

Approximate Nearest Neighbor Searching in Fixed Dimensions

2/9/2000

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

1 Essential Properties

The leafs of the BBD tree define a subdivision of space, with the following properties.

1. Each cell contains a constant number of points.
2. If a cell contains one or more data points, then those points are associated with the cell. Otherwise the point is associated with the outer box of the cell (i.e., the point is inside the outer bounding box, although it might lie inside the inner box). To see that, notice that the inner box when created always contains points which are being borrowed by the process described in the previous lecture [notes at least]).
3. Point location in $O(d \log n)$ time.
4. The number of cells of “large” cells of size s intersecting a ball of radius r is bounded by a function of s, r, d .
5. cells can be enumerated by their distance from a query point q in $O(md \log n)$ time (where m is the number of cells).

Observe that we can compute the distance of a point to a cell in $O(d)$ time (how?). Thus, by the depth property of the tree we insert at most $O(\log n)$ elements to

This is done by using a heap to maintain the cells visited so far (with their length). In each stage, we extract the closest cell in the heap, and descent to the leaf closest to our query point - entering all the cells adjacent to our path to the heap. using Fibonacci heaps (see [CLR90]) we do insertion in $O(1)$ (amortized) and delete/get min in $O(\log n)$. Thus the claimed bound follows.

Lemma 1.1 (Lemma 4) *Given a BBD-Tree for a set of data points in \mathbb{R}^d , the number of leaf cells of size at least $s > 0$ that intersect a ball of radius r is at most $\lceil 1 + 6r/s \rceil^d$.*

Proof: By aspect ratio, the smallest side of the boxes involved is $\geq s/3$. Now use space packing or grid argument as in the WSPD paper. One has to be careful because the cells might be the difference between outer and inner boxes. However, by stickyness, the volume of the difference is still large enough to contain a grid point. ■

2 Approximate nearest neighbor queries

We first perform a point location to our query point q . This leaf of the tree contains an associated point p . We take p to be our “candidate” to be the nearest point. For this point, we use the “proximity” heap to traverse the leafs of the subdivision in an increasing order from our query point q . Each such leaf of the subdivision is associated with query points (a constant number of those!). Thus, we can maintain in constant time the closes point p seems so far (i.e., p changes as the algorithm progresses). The algorithm stops as soon as the distance to the current cell being handled is $\geq \text{dist}(p, q)/(1 + \varepsilon)$. Clearly, the algorithm visited all the leaf cells that are covering the ball of radius $\text{dist}(p, q)/(1 + \varepsilon)$. Note also, that the algorithm did not visit any leaf cell which is too small (i.e., size $\leq \varepsilon \text{radius}$) as the associated point with this cell would be closer to q then p , which is not possible. The following, is the formal proof of this intuition.

Lemma 2.1 (Lemma 5) *The number of leaf cells visited by the nearest neighbor algorithm is at most $C_{d,\varepsilon} \leq \lceil 1 + 6d/\varepsilon \rceil^d$*

Proof: Look on the leaf visited by the algorithm before it stopped (namely, the algorithm visited just another leaf where it stopped). Let p be the nearest point found so far, and let r be the distance between the query point q and this leaf cell. Since, the algorithm did not stop, we know that

$$r(1 + \varepsilon) \leq \text{dist}(p, q).$$

We claim, that no cell leaf visited so far is of size smaller than $r\varepsilon/d$. Indeed, if there was such a cell visited, then it contains inside its outer box a point p' . The distance of this point for q is the distance to the cell to q , plus the diameter of the cell. Namely,

$$\text{dist}(q, p') \leq r + \sqrt{dr}\varepsilon/d \leq r(1 + \varepsilon) \leq \text{dist}(p, q).$$

Namely, the algorithm had encountered a point which is closer to q than p . A contradiction to the definition of p .

Namely, all the leafs encountered by the algorithm are large (i.e., their size larger than $r\varepsilon/d$), and they intersect the ball of radius r centered around q . since those cells are disjoint, the result follows. ■

The algorithm can be extended in a straightforward way to compute the k approximate nearest neighbor in $O((k + 1/\varepsilon^d) \log n)$ time.

Remark 2.2 Can improve the analysis of the algorithm, or give an input example that requires the time stated in the theorem?

References

- [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.
- [CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1990.