lists of the points according to the d -coordinates. This enables us to perform split of the point-set according to the i -th coordinate in time which is linear in the size of the *smaller* set. Thus, we can in *linear time* compute the node in the quadtree that contains at most $2n/3$ of the points (note that regular split is easier). To do so, we simulate quad-tree construction, where in each dimension we skip levels if needed (using “suspicious” operations). There are two problems:

- How to perform the above shrinking process so that the splitting time is indeed linear...
- The resulting shrinking box, does not necessarily contain the inner bounding box of the parent.

The first problem is solved by careful implementation. The second problem is solved by constructing a constant depth tree of shrinking and splits.

In particular, we will show that the depth of the tree is $O(\log n)$, which would imply that the overall constructing time is $O(dn \log n)$.

Since we alternate splits with shrinks, we have that the depth of the tree is $O(\log n)$, as the number of elements in a node is reduced by a constant factor as we traverse a constant number of levels.

2.1 Further Modifications

A split is trivial if one of its children contain no data-point. Observe that a sequence of trivial splits in the tree can always be compressed into a single split. Thus, we can assume that there are no consecutive shrinks in the tree.

We want to associate with each leaf a data-point. To do so, each parent, borrow two points for its children. If it is a trivial shrink, it uses one of those points for the empty children, and passes the other one upward (we need to be careful here - if there is an empty shrink, one of its children must not be empty, and it contains at *least* two points - otherwise we had stopped the constructed earlier.) Since there are not trivial consecutive shrinks (or splits), we always succeeded in this borrowing scheme. Each point, is associated with at most two leafs.

Theorem 2.3 *Given a set of n data points S in \mathbb{R}^d , in $O(dn \log n)$ time we can construct a binary tree having $O(n)$ nodes representing a hierarchical decomposition of \mathbb{R}^d into cells (satisfying the stickiness properties given earlier) such that*

1. *The height of the tree is $O(\log n)$ and in general, with every 4 levels of descent in the tree, the number of points associated with the nodes decreases by at least a factor of $2/3$.*
2. *The cells have bounded aspect ration, and with every $4d$ levels of descent in the tree, the sizes of the associated cells decrease by at least a factor of $2/3$.*
3. *Each leaf cell is associated with one data point, which is either contained within the cell, or contained within the inner box of the cell. No data point is associated with more than two leaf cells.*

References

- [Aga97] P.K. Agarwal. Range searching. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 31, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.
- [AMN⁺98] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6), 1998.