Geometric Approximation Algorithms

Sariel Har-Peled[1]

August 23, 2006

[2]Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; sariel@uiuc.edu; http://www.uiuc.edu/~sariel/. Work on this paper was partially supported by a NSF CAREER award CCR-0132901.

Contents

Chapter 1

The Power of Grids - Computing the Minimum Disk Containing k Points

The Peace of Olivia. How sweat and peaceful it sounds! There the great powers noticed for the first time that the land of the Poles lends itself admirably to partition.
– The tin drum, Gunter Grass

In this chapter, we are going to discuss two basic geometric algorithms. The first one, computes the closest pair among a set of n points in linear time. This is a beautiful and surprising result that exposes the computational power of using grids for geometric computation. Next, we discuss a simple algorithm for approximating the smallest enclosing ball that contains k points of the input. This at first looks like a bizarre problem, but turns out to be a key ingridiant to our later discussion.

1.1 Preliminaries

For r a real positive number and a point $p = (x, y)$ in $\mathbb{R}^2$, define $\mathrm{G}_r(p)$ to be the point $(\lfloor x/r \rfloor r, \lfloor y/r \rfloor r)$. We call r the *width* of the *grid* G_r. Observe that G_r partitions the plane into square regions, which we call grid *cells*. Formally, for any $i, j \in \mathbb{Z}$, the intersection of the half-planes $x \geq ri$, $x < r(i+1)$, $y \geq rj$ and $y < r(j+1)$ is said to be a grid cell. Further we define a *grid cluster* as a block of 3×3 contiguous grid cells.

Note, that every grid cell C of G_r, has a unique ID; indeed, let $p = (x, y)$ be any point in C, and consider the pair of integer numbers $\mathrm{id}_C = \mathrm{id}(p) = (\lfloor x/r \rfloor, \lfloor y/r \rfloor)$. Clearly, only points inside C are going to be mapped to id_C. This is very useful, since we store a set P of points inside a grid efficiently. Indeed, given a point p, compute its $\mathrm{id}(p)$. We associate with each unique id a data-structure that stores all the points falling into this grid cell (of course, we do not maintain such data-structures for grid cells which are empty). So, once we computed $\mathrm{id}(p)$, we fetch the data structure for this cell, by using hashing. Namely, we store pointers to all those data-structures in a hash table, where each such data-structure is indexed by its unique id. Since the ids are integer numbers, we can do the hashing in constant time.

Assumption 1.1.1 *Through out the discourse, we are going to assume that every hashing operation takes (worst case) constant time. This is quite a reasonable assumption when true randomness is available (using for example perfect hashing [CLRS01]).*

For a point set P, and parameter r, the partition of P into subsets by the grid G_r, is denoted by $\mathrm{G}_r(P)$. More formally, two points $p, q \in P$ belong to the same set in the partition $\mathrm{G}_r(P)$, if both points are being mapped to the same grid point or equivalently belong to the same grid cell.

1.2 Closest Pair

We are interested in solving the following problem:

Problem 1.2.1 Given a set P of n points in the plane, find the pair of points closest to each other. Formally, return the pair of points realizing $\mathcal{CP}(P) = \min_{p,q \in P} \|pq\|$.

Lemma 1.2.2 *Given a set P of n points in the plane, and a distance r, one can verify in linear time, whether or not $\mathcal{CP}(P) < r$ or $\mathcal{CP}(P) \geq r$.*

Proof: Indeed, store the points of P in the grid G_r. For every non-empty grid cell, we maintain a linked list of the points inside it. Thus, adding a new point p takes constant time. Indeed, compute $\mathrm{id}(p)$, check if $\mathrm{id}(p)$ already appears in the hash table, if not, create a new linked list for the cell with this ID number, and store p in it. If a data-structure already exist for $\mathrm{id}(p)$, just add p to it.

This takes $O(n)$ time. Now, if any grid cell in $\mathrm{G}_r(P)$ contains more than, say, 9 points of p, then it must be that the $\mathcal{CP}(P) < r$. Indeed, consider a cell C containing more than four points of P, and partition C into 3×3 equal squares. Clearly, one of those squares must contain two points of P, and let C' be this square. Clearly, the diameter of $C' = \mathrm{diam}(C)/3 = \sqrt{r^2 + r^2}/3 < r$. Thus, the (at least) two points of P in C' are distance smaller than r from each other.

Thus, when we insert a point p, we can fetch all the points of P that were already inserted, for the cell of P, and the 8 adjacent cells. All those cells, must contain at most 9 points of P (otherwise, we would already have stopped since the $\mathcal{CP}(\cdot)$ of inserted points, is smaller than r). Let S be the set of all those points, and observe that $|S| \leq 9 \cdot 9 = O(1)$. Thus, we can compute by brute force the closest point to p in S. This takes $O(1)$ time. If $\mathbf{d}(p, S) < r$, we stop, otherwise, we continue to the next point.

Overall, this takes $O(n)$ time. As for correctness, first observe that if $\mathcal{CP}(P) > r$ then the algorithm would never make a mistake, since it returns '$\mathcal{CP}(P) < r$' only after finding a pair of points of P with distance smaller than r. Thus, assume that p, q are the pair of points of P realizing the closest pair, and $\|pq\| = \mathcal{CP}(P) < r$. Clearly, when the later of them, say p, is being inserted, the set S would contain q, and as such the algorithm would stop and return '$\mathcal{CP}(P) < r$'. ■

Lemma 1.2.2 gives a natural way of computing $\mathcal{CP}(P)$. Indeed, permute the points of P in arbitrary fashion, and let $P = \langle p_1, \ldots, p_n \rangle$. Next, let $r_i = \mathcal{CP}(\{p_1, \ldots, p_i\}$. We can check if $r_{i+1} < r_i$, by just calling the algorithm for Lemma 1.2.2 on P_{i+1} and r_i. In fact, if $r_{i+1} < r_i$, the algorithm of Lemma 1.2.2, would give us back the distance r_{i+1} (with the other point realizing this distance).

In fact, consider the "good" case, where $r_{i+1} = r_i = r_{i-1}$. Namely, the length of the shortest pair does not check for awhile. In this case, we do not need to rebuild the data structure of Lemma 1.2.2, for each point. We can just reuse it from the previous iteration. Thus, inserting a single point takes constant time, as long as the closest pair does not change.

Things become bad, when $r_i < r_{i-1}$. Because then, we need to rebuild the grid, and reinsert all the points of $P_i = \langle p_1, \ldots, p_i \rangle$ into the new grid $\mathrm{G}_{r_i}(P_i)$. This takes $O(i)$ time.

So, if the closest pair radius, in the sequence $r_1, \ldots, r_n$ changes only k times, then the running time of our algorithm would be $O(nk)$. In fact, we can do even better.

Theorem 1.2.3 *Let P be a set of n points in the plane, one can compute the closest pair of points of P in expected linear time.*

Proof: Pick a random permutation of the points of P, let $\langle p_1, \ldots, p_n \rangle$ be this permutation. Let $r_2 = \|p_1 p_2\|$, and start inserting the points into the data structure of Lemma 1.2.2. In the ith iteration, if $r_i = r_{i-1}$, then this insertion takes constant time. If $r_i < r_{i-1}$, then we rebuild the grid and reinsert the points. Namely, we recompute $\mathrm{G}_{r_i}(P_i)$.

To analyze the running time of this algorithm, let X_i be the indicator variable which is 1 if $r_i \neq r_{i-1}$, and 0 otherwise. Clearly, the running time is proportional to

$$R = 1 + \sum_{i=2}^{n} (1 + X_i \cdot i) .$$

Thus, the expected running time is

$$\mathbf{E}[R] \quad = \quad 1 + \mathbf{E}\left[1 + \sum_{i=2}^{n} (1 + X_i \cdot i) \right] = n + \sum_{i=2}^{n} (\mathbf{E}[X_i] \cdot i) = n + \sum_{i=2}^{n} i \cdot \mathbf{Pr}[X_1 = 1] ,$$

by linearity of expectation and since for indicator variable X_i, we have $\mathbf{E}[X_i] = \mathbf{Pr}[X_i = 1]$.

Thus, we need to bound $\mathbf{Pr}[X_i = 1] = \mathbf{Pr}[r_i < r_{i-1}]$. To bound this quantity, fix the points of P_i, and randomly permute them. A point $q \in P_i$ is called *critical*, if $\mathcal{CP}(P_i \setminus \{q\}) > \mathcal{CP}(P_i)$. If there are no critical points, then $r_{i-1} = r_i$ and then $\mathbf{Pr}[X_i = 1] = 0$. If there is one critical point, than $\mathbf{Pr}[X_i = 1] = 1/i$, as this is the probability that this critical point, would be the last point in the random permutation of P_i.

If there are two critical points, and let p, q be this unique pair of points of P_i realizing $\mathcal{CP}(P_i)$. The quantity r_i is smaller than r_{i-1}, one if either p or q are p_i. But the probability for that is $2/i$ (i.e., the probability in a random permutation of i objects, that one of two marked objects would be the last element in the permutation).

Observe, that there can not be more than two critical points. Indeed, if p and q are two points that realizing the closest distance, than if there is a third critical point r, then $\mathcal{CP}(P_i \setminus \{r\}) = \|pq\|$, and r is not critical.

We conclude that

$$\mathbf{E}[R] = n + \sum_{i=2}^{n} i \cdot \mathbf{Pr}[X_1 = 1] \leq n + \sum_{i=2}^{n} i \cdot \frac{2}{i} \leq 3n.$$

We have that the expected running time is $O(\mathbf{E}[R]) = O(n)$. ■

Theorem 1.2.3 is a surprising result, since it implies that *uniqueness* (i.e., deciding if n real numbers are all distinct) can be solved in linear time. However, there is a lower bound of $\Omega(n \log n)$ on uniqueness, using the comparison tree model. This reality dysfunction, can be easily explained, once one realizes that the computation of Theorem 1.2.3 is considerably stronger, using hashing, randomization, and the floor function.

1.3 A Slow 2-Approximation Algorithm for the k-Enclosing Minimum Disk

For a circle D, we denote by radius(D) the *radius* of D.

Let $D_{\mathrm{opt}}(P, k)$ be a disk of minimum radius which contains k points of P, and let $r_{\mathrm{opt}}(P, k)$ denote the radius of $D_{\mathrm{opt}}(P, k)$.

Let P be a set of n points in the plane. Compute a set of $m = O(n/k)$ horizontal lines $h_1, \ldots, h_m$ such that between two consecutive horizontal lines, there are at most $k/4$ points of P in the strip they define. This can be easily done in $O(n \log(n/k))$ time using deterministic median selection together with recursion.[②] Similarly, compute a set of vertical lines $v_1, \ldots, v_m$, such that between two consecutive lines, there are at most $k/4$ points of P.

Consider the (non-uniform) grid G induced by $h_1, \ldots, h_m$ and $v_1, \ldots, v_m$. Let X be the set of all intersection points of G. We claim that $D_{\mathrm{opt}}(P, k)$ contains at least one point of X. Indeed, consider the center u

[②]Indeed, compute the median in the x-order of the points of P, split P into two sets, and recurse on each set, till the number of points in a subproblem is of size $\leq k/4$. We have $T(n) = O(n) + 2T(n/2)$, and the recursion stops for $n \leq k/4$. Thus, the recursion tree has depth $O(\log(n/k))$, which implies running time $O(n \log(n/k))$.

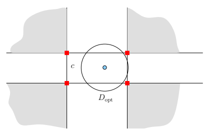


Figure 1.1: If the disk $D_{\rm opt}(P, k)$ does not contain and vertex of the cell c, then it does not cover any shaded area. As such, it can contain at most $k/2$ points, since the vertical and horizaontal strips containing c, each has at most $k/4$ points of P inside them.

of $D_{\rm opt}(P, k)$, and let c be the cell of G that contains u. Clearly, if $D_{\rm opt}(P, k)$ does not cover any of the four vertices of c, then it can cover only points in the vertical strip of G that contains c, and only points in the horizontal strip of G that contains c. See Figure 1.1. However, each such strip contains at most $k/4$ points. It follows that $D_{\rm opt}(P, k)$ contains at most $k/2$ points of P, a contradiction. Thus, $D_{\rm opt}(P, k)$ must contain a point of X. For every point $p \in X$, compute the smallest circle centered at p that contains k points of P. Clearly, for a point $q \in X \cap D_{\rm opt}(P, k)$, this yields the required 2-approximation. Indeed, the disk of radius $2D_{\rm opt}(P, k)$ centered at q contains at least k points of P since it also covers $D_{\rm opt}(P, k)$. We summarize as follows:

Lemma 1.3.1 *Given a set P of n points in the plane, and parameter k, one can compute in $O(n(n/k)^2)$ deterministic time, a circle D that contains k points of P, and* radius$(D) \leq 2r_{\rm opt}(P, k)$.

Corollary 1.3.2 *Given a set of P of n points and a parameter $k = \Omega(n)$, one can compute in linear time, a circle D that contains k points of P and* radius$(D) \leq 2r_{\rm opt}(P, k)$.

1.4 A Linear Time 2-Approximation Algorithm for the k-Enclosing Minimum Disk

In the following, we present a linear time algorithm for approximating the minimum eclosing disk. While interesting on their own right, the results here are not used later and can be skipped on first reading.

1.4.1 The algorithm

We refer to the algorithm of Lemma 1.3.1 as ApproxHeavy(P, k).

Remark 1.4.1 For a point set P of n points, the radius r returned by the algorithm of Lemma 1.3.1 is the distance between a vertex of the non-uniform grid, a point of P. As such, a grid G_r computed using this distance is one of $O(n^3)$ possible grids. Indeed, a circle is defined by the distance between a vertex of the non-uniform grid of Lemma 1.3.1, and a point of P. A vertex of such a grid is determined by two points of P, through which the vertical and horizontal line passes. Thus, there are $O(n^3)$ such triples.

Let $\mathtt{gd}_r(P)$ denote the maximum number of points of P mapped to a single point by the mapping G_r. Define $\mathtt{depth}(P, r)$ to be the maximum number of points of P that a circle of radius r can contain.

Lemma 1.4.2 *For any point set P, and $r > 0$, we have: (i) For any real number $A > 0$, it holds* $\texttt{depth}(P, Ar) \leq (A+1)^2 \, \texttt{depth}(P, r)$*, (ii)* $\texttt{gd}_r(P) \leq \texttt{depth}(P, r) \leq 9\texttt{gd}_r(P)$*, (iii) if* $r_{\mathrm{opt}}(P, k) \leq r \leq 2r_{\mathrm{opt}}(P, k)$ *then* $\texttt{gd}_r(P) \leq 5k$*, and (iv) Any circle of radius r is covered by at least one grid cluster in* G_r*.*

Proof: (i) Consider the disk D of radius Ar realizing $\texttt{depth}(P, Ar)$, and let D' be the disk of radius $(A + 1)r$ having the same center as D. For every point $p \in V = P \cap D$, place a disk around it of radius r, and let S denote the resulting set of disks. Since every disk of S has area πr^2, and they are all contained in D', which is of area $\pi(A + 1)^2 r^2$, it follows that there must be a point p inside D' which is contained in $\mu = \left\lceil \frac{|V| \pi r^2}{\pi(A+1)^2 r^2} \right\rceil = \left\lceil \frac{\texttt{depth}(P, Ar)}{(A+1)^2} \right\rceil$ disks. This means, that the disk $\mathcal{D}$ of radius r centered at p contains at least μ points of P. Now, $\mu \leq |\mathcal{D} \cap P| \leq \texttt{depth}(P, r)$. Thus, $\frac{\texttt{depth}(P, Ar)}{(A+1)^2} \leq \mu \leq \texttt{depth}(P, r)$, as claimed.

(iv) Consider a (closed) disk D of radius r, and let c be its center. If c is in the interior of a grid cell C, then the claim easily holds, since D can intersect only C or the cells adjacent to C. Namely, D is contained in the cluster centered at C. The problematic case, is when c is on the grid boundaries. Since grid cells are closed on one side, and open on the other, it is easy to verify that the claim holds again by careful and easy case analysis, which we will skip here.

(ii) Consider the grid cell C of $\mathrm{G}_r(P)$ that realizes $\texttt{gd}_r(P)$, and let c be a point placed in the center of C. Clearly, a disk D of radius $\sqrt{r^2 + r^2}/2 = r/\sqrt{2}$ centered at c, would cover completely the cell C. Thus, $\texttt{gd}_r(P) \leq |D \cap P| \leq \texttt{depth}(P, r)$. As for the other direction, observe that the disk D' realizing $\texttt{depth}(P, r)$, can intersect at most 9 cells of the grid G_r, by (iv). Thus, $\texttt{depth}(P, r) \leq |D' \cap P| \leq 9\texttt{gd}_r(P)$.

(iii) Let C be the grid cell of G_r realizing $\texttt{gd}_r(P)$. Place 4 points, at the corners of C, and one point in the center of C. Placing a disk of radius $r_{\mathrm{opt}}(P, k)$ at each of those points, completely covers C, as can be easily verified (since the side length of C is at most $2r_{\mathrm{opt}}(P, k)$). Thus, $|P \cap C| \leq \texttt{gd}_r(P) \leq 5 \, \texttt{depth}(P, r_{\mathrm{opt}}(P, k)) = 5k$. ■

1.4.1.1 Description

As in the previous sections, we construct a grid which partitions the points into small ($O(k)$ sized) groups. The key idea behind speeding up the grid computation is to construct the appropriate grid over several rounds. Specifically, we start with a small set of points as seed and construct a suitable grid for this subset. Next, we incrementally insert the remaining points, while adjusting the grid width appropriately at each step.

Definition 1.4.3 (Gradation) Given a set P of n points, a *sampling sequence* $(S_m, \ldots, S_1)$ of P is a sequence of subsets of P, such that (i) $S_1 = P$, (ii) S_i is formed by picking each point of S_{i-1} with probability $1/2$, and (iii) $|S_m| \leq 2k$, and $|S_{m-1}| > 2k$. The sequence $(S_m, S_{m-1}, \ldots, S_1)$ is called a *gradation* of P.

Lemma 1.4.4 *Given P, a* sampling sequence *can be computed in expected linear time.*

Proof: Observe that the sampling time is $O\left(\sum_{i=1}^{m} |S_i|\right)$, where m is the length of the sequence. Also observe that $\mathbf{E}[|S_1|] = |S_1| = n$ and

$$\mathbf{E}[|S_i|] = \mathbf{E}\left[\mathbf{E}\left[|S_i| \;\middle|\; |S_{i-1}|\right]\right] = \mathbf{E}\left[\frac{|S_{i-1}|}{2}\right] = \frac{1}{2}\,\mathbf{E}[|S_{i-1}|]\,.$$

Now by induction, we get

$$\mathbf{E}[|S_i|] = \frac{n}{2^{i-1}}.$$

Thus, the running time is $O\left(\mathbf{E}\left[\sum_{i=1}^{m} |S_i|\right]\right) = O(n)$. ■

```
GROW($P_i$,$r_{i-1}$,$k$)
        Output: $r_i$
    begin
        $\mathrm{G}_{i-1} \leftarrow \mathrm{G}_{r_{i-1}}(P_i)$
        for every grid cluster $c \in \mathrm{G}_{i-1}$ with $|c \cap P_i| \geq k$ do
            $P_c \leftarrow c \cap P_i$
            $r_c \leftarrow$ APPROXHEAVY($P_c, k$)
            // APPROXHEAVY is the algorithm of Lemma 1.3.1
            We have $r_{\mathrm{opt}}(P_c, k) \leq r_c \leq 2r_{\mathrm{opt}}(P_c, k)$,

        return minimum $r_c$ computed.
    end
```

Figure 1.2: Algorithm for the *i*th round.

Let $\mathcal{P} = (P_1, \ldots, P_m)$ be a gradation of P (see Definition 1.4.3), where $P = P_m$ and $|P_1| \geq \max(k, n/\log n)$ (i.e. if $k \geq n/\log n$ start from the first set in $\mathcal{P}$ that has more than k elements). The sequence $\mathcal{P}$ can be computed in expected linear time as shown in Lemma 1.4.4.

Since $|P_1| = O(k)$, we can compute r_1, in $O(|P_1| (|P_1| / k)^2) = O(k) = O(n)$ time, using Lemma 1.3.1.

The remaining algorithm works in m rounds, where m is the length of the sequence $\mathcal{P}$. At the end of the *i*th round, we have a distance r_i such that $\mathsf{gd}_{r_i}(P_i) \leq 5k$, and there exists a grid cluster in G_{r_i} containing more than k points of P_i and $r_{\mathrm{opt}}(P_i) \leq r_i$.

At the *i*th round, we first construct a grid G_{i-1} for points in P_i using r_{i-1} as grid width. We know that there is no grid cell containing more than $5k$ points of P_{i-1}. As such, intuitively, we expect every cell of G_{i-1} to contain at most $10k$ points of P_i, since $P_{i-1} \subseteq P_i$ was formed by choosing each point of P_i into P_{i-1} with probability $1/2$. (This is of course too good to be true, but something slightly weaker does hold.) Thus allowing us to use the slow algorithm of Lemma 1.3.1 on those grid clusters. Note that, for $k = \Omega(n)$, the algorithm of Lemma 1.3.1 runs in linear time, and thus the overall running time is linear.

The algorithm used in the *i*th round is more concisely stated in Figure 1.2. At the end of the m rounds we have r_m, which is a 2-approximation to the radius of the optimal k enclosing circle of $P_m = P$. The overall algorithm is summarized in Figure 1.3.

1.4.1.2 Analysis

Lemma 1.4.5 *For $i = 1, \ldots, m$, we have $r_{\mathrm{opt}}(P_i, k) \leq r_i \leq 2r_{\mathrm{opt}}(P_i, k)$, and the heaviest cell in $\mathrm{G}_{r_i}(P_i)$ contains at most $5k$ points of P_i.*

Proof: Consider the optimal circle D_i that realizes $r_{\mathrm{opt}}(P_i, k)$. Observe that there is a cluster c of $\mathrm{G}_{r_{i-1}}$ that contains D_i, as $r_{i-1} \geq r_i$. Thus, when GROW handles the cluster c, we have $D_i \cap P_i \subseteq c$. The first part of the lemma then follows from the correctness of the algorithm of Lemma 1.3.1.

As for the second part, observe that any grid cell of width r_i can be covered with 5 circles of radius $r_i/2$, and $r_i/2 \leq r_{\mathrm{opt}}(P_i, k)$. It follows that each grid cell of $\mathrm{G}_{r_i}(P_i)$ contains at most $5k$ points. ■

Now we proceed to upper-bound the number of cells of $\mathrm{G}_{r_{i-1}}$ that contains "too many" points of P_i. Since each point of P_{i-1} was chosen from P_i with probability $1/2$, we can express this bound as a sum of independent random variables, and bound this using tail-bounds.

```
LinearApprox($P, k$)
        Output: $r$ - a 2-approximation to $r_{\mathrm{opt}}(P, k)$
    begin
        Compute a gradation $\{P_1, \ldots, P_m\}$ of P as in Lemma 1.4.4
        $r_1 \leftarrow$ ApproxHeavy($P_1, k$)
            // ApproxHeavy is the algorithm of Lemma 1.3.1
            // which outputs a 2-approximation

        for $i \leftarrow 2$ to $m$ do
            $r_j \leftarrow$ Grow($P_i, r_{i-1}, k$)

        for every grid cluster $c \in \mathrm{G}_{r_m}$ with $|c \cap P| \geq k$ do
            $r_c \leftarrow$ ApproxHeavy($c \cap P, k$)

        return minimum $r_c$ computed over all clusters
    end
```

Figure 1.3: 2-Approximation Algorithm.

Definition 1.4.6 For a point set P, and parameters k and r, the *excess* of $\mathrm{G}_r(P)$ is

$$\mathcal{E}(P, k, \mathrm{G}_r) = \sum_{c \in \mathrm{Cells}(\mathrm{G}_r)} \left\lfloor \frac{|c \cap P|}{50k} \right\rfloor,$$

where $\mathrm{Cells}(\mathrm{G}_r)$ is the set of cells of the grid G_r.

Remark 1.4.7 The quantity $100k \cdot \mathcal{E}(P, k, \mathrm{G}_r)$ is an upper bound on the number of points of P in an heavy cell of $\mathrm{G}_r(P)$, where a cell of $\mathrm{G}_r(P)$ is *heavy* if it contains more than $50k$ points.

Lemma 1.4.8 *For any positive real t, the probability that $\mathrm{G}_{r_{i-1}}(P_i)$ has excess $\mathcal{E}(P_i, k, \mathrm{G}_{r_{i-1}}) \geq \alpha = t + 5\lceil \log(n) \rceil$, is at most 2^{-t}.*

Proof: Let $\mathfrak{G}$ be the set of $O(n^3)$ possible grids that might be considered by the algorithm (see Remark 1.4.1), and fix a grid $\mathsf{G} \in \mathfrak{G}$ with excess $M = \mathcal{E}(P_i, k, \mathsf{G}) \geq \alpha$.

Let $U = \left\{ \{P_i \cap c\} \;\middle|\; c \in \mathsf{G}, |P_i \cap c| > 50k \right\}$ be all the heavy cells in $\mathsf{G}(P_i)$. Furthermore, let $V = \bigcup_{X \in U} \psi(X, 50k)$, where $\psi(X, \nu)$ denotes an arbitrary partition of the set X into disjoint subsets such that each one of them contains ν points, except maybe the last subset that might contain between ν and $2\nu - 1$ points.

It is clear that $|V| = \mathcal{E}(P_i, k, \mathsf{G})$. From the Chernoff inequality, for any $S \in V$, we have $\mu = \mathbf{E}[|S \cap P_{i-1}|] \geq 25k$, and setting $\delta = 4/5$ we have

$$\mathbf{Pr}[|S \cap P_{i-1}| \leq 5k] \leq \mathbf{Pr}[|S \cap P_{i-1}| \leq (1-\delta)\mu] < \exp\left(-\mu\frac{\delta^2}{2}\right) = \exp\left(-\frac{25k(4/5)^2}{2}\right) < \frac{1}{2}.$$

Furthermore, since $\mathsf{G} = \mathrm{G}_{r_{i-1}}$ implying that each cell of $\mathsf{G}(P_{i-1})$ contains at most $5k$ points. Thus we have

$$\mathbf{Pr}[\mathrm{G}_{r_{i-1}} = \mathsf{G}] \leq \prod_{S \in V} \mathbf{Pr}[|S \cap P_{i-1}| \leq 5k] \leq \frac{1}{2^{|V|}} = \frac{1}{2^M} \leq \frac{1}{2^\alpha}.$$

Since there are n^3 different grids in $\mathfrak{G}$, we have

$$\begin{aligned}\mathbf{Pr}[\mathcal{E}(P_i, k, \mathsf{G}_{r_{i-1}}) \geq \alpha] &= \mathbf{Pr}\left[\bigcup_{\substack{\mathsf{G}\in\mathfrak{G},\\ \mathcal{E}(P_i,k,\mathsf{G})\geq\alpha}} (\mathsf{G} = \mathsf{G}_{r_{i-1}})\right] \\ &\leq \sum_{\substack{\mathsf{G}\in\mathfrak{G},\\ \mathcal{E}(P_i,k,\mathsf{G})\geq\alpha}} \mathbf{Pr}[\mathsf{G} = \mathsf{G}_{r_{i-1}}] \leq n^3 \frac{1}{2^\alpha} \leq \frac{1}{2^t}.\end{aligned}$$

■

We next bound the expected running time of the algorithm LinearApprox by bounding the expected time spent in the ith iteration. In particular, let Y be the random variable which is the excess of $\mathsf{G}_{r_{i-1}}(P_i)$. In this case, there are at most Y cells which are heavy in $\mathsf{G}_{r_{i-1}}(P_i)$, and each such cell contains at most $O(Yk)$ points. Thus, invoking the algorithm of ApproxHeavy on such a heavy cell takes $O(Yk \cdot ((Yk)/k)^2) = O(Y^3 k)$ time. Overall, the the running time of Grow, in the ith iteration, is $T(Y) = O\big(|P_i| + Y \cdot Y^3 k\big) = O\big(|P_i| + Y^4 k\big)$.

For technical reasons, we need to consider the light and heavy cases separately to bound Y.

The Light Case: $k < 4\log n$**.** We have that the expected running time is proportional to

$$\begin{aligned}\sum_{t=0}^{\lceil n/k\rceil} \mathbf{Pr}[Y = t]\, T(t) &= |P_i| + \mathbf{Pr}\Big[0 \leq Y \leq 2\lceil\log n\rceil\Big]\, T(2\lceil\log n\rceil) + \sum_{t=2\lceil\log n\rceil+1}^{n/k} \mathbf{Pr}[Y = t]\, T(t) \\ &\leq |P_i| + T(2\lceil\log n\rceil) + \sum_{t=1}^{n/k} \frac{1}{2^t} T(t + 2\lceil\log n\rceil) \\ &= O(|P_i|) + O(k\log^4 n) + \sum_{t=1}^{n/k} \frac{(t + 2\lceil\log n\rceil)^4 k}{2^t} \\ &= O\Big(|P_i| + k\log^4 n\Big) = O(|P_i|),\end{aligned}$$

by Lemma 1.4.8 and since $T(\cdot)$ is a monotone increasing function.

The Heavy Case: $k \geq 4\log n$**.**

Lemma 1.4.9 *The probability that* $\mathsf{G}_{r_{i-1}}(P_i)$ *has excess larger than t, is at most* 2^{-t}*, for* $k \geq 4\log n$*.*

Proof: We use the same technique as in Lemma 1.4.8. By the Chernoff inequality, the probability that any $50k$ size subset of P_i would contain at most $5k$ points of P_{i-1}, is less than

$$\leq \exp\left(-25k \cdot \frac{16}{25} \cdot \frac{1}{2}\right) \leq \exp(-5k) \leq \frac{1}{n^4}.$$

In particular, arguing as in Lemma 1.4.8, it follows that the probability that $\mathcal{E}(P_i, k, r_{i-1})$ exceeds t, is smaller than $n^3/n^{4t} \leq 2^{-t}$. ■

Thus, if $k \geq 4\log n$, the expected running time of Grow, in the ith iteration, is at most

$$O\left(\sum_{c\in\mathsf{G}_{r_{i-1}}} |c \cap P_i| \left(\frac{|c \cap P_i|}{k}\right)^2\right) = O\left(|P_i| + \sum_{t=1}^{\infty} t \cdot \left(tk \cdot \left(\frac{tk}{k}\right)^2\right) \cdot \frac{1}{2^t}\right) = O(|P_i| + k) = O(|P_i|),$$

by Lemma 1.4.9.

Overall Running Time Analysis Thus, by the above analysis and by Lemma 1.4.4, the total expected running time of LinearApprox inside the inner loop is $O(\sum_i |P_i|) = O(n)$. As for the last step, of computing a 2-approximation, consider the grid $G_{r_m}(P)$. Each grid cell contains at most $5k$ points, and hence each grid cluster contains at most $45k$ points. Also the smallest k enclosing circle is contained in some grid cluster. In each cluster that contain more than k points, we use the algorithm of Corollary 1.3.2 and finally output the minimum over all the clusters. The overall running time is $O((n/k)k) = O(n)$ for this step, since each point belongs to at most 9 clusters.

Theorem 1.4.10 *Given a set P of n points in the plane, and a parameter k, one can compute, in expected linear time, a radius r, such that $r_{\mathrm{opt}}(P,k) \le r \le 2r_{\mathrm{opt}}(P,k)$.*

Once we compute r such that $r_{\mathrm{opt}}(P,k) \le r \le 2r_{\mathrm{opt}}(P,k)$, using the algorithm of Theorem 1.4.10, we apply an exact algorithm to each cluster of the grid $G_r(P)$ which contains more than k points.

Matoušek presented such an exact algorithm [Mat95], and it has running time of $O(n \log n + nk)$ and space complexity $O(nk)$. Since r is a 2 approximation to $r_{\mathrm{opt}}(P,k)$, each cluster has $O(k)$ points. Thus the running time of the exact algorithm in each cluster is $O(k^2)$ and requires $O(k^2)$ space. The number of clusters which contain more than k points is $O(n/k)$. Hence the overall running time is $O(nk)$, and the space used is $O(n + k^2)$.

Theorem 1.4.11 *Given a set P of n points in the plane and a parameter k, one can compute, in expected $O(nk)$ time, using $O(n + k^2)$ space, the radius $r_{\mathrm{opt}}(P,k)$, and a circle $D_{\mathrm{opt}}(P,k)$ that covers k points of P.*

1.5 Bibliographical notes

Our closest-pair algorithm follows Golin *et al.* [GRSS95]. This is in turn a simplification of a result of Rabin [Rab76]. Smid provides a survey of such algorithms [Smi00]).

The proof of Lemma 1.4.2 is from [Mat95]. The min-disk approximation algorithm follows roughly the work of Har-Peled and Mazumdar [HM03]. Exercise 1.6.2 is also taken from there.

1.6 Exercises

Exercise 1.6.1 [10 Points] Let C and P be two sets of points in the plane, such that $k = |C|$ and $n = |P|$. Let $r = \max_{p \in P} \min_{c \in C} \|c - p\|$ be the *covering radius* of P by C (i.e., if we place a disk of radius r around each point of C all those disks covers the points of P).

1. Give a $O(n + k \log n)$ expected time algorithm that outputs a number α, such that $\alpha \le r \le 10\alpha$.

2. For $\varepsilon > 0$ a prescribed parameter, give a $O(n + k\varepsilon^{-2} \log n)$ expected time algorithm that outputs a number α, such that $\alpha \le r \le (1 + \varepsilon)\alpha$.

Exercise 1.6.2 Given a set P of n points in the plane, and parameter k, present a (simple) randomized algorithm that computes, in expected $O(n(n/k))$ time, a circle D that contains k points of P, and radius$(D) \le 2r_{\mathrm{opt}}(P,k)$.

(This is a faster and simpler algorithm than the one presented in Lemma 1.3.1.)

Chapter 2

Quadtrees - Hierarchical Grids

In this chapter, we discuss quadtrees and compressed quadtrees which are arguably one of the simplest geometric data-structure. We begin in Section 2.1 by giving a simple application of quadtrees and describe a clever way for performing point-location queries quickly in such a quadtree. In Section 2.2 we describe how such quadtrees can be compressed and how can they be quickly constructed and used for point-location queries. In Section 2.3 we describe a randomized extension of this data-structure, known as *skip-quadtree*, which enables us to maintain the compressed quadtree efficiently under insertions and deletions. In Section 2.4, we turn our attention to applications of compressed quadtrees, showing how quadtrees can be used to compute good triangulations of an input point set.

2.1 Quadtrees - a simple point-location data-structure

Let P be a planar map. To be more concrete, let P be a partition of the unit square into triangles (i.e., a mesh). Since any simple polygon can be triangulated, P can represent any planar map, and its partition of the unit square into different regions, with different properties. For the sake of simplicity, assume that every vertex in P shares at most, say, nine triangles.

Let us assume that we want to preprocess P for point-location queries. Of course, there are data-structures that can do it with $O(n \log n)$ preprocessing time, linear space, and logarithmic query time. Instead, let us consider the following simple solution (which in the worst case, can be much worse).

Build a tree T, where the root corresponds to the unit square. Every node $v \in T$ corresponds to a cell $\square_v$ (i.e., a square), and it has four children. The four children correspond to the four squares formed by splitting $\square_v$ into four equal size squares, by horizontal and vertical cuts. The construction is recursive, and we start from $v = \mathrm{root}_T$. As long as the current node intersects more than, say, nine triangles, we create its children nodes, and we call recursively on each child, with the list of input triangles that intersect its square. We stop at a leaf, if its "conflict-list" (i.e., list of triangles it intersects) is of size at most nine. We store this conflict-list in the leaf.

Given a query point q, in the unit square, we can compute the triangle of P containing q, by traversing down T from the root, repeatedly going into the child of the current node, whose square contains q. We stop at soon as we reach a leaf, and then we scan the leaf conflict-list, and check which of the triangles contains q.

Of course, in the worst case, if the triangles are long and skinny, this quadtree might have unbounded complexity. However, for reasonable inputs (say, the triangles are fat), then the quadtree would have linear complexity in the input size (see Exercise 2.6.1) .The big advantage of quadtrees of course, is their simplicity. In a lot of cases, quadtree would be a sufficient solution, and seeing how to solve a problem using a quadtree might be a first insight into a problem.

```
FastPntLocInner($T, q, lo, hi$).
    $mid \leftarrow \lfloor (lo + hi)/2 \rfloor$
    $v \leftarrow$ GetNode($T, q, mid$)
    if $v = null$ then
        return FastPntLocInner($T, q, lo, mid - 1$).
    $w \leftarrow Child(v, q)$ //$w$ is child of $v$ containing the point $q$.
    If $w = null$ then
        return $v$
    return FastPntLocInner($T, q, mid + 1, hi$)
```

Figure 2.1: One can perform point-location in a quadtree T by calling `FastPntLocInner` $(T, q, 0, \mathrm{height}(T))$.

2.1.1 Fast point-location in a quadtree

One possible interpretation of quadtrees is that they are a multi-grid representation of a point-set. In particular, given a node v, with a square S_v, which is of depth i (the root has depth zero), then the side length of S_v is 2^{-i}, and it is a square in the grid $G_{2^{-i}}$. In fact, we will refer to $\ell(v) = -i$ as the *level* of v. However, a cell in a grid has a unique ID made out of two integer numbers. Thus, a node v of a quadtree is uniquely defined by the triple $\mathrm{id}(v) = (\ell(v), \lfloor x/r \rfloor, \lfloor y/r \rfloor)$, where (x, y) is any point in $\square_v$, and $r = 2^{\ell(v)}$.

Furthermore, given a query point q, and a desired level ℓ, we can compute the the ID of the quadtree cell of this level that contains q in constant time. Thus, this suggests a very natural algorithm for doing a point-location in a quadtree: Store all the IDs of nodes in the quadtree in a hash-table, and also compute the maximal depth h of the quadtree. Given a query point q, we now have access to any node along the point-location path of q in T, in constant time. In particular, we want to find the point in T where the point-location path "falls off" the quadtree. This we can find by performing a binary search for the dropping off point. Let `GetNode`(T, q, d) denote the procedure that, in constant time, returns the node v of depth d in the quadtree T such that $\square_v$ contains the point q. Given a query point q, we can perform point-location in T by calling `FastPntLocInner`$(T, q, 0, \mathrm{height}(T))$. See Figure 2.1 for the pseudo-code for `FastPntLocInner`.

Lemma 2.1.1 *Given a quadtree T of size n and of height h, one can preprocess it in linear time, such that one can perform a point-location query in T in $O(\log h)$ time. In particular, if the quadtree has height $O(\log n)$ (i.e., it is "balanced"), then one can perform a point-location query in T in $O(\log \log n)$ time.*

2.2 Compressed Quadtrees: Range Searching Made Easy

Let P be a set of n points in the plane of spread $\Phi = \Phi(P)$.

Definition 2.2.1 For a set P of n points in any metric space, let $\Phi(P) = \left(\max_{p,q \in P} \|pq\|\right) / \left(\min_{p,q \in P, p \neq q} \|pq\|\right)$ be the *spread* of P. In words, the spread of P is the ratio between the diameter of P and the distance between the two closest points. Intuitively, the spread tells us the range of distances that P posses.

One can build a quadtree for P, storing the points of P in the leaves of P, where one keep splitting a node as long as it contains more than one point of P. During this recursive construction, if a leaf contains no points of P, we save space by not creating this leaf, and instead creating a null pointer in the parent node for this child.

Lemma 2.2.2 *Let P be a set of n points in the unit square, such that* $\mathrm{diam}(P) = \max_{p,q \in P} \|pq\| \geq 1/2$. *Let T be a quadtree of P constructed over the unit square. Then, the depth of T is bounded by $O(\log \Phi(P))$, it can be constructed in $O(n \log \Phi(P))$ time, and the total size of T is $O(n \log \Phi(P))$.*

Proof: The construction is done by a straightforward recursive algorithm. Let us bound the depth of T. Consider any two points $p, q \in P$, and observe that a node v of T of level $u = \lfloor \lg \|pq\| \rfloor - 1$ containing p must not contain q (we remind the reader that $\lg n = \log_2 n$). Indeed, the diameter of $\square_v$ is smaller than $\sqrt{2}2^u < \sqrt{2}\|pq\|/2 < \|pq\|$. Thus, $\square_v$ can not contain both p and q. In particular, any node of T of level $r = -\lfloor \lg \Phi \rfloor - 1$ can contain at most one point of P, where $\Phi = \Phi(P)$. Thus, all the nodes of T are of depth $O(\log \Phi)$.

Since the construction algorithm spends $O(n)$ time at each level, it follows that the construction time is $O(n \log \Phi)$, and this also bounds the size of the quadtree T. ■

The bounds of Lemma 2.2.2 are tight, as one can easily verify, see Exercise 2.6.2. But in fact, if you inspect a quadtree generated by Lemma 2.2.2, you would realize that there are a lot of nodes of T which are of degree one. Indeed, a node v of T has degree larger than one, only if it has two children, and let P_v be the subset of points of P stored in the subtree of v. Such a node v splits P_v into, at least, two subsets and globally there can be only $n - 1$ such splitting nodes.

Thus, a quadtree T contains a lot of "useless" nodes. We can replace such a sequence of edges by a single edge. To this end, we will store inside each quadtree node v, its square $\square_v$, and its level $\ell(v)$. Given a path of vertices in the quadtree that are of degree one, we will replace them with a single edge, from the first node in the path, till the last vertex in this path (this is the first node of degree larger than one). We call the resulting tree a *compressed* quadtree. Since all internal nodes in the new compressed quadtree have degree larger than one, it follows that it has linear size (however, it still can have linear depth).

As an application for such a compressed quadtree, consider the problem of counting how many points are inside a query rectangle r. We can start from the root of the quadtree, and recursively traverse it, going down a node only if its region intersects the query rectangle. Clearly, we will report all the points contained inside r. Of course, we have no guarantee about the query time performance of the query, but in practice, this might be fast enough.

2.2.1 Efficient construction of compressed quadtrees

Let P be a set of n points in the unit square, with unbounded spread. We are interested in computing the compressed quadtree of P. The regular algorithm for computing a quadtree when applied to P might required unbounded time. Modifying it so it requires only quadratic time is an easy exercise.

Instead, compute in linear time a disk D of radius r, which contains at least $n/10$ of the points of P, such that $r \leq 2r_{\mathrm{opt}}(P, n/10)$, where $r_{\mathrm{opt}}(P, n/10)$ denotes the radius of the smallest disk containing $n/10$ points. Computing D can be done in linear time, by a rather simple algorithm (Lemma 1.3.1).

Let $l = 2^{\lfloor \lg r \rfloor}$. Consider the grid G_l. It has a cell that contains $(n/10)/25$ points (since D is covered by $5 \times 5 = 25$ grid cells of G_l, since $l \geq r/2$), and no grid cell contains more than $5(n/10)$ points, by Lemma 1.4.2 (iii). Thus, compute $\mathrm{G}_l(P)$, and find the cell c containing the largest number of points. Let P_{in} be the points inside this cell c, and P_{out} the points outside this cell. We know that $|P_{\mathrm{in}}| \geq n/250$, and $|P_{\mathrm{out}}| \geq n/2$. Next, compute the compressed quadtrees for P_{in} and P_{out}, respectively, and let T_{in} and T_{out} denote the respective quadtrees. Since the cell of the root of T_{in} has side length which is a power of two, and it belongs to the grid G_l, it follows that c represents a valid region, which can be a node in T_{out} (note that if it is a node in T_{out}, then it is empty). Thus, we can do a point-location query in T_{out}, and hang the root of T_{in} in the appropriate node of T_{out}. This takes linear time (ignoring the time to construct T_{in} and T_{out}). Thus, the overall construction time is $O(n \log n)$.

Theorem 2.2.3 *Given a set P of n points in the plane, one can compute a compressed quadtree of P in $O(n \log n)$ deterministic time.*

A square is a *canonical square*, if it is contained inside the unit square, it is a cell in a grid G_r, and r is a power of two (i.e., it might correspond to a node in a quadtree). For reasons that would become clear later, we want to construct the quadtree out of a list of quadtree nodes that must appear in the quadtree. Namely, we get a list of canonical grid cells that must appear in the quadtree (i.e., the level of the node, together with its grid ID).

Lemma 2.2.4 *Given a list C of n canonical squares, all lying inside the unit square, one can construct a compressed quadtree T such that for any square $c \in C$, there exists a node $v \in T$, such that $\square_v = c$. The construction time is $O(n \log n)$.*

Proof: The construction is similar to Theorem 2.2.3. Let P be a set of n points, where $p_c \in P$, if $c \in C$, and p_c is the center of c. Next, find, in linear time, a canonical square C that contains at least $n/250$ points of P, and at most $n/2$ points of P. Let U be the list of all squares of C that contain c, let C_{in} be the list of squares contained inside c, and let C_{out} be the list of squares of C that do not intersect the interior of c. Recursively, build a compressed quadtree for C_{in} and C_{out}, denoted by T_{in} and T_{out}, respectively.

Next, sort the nodes of U in decreasing order of their level. Also, let π be the point-location path of c in T_{out}. Clearly, adding all the nodes of U to T_{out} is no more than performing a merge of π together with the sorted nodes of U. Whenever we encounter a square of U that does not have a corresponding node at π, we create this node, and insert it into π. Let T'_{out} denote the resulting tree. Next, we just hang T_{in} in the right place in T'_{out}. Clearly, the resulting quadtree has all the squares of C as nodes.

As for the running time, we have $T(C) = T(C_{\text{in}}) + T(C_{\text{out}}) + O(n) + O(|U| \log |U|) = O(n \log n)$, since $|C_{\text{out}}| + |C_{\text{in}}| + |U| = n$ and $|C_{\text{in}}|, |C_{\text{out}}| \le (249/250)n$. ■

2.2.2 Fingering a Compressed Quadtree - Fast Point Location

Let T be a compressed quadtree of size n. We would like to preprocess it so that given a query point, we can find the lowest node of T whose cell contains a query point q. As before, we can perform this by traversing down the quadtree, but this might require $\Omega(n)$ time. Since the range of levels of the quadtree nodes is unbounded, we can no longer use binary search on the levels of T to answer the query.

Instead, we are going to use a rebalancing technique on T. Namely, we are going to build a balanced tree T', which would have cross pointers (i.e., fingers) into T. The search would be performed on T' instead of on T. In the literature, the tree T is known as a *finger tree*.

Definition 2.2.5 Let T be a tree with n nodes. A *separator* in T is a node v, such that if we remove v from T, we remain with a forest, such that every tree in the forest has at most $\lceil n/2 \rceil$ vertices.

Lemma 2.2.6 *Every tree has a separator, and it can be computed in linear time.*

Proof: Consider T to be a rooted tree, and initialize v to be the root of T. We perform a walk on T. If v is not a separator, then one of the children of v in T must have a subtree of T of size $\ge \lceil n/2 \rceil$ nodes. Set v to be this node. Continue in this walk, till we get stuck. The claim is that v is the required node. Indeed, since we always go down, and the size of the subtree shrinks, we must get stuck. Thus, consider w as the node we got stuck at. Clearly, the subtree of w contains at least $\lceil n/2 \rceil$ nodes (otherwise, we would not set $v = w$). Also, all the subtrees of w have size $\le \lceil n/2 \rceil$, and the connected component of $T \setminus \{w\}$ containing the root contains at most $n - \lceil n/2 \rceil \le \lfloor n/2 \rfloor$ nodes. Thus, w is the required separator. ■

This suggests a natural way for processing a compressed quadtree for point-location queries. Find a separator $v \in T$, and create a root node f_v for T' which has a pointer to v; now recursively build finger trees to each tree of $T \setminus \{v\}$, and hang them on w. Given a query point q, we traverse T', where at node $f_v \in T'$, we check whether the query point $q \in \square_v$, where v is the corresponding node of T. If $q \notin \square_v$, we continue the search into the child of f_v, which corresponds to the connected component outside $\square_v$ that was hung on f_v. Otherwise, we continue into the child that contains q. This takes constant time per node. As for the depth for the finger tree T', observe $D(n) \leq 1 + D(\lceil n/2 \rceil) = O(\log n)$. Thus, a point-location query in T' takes logarithmic time.

Theorem 2.2.7 *Given a compressed quadtree T of size n, one can preprocess it in $O(n \log n)$ time, such that given a query point q, one can return the lowest node in T whose region contains q in $O(\log n)$ time.*

2.3 Dynamic Quadtrees

What if we want to maintain the compressed quadtree under insertions and deletions? There is an elegant way of doing this by using randomization. The resulting structure has similar behavior to skip-list where instead of linked list we use quadtrees.

We remind the reader the concept of gradation:

Definition 2.3.1 (Gradation) Given a set P of n points, a *sampling sequence* $(S_m, \ldots, S_1)$ of P is a sequence of subsets of P, such that (i) $S_1 = P$, (ii) S_i is formed by picking each point of S_{i-1} with probability $1/2$, and (iii) $|S_m| \leq 2k$, and $|S_{m-1}| > 2k$. The sequence $(S_m, S_{m-1}, \ldots, S_1)$ is called a *gradation* of P.

Let $T_1, \ldots, T_m$ be the quadtrees of the sets $P = S_1, \ldots, S_m$. Note, that the nodes of T_i are a subset of the nodes appear in T_{i-1}. As such, every node in T_i would have pointers to its own copy in T_{i-1} and a pointer to its copy in T_{i+1} if it exists there. We will refer to this data-structure is *skip-quadtree*.

Point-location queries. Given a query point q we want to find the leaf of T_1 that contains it. The search algorithm is quite simple, starting at T_m you find the leaf in T_i that contains the query point, and then move to the corresponding node in T_{i-1}, and continue the search from there.

2.3.1 Inserting a point into the skip-quadtree.

Let p be the point to be inserted into the skip-quadtree. We perform a point-location query and find the lowest node v in T_1 that contains p. Next, we split v and establish a new node (hanging from v) that contains p. Next, we flip an unbiased coin, if the coin comes up tail, we are done. Otherwise, we add p to T_2. We continue in this fashion, adding p to the quadtrees in the relevant levels, till the coin comes up tail.

Note, that the amount of work and space needed at each level is a constant (ignoring the initial point-location query), and by implementing this operation carefully, the time to perform it would proportional to the point-location query time.

2.3.2 Deleting a point from the skip-quadtree.

Done in a similar fashion to the insertion described above.

2.3.3 Constructing the skip-quadtree.

Given a point-set P, we just insert the points one by one into the skip-quadtree.

2.3.4 Running time analysis

We analyze the time needed to perform a point-location query. In particular, we claim that the expected query time $O(\log n)$. To see that we will use the standard backward analysis. Consider the leaf v of T_i that contains q, and consider the path $v = v_1, v_2, \ldots, v_r$ from v to the root v_r of the compressed quadtree T_i. Let π denote this path. Clearly, the amount of time spent in the search in the tree T_i, is proportional to how far we have to go on this list, till we hit a node that appears in T_{i+1}. Note, that the node v_j stores (at least) j points of S_i, and if any pair of them appears in S_{i+1} then at least one of the nodes on π below the node v_j would appear in T_{i+1} (since the quadtree T_{i+1} needs to "separate" these two points and this is done by a node of the path). Let denote this set of points by U_j, and let X_j be an indicator variable which is one if v_j does not appear in T_{i+1}. Clearly,

$$\mathbf{E}\Big[X_j\Big] = \mathbf{Pr}\Big[\text{no pair of points of } U_j \text{ is in } T_{i+1}\Big] \leq \frac{j+1}{2^j},$$

since the points of S_i are randomly and independently chosen to be in S_{i+1} and the event happens only if zero or one points of U_j are in S_{i+1}.

Thus, the expected search time in T_i is $\mathbf{E}\Big[\sum_j X_j\Big] = \sum_j \mathbf{E}\Big[X_j\Big] = \sum_j (j+1)/2^j = O(1)$.

Thus, the overall expected search time in the skip-quadtree is proportional to the number of levels in the gradation.

Let $Z_i = |S_i|$ be the nub-mer elements stored in the ith level of the gradation. We know that $Z_1 = n$, and $\mathbf{E}[Z_i] =$. In particular, $\mathbf{E}[Z_i] = \mathbf{E}[\mathbf{E}[Z_i]] = \mathbf{E}[Z_{i-1}/2] = \cdots = n/2^{i-1}$. Thus, $\mathbf{E}[Z_\alpha] \leq 1/n^{10}$, where $\alpha = \lceil 11 \lg n \rceil$. Thus, by Markov's inequality, we have that

$$\mathbf{Pr}[m > \alpha] = \mathbf{Pr}[Z_\alpha \geq 1] \leq \frac{\mathbf{E}[Z_\alpha]}{1} = \frac{1}{n^{10}}.$$

We summarize:

Lemma 2.3.2 *A gradation defined over n elements has $O(\log n)$ levels both in expectation and with high probability.*

This implies that a point-location query in the skip quadtree takes, in expectation, $O(\log n)$ time.

Since, with high probability, there are only $O(\log n)$ levels in the gradation, it follows that the expected search time is $O(\log n)$.

High Probability. In fact, one can show that this point-location query time bound holds with high probability, and we sketch (informally) the argument why this is true. Consider the variable Y_i that is the number of nodes of T_i being visited during the point-location query. We have that $\mathbf{Pr}[Y_i \geq k] \leq \sum_{j=k}(k+1)/2^k = O(k/2^k) = O(1/c^k)$, for some constant $c > 1$. Thus, we have a sum of logarithmic number of independent random variables, each one of them behaves like a variable with geometric distribution. As such, we can apply a Chernoff-type inequality (see Exercise 19.4.3) to get an upper bound on the probably that the sum of these variables exceeds $O(\log n)$. This probability is bounded by $1/n^{O(1)}$.

Note, the longest query time is realized by one of the points stored in the quadtree. Since there are n points stored in the quadtree, this implies that with high probability *all* point-location queries takes $O(\log n)$ time. Also, observe that the structure of the skip-quadtree is uniquely determined by the gradation. Since the gradation is oblivious to the history of the data-structure (i.e., what points where inserted and deleted). As such, these bounds on the performance hold at any point in time during the usage of the skip-quadtree. We summarize:

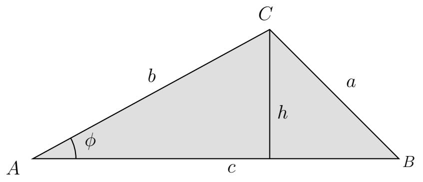
Theorem 2.3.3 *Let T be an empty skip-quadtree used for a sequence of n operations (i.e., insertions,deletions and point-location queries). Then, with high probability (and thus also in expectation), the time to perform each such operation takes* $O(\log n)$ *time.*

2.4 Balanced quadtrees, and good triangulations

The *aspect ratio* of a convex body is the ratio between its longest dimension and its shortest dimension. For a triangle $\triangle = abc$, the aspect ratio $\mathcal{A}_{\text{ratio}}(\triangle)$ is the length of the longest side divided by the height of the triangle on the longest edge.

Lemma 2.4.1 *Let ϕ be the smallest angle for a triangle. We have that* $1/\sin\phi \le \mathcal{A}_{\text{ratio}}(\triangle) \le 2/\sin\phi$.

Proof: Consider the triangle $\triangle = \triangle abc$.



We have $\mathcal{A}_{\text{ratio}}(\triangle) = c/h$. However, $h = b\sin\phi$, and since a is the shortest edge in the triangle (since it is facing the smallest angle), it must be that b is the middle length edge. As such, $2b \ge a + b \ge c$. Thus, $\mathcal{A}_{\text{ratio}}(\triangle) \ge b/h = b/(b\sin\phi) = 1/\sin\phi$. And similarly, $\mathcal{A}_{\text{ratio}}(\triangle) \le 2b/h = 2b/(b\sin\phi) = 2/\sin\phi$. ■

Another natural measure of sharpness is the *edge ratio* $E_{\text{ratio}}(\triangle)$, which is the ratio between a triangle's longest and shortest edges. Clearly, $\mathcal{A}_{\text{ratio}}(\triangle) > E_{\text{ratio}}(\triangle)$, for any triangle $\triangle$. For a triangulation $\mathcal{T}$, we denote by $\mathcal{A}_{\text{ratio}}(\mathcal{T})$ the maximum aspect ratio of a triangle in $\mathcal{T}$. Similarly, $E_{\text{ratio}}(\mathcal{T})$ denotes the maximum edge ratio of a triangle in $\mathcal{T}$.

Definition 2.4.2 A *corner* of a quadtree cell is one of the four vertices of its square. The *corners* of the quadtree are the points that are corners of its cells. We say that the side of a cell is *split* if either of the neighboring boxes sharing it is split. A quadtree is *balanced* if any side of an unsplit cell may contain only one quadtree corner in its interior. Namely, adjacent leaves are either of the same level, or of adjacent levels.

Lemma 2.4.3 *Let P be a set of points in the plane, such that* $\text{diam}(P) = \Omega(1)$ *and* $\Phi = \Phi(P)$. *Then, one can compute a (minimal size) balanced quadtree T of P, in time* $O(n\log n + m)$ *time, where m is the size of the output quadtree.*

Proof: Compute a compressed quadtree T of P in $O(n\log n)$ time. Next, we traverse T, and replace every compressed edge of T by the sequence of quadtree nodes that defines it. To guarantee the balance condition, we create a queue of the nodes of T, and store the nodes of T in a hash table, with their IDs.

We handle the nodes in the queue, one by one. For a node v, we check whether the current adjacent nodes to $\square_v$ are balanced. Specifically, let c be one of $\square_v$'s neighboring cells in the grid of $\square_v$, and let c_p be the square containing c in a grid one level up. We compute $\text{id}(c), \text{id}(c_p)$, and check if there is a node in T with those IDs. If not, we create a node w with region c_p and $\text{id}(c_p)$, and recursively retrieve its parent (i.e., if it exists we retrieve it, otherwise, we create it), and hang w from the parent node. We credit the work involved in creating w to the output size. We add all the new nodes to the queue. We repeat the process till the queue is empty.

Since the algorithm never creates nodes smaller than the smallest cell in the original compressed quadtree, it follows that this algorithm terminates. It is also easy to argue by induction that any balanced quadtree of P must contain all the nodes we created. Overall, the running time of the algorithm is $O(n\log n + m)$, since the work associated with any newly created quadtree node is constant. ■

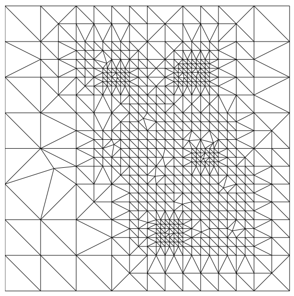


Figure 2.2: A well balanced triangulation.

Definition 2.4.4 The *extended cluster* of a cell c in a quadtree T is the set of 5×5 neighboring cells of c in the grid containing c, which are all the cells in distance $< 2l$ from c, where l is the sidelength of c.

A quadtree T over a point set P is *well-balanced*, if it is balanced, and for every leaf node v that contains a (single) point of P, we have the property that all the nodes of the extended cluster of v are *leaves* in T (i.e., none of them is split and has children), and they do not contain any other point of P. In fact, we will also require that for every non-empty node v, all the nodes of the extended cluster of v are nodes in the quadtree.

Lemma 2.4.5 *Given a point set P of n points in the plane, one can compute a well-balanced quadtree of P in $O(n \log n + m)$ time, where m is the size of the output quadtree.*

Proof: We compute a balanced quadtree T of P. Next, for every leaf node v of T which contains a point of P, we verify that all its extended cluster are leaves of T. If any other of the nodes of the extended cluster of v contains a point of P, we split v. If any of the extended cluster nodes is missing as a leaf, we insert it into the quadtree (with its ancestors if necessary). We repeat this process till we stop. Of course, during this process, we keep the balanced property valid, by adding necessary nodes. Clearly, all this work can be charged to newly created nodes, and as such takes linear time in the output size once the compressed quadtree is computed. ■

A well-balanced quadtree T of P provides for every point, a region (i.e., extended cluster) where it is well protected from other points. It is now possible to turn the partition of the plane induced by the leaves of T into a triangulation of P.

We "warp" the quadtree framework as follows. Let y be the corner nearest x of the leaf of T containing x; we replace y by x as a corner of the quadtree. Finally, we triangulate the resulting planar subdivision. Unwarped boxes are triangulated with isosceles right triangles by adding a point in the center. Only boxes with unsplit sides have warped corners; for these we choose the diagonal that gives better aspect ratio. Figure 2.2 shows a triangulation resulting from a variant of this method.

Lemma 2.4.6 *The method above gives a triangulation $\mathcal{QT}(P)$ with $\mathcal{A}_{\text{ratio}}(\mathcal{QT}(P)) \leq 4$.*

Proof: The right triangles used to triangulate the unwarped cells have aspect ratio 2. If a cell with side length l is warped, we have two cases.

In the first case, the input point of P is inside the square of the original cell. Then we assume that the diagonal touching the warped point is chosen; otherwise, the aspect ratio can only be better than what we

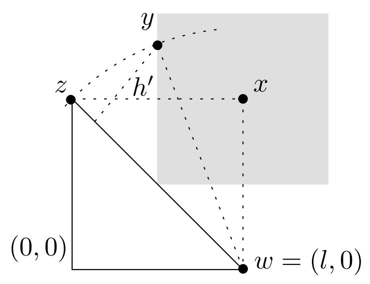


Figure 2.3: Illustration of the proof of Lemma 2.4.6.

prove. Consider one of the two triangles formed, with corners the input point and two other cell corners. The maximum length hypotenuse is formed when the warped point is on its original location, and has length $h = \sqrt{2}l$. The minimum area is formed when the point is in the center of the square, and has area $a = l^2/4$. Thus, the minimum height of such a triangle $\triangle$ is $\geq 2a/h$, and $\mathcal{A}_{\text{ratio}}(\triangle) \leq h/(2a/h) = h^2/2a = 4$.

In the second case, the input point is outside the original square. Since the quadtree is well balanced, the new point y is somewhere inside a square of sidelength l centered at x (since we always move the closest leaf corner to the new point). In this case, we assume that the diagonal not touching the warped point is chosen. This divides the cell into an isosceles right triangle and another triangle. If the chosen diagonal is the longest edge of the other triangle, then one can argue as before, and the aspect ratio is bounded by 4. Otherwise, the longest edge touches the input point. The altitude is minimized when the triangle is isosceles with as sharp an angle as possible; see Figure 2.3. Using the notation of Figure 2.3, we have $y = (l/2, \sqrt{7}l/2)$. Thus,

$$\mu = \text{area}(\triangle wyz) = \frac{1}{2}\begin{vmatrix} 1 & 0 & l \\ 1 & l & 0 \\ 1 & l/2 & (\sqrt{7}/2)l \end{vmatrix} = \frac{1}{2}\begin{vmatrix} l & -l \\ l/2 & (\sqrt{7}/2-1)l \end{vmatrix} = \frac{\sqrt{7}-1}{4}l^2.$$

We have $h\sqrt{2}l/2 = \mu$, and thus $h' = \sqrt{2}\mu/l = \frac{\sqrt{7}-1}{2\sqrt{2}}l$. The longest distance y can be from w is $\alpha = \sqrt{(1/2)^2 + (3/2)^2}l = (\sqrt{10}/2)l$. Thus, the aspect ratio of the new triangle is bounded by $\alpha/h' = \left(\sqrt{10}/2\right)/\frac{\sqrt{7}-1}{2\sqrt{2}} \approx 2.717 \leq 4$. ■

For a triangulation $\mathcal{T}$, let $|\mathcal{T}|$ denote the number of triangles of $\mathcal{T}$. The *Delaunay triangulation* of a point set is the triangulation formed by all triangles defined by the points such that their circumscribing triangles are empty (the fact that this collection of triangles forms a triangulation requires a proof). Delaunay triangulations are extremely useful, and have a lot of useful properties. We denote by $\mathcal{DT}(P)$ the Delaunay triangulation of P.

Lemma 2.4.7 *There is a constant c', independent of P, such that $|\mathcal{QT}(P)| \leq c' \sum_{\triangle \in \mathcal{DT}(P)} \log E_{\text{ratio}}(\triangle)$.*

Proof: For this lemma, we modify the description of our algorithm for computing $\mathcal{QT}(P)$. We compute the compressed quadtree T'' of P, and we uncompress the edges by inserting missing cells. Next, we split a leaf of T'' if it has side length $\varkappa$, it is not empty (i.e., it contains a point of P), and there is another point of P of distance $\leq 2\varkappa$ from it. We refer to such a node as being *crowded*. We repeat this, till there are no crowded leaves. Let T' denote the resulting quadtree. We now iterate over all the nodes v of T', and insert all the nodes of the extended cluster of v into T'. Let T denote the resulting quadtree. It is easy to verify that T is well-balanced, and identical to the quadtree generated by the algorithm of Lemma 2.4.5 (although it is unclear how to implement the algorithm described here efficiently).

Now, all the nodes of T that were created when adding the extended cluster nodes can be charged to nodes of T'. Therefore we need only count the total number of crowded cells in T'.

Linearly many crowded cells have more than one child with points in them. It can happen at most linearly many times that a non-empty cell c has a point of P outside it of distance $2\varkappa$ from it, which in the next level is in a cell non-adjacent to the children of c, where $\varkappa$ is the side length of the cell, as this point becomes further away due to the shrinking sizes of cells as they split.

If a cell b containing a point is split because an extended neighbor was split, but no extended neighbor contains any point, then, when either b or b's parent was split, a nearby point became farther away than $2\varkappa$. Again, this can only happen linearly many times.

Finally a cell may contain two points, or several extended neighbor cells may contain points, and this situation may persist when the cells split. If splitting the children of the cell or of its neighbors separates the points, we can charge linear total work. Otherwise, let Y be a maximal set of points in the union of cell b and its neighbors, such that splitting b, its neighbors, or the children of b and its neighbors does not further divide Y . Then some triangle of $\mathcal{DT}(P)$ connects two points y_1 and y_2 in Y with a point z outside Y.[②]

Each split not yet accounted for occurs between the step when Y is separated from z, and the step when y_1 and y_2 become more than $2\varkappa$ units apart. These steps are at most $O(\log E_{\text{ratio}}(\triangle y_1 y_2 z))$ quadtree levels apart, so we can charge all the crowded cells caused by Y to $\triangle y_1 y_2 z$. This triangle will not be charged by any other cells, because once we perform the splits charged to it all three points become far away from each other in the quadtree.

Therefore the number of crowded cells can be counted as a linear term, plus terms of the form $O(\log E_{\text{ratio}}(\triangle abc))$ for some Delaunay triangles $\triangle abc$. ■

Theorem 2.4.8 *Given any point set P, we can find a triangulation $\mathcal{QT}(P)$ such that each point of P is a vertex of $\mathcal{QT}(P)$ and $\mathcal{A}_{\text{ratio}}(\mathcal{QT}(P)) \leq 4$. There is a constant c'', independent of P, such that if $\mathcal{T}$ is any triangulation containing the points of P as vertices, $|\mathcal{QT}(P)| \leq c'' |\mathcal{T}| \log \mathcal{A}_{\text{ratio}}(\mathcal{T})$.*

In particular, any triangulation with constant aspect ratio containing P is of size $\Omega(\mathcal{QT}(P))$. Thus, up to a constant, $\mathcal{QT}(P)$ is an optimal triangulation.

Proof: Let Y be the set of vertices of $\mathcal{T}$. Lemma 2.4.7 states that there is a constant c such that $|\mathcal{QT}(Y)| \leq c \sum_{\triangle \in \mathcal{DT}(Y)} \log E_{\text{ratio}}(\triangle)$. The Delaunay triangulation has the property that it maximizes the minimum angle of the triangulation, among all triangulations of the point set [For97].

If $Y = P$, then using this maxmin̲angle property, we have $\mathcal{A}_{\text{ratio}}(\mathcal{T}) \geq \frac{1}{2}\mathcal{A}_{\text{ratio}}(\mathcal{DT}(P)) \geq \frac{1}{2}E_{\text{ratio}}(\mathcal{DT}(P))$, by Lemma 2.4.1. Hence

$$|\mathcal{QT}(P)| \leq c \sum_{\triangle \in \mathcal{DT}(P)} \log E_{\text{ratio}}(\mathcal{DT}(P)) = c\,|\mathcal{T}|\, E_{\text{ratio}}(\mathcal{DT}(P)) \leq 2c\,|\mathcal{T}|\, \mathcal{A}_{\text{ratio}}(\mathcal{T}).$$

Otherwise, $P \subset Y$. Imagine running our algorithm on point set Y, and observe that $|\mathcal{QT}(P)| \leq |\mathcal{QT}(Y)|$. By the same argument as above, $|\mathcal{QT}(Y)| \leq c\,|\mathcal{T}| \log \mathcal{A}_{\text{ratio}}(\mathcal{T})$. ■

Corollary 2.4.9 $|\mathcal{QT}(P)| = O(nlog\mathcal{A}_{\text{ratio}}(\mathcal{DT}(P)))$.

Corollary 2.4.9 is tight, as can be easily verified.

2.5 Bibliographical notes

The authoritative text on quadtrees is the book by Samet [Sam89]. The idea of using hashing in quadtrees in a variant of an idea due to Van Emde Boas, and is also used in performing fast lookup in IP routing (using PATRICIA tries which are one dimensional quadtrees [WVTP97]), among a lot of other applications.

[②]To see that, observe that there must be an edge connecting a point $y_1 \in Y$ with a point $z \in P \setminus Y$ (since the triangulation is connected). Next, by going around y_1 and the points it is connected to, it is easy to observe that since Y diameter is (considerably) smaller then the distances between y_1 and z, there must be an edge between y_1 and another point y_2 of Y (for example, take y_2 to be the closest point in Y to y_1). This edge, together with the edge before it in the ordering around y_1, form the required triangle.

The algorithm described for the efficient construction of compressed quadtrees, is as far as I know new. The classical algorithms for computing compressed quadtrees efficiently achieve the same running time, but require considerably more careful implementation, and paying careful attention to details [CK95, AMN$^+$98]. The idea of fingering a quadtree is from [AMN$^+$98] (although their presentation is different than ours).

The elegant skip-quadtree is from the recent work of Eppstein *et al.* [EGS05].

Balanced quadtree and good triangulations are due to Bern *et al.* [BEG94], and our presentation closely follows theirs. The problem of generating good triangulations had received considerable attention recently, as it is central to the problem of generating good meshes, which in turn are important for efficient numerical simulations of physical processes. The main technique used in generating good triangulations is the method of Delaunay refinement. Here, one computes the Delaunay triangulation of the point set, and inserts circumscribed centers as new points, for "bad" triangles. Proving that this method converges and generates optimal triangulations is a non-trivial undertaking, and is due to Ruppert [Rup93]. Extending it to higher dimensions, and handling boundary conditions make it even more challenging. However, in practice, the Delaunay refinement method outperforms the (more elegant and simpler to analyze) method of Bern *et al.* [BEG94], which easily extends to higher dimensions. Namely, the Delaunay refinement method generates good meshes with fewer triangles.

Furthermore, Delaunay refinement methods are slower in theory. Getting an algorithm to perform Delaunay refinement in the same time as the algorithm of Bern *et al.* is still open, although Miller [Mil04] got an algorithm with only slightly slower running time.

Very recently, Alper Üngör came up with a "Delaunay-refinement type" algorithm, which outputs better meshes than the classical Delaunay refinement algorithm [Üng04]. Furthermore, by merging the quadtree approach with Üngör technique, one can get an optimal running time algorithm [HÜ05].

2.6 Exercises

Exercise 2.6.1 [5 Points]A triangle $\triangle$ is called α-fat if each one of its angles is at least α, where $\alpha > 0$ is a prespecified constant (for example, α is 5 degrees). Let P be a triangular planar map of the unit square (i.e., each face is a triangle), where all the triangles are fat, and the total number of triangles is n. Prove that the complexity of the quadtree constructed for P is $O(n)$.

Exercise 2.6.2 [5 Points] Prove that the bounds of Lemma 2.2.2 are tight. Namely, show that for any $r > 2$ and any positive integer $n > 2$, there exists a set of n points with diameter $\Omega(1)$ and spread $\Phi(P) = \Theta(r)$, and such that its quadtree has size $\Omega(n \log \Phi(P))$.

Acknowledgments

The author wishes to thank John Fischer for his detailed comments on the manuscript.

Chapter 3

Clustering - k-center clustering

Do not read this story; turn the page quickly. The story may upset you. Anyhow, you probably know it already. It is a very disturbing story. Everyone knows it. The glory and the crime of Commander Suzdal have been told in a thousand different ways. Don't let yourself realize that the story is the truth.

It isn't. not at all. There's not a bit of truth to it. There is no such planet as Arachosia, no such people as klopts, no such world as Catland. These are all just imaginary, they didn't happen, forget about it, go away and read something else.

– The Crime and Glory of Commander Suzdal, Cordwainer Smith

In this chapter, we will initiate our discussion of *clustering*. Clustering is one of the most fundamental computational tasks, but frustratingly, one of fuzziest. It can be stated informally as: "Given data, find interesting structure in the data. Go!"

The fuzziness arise naturally from the requirement that it would be "interesting", as this is not well defined and depends on human perception which is sometime impossible to quantify clearly. Similarly, what is "structure" is also open to debate. Nevertheless, clustering is inherent to many computational tasks like learning, searching and data-mining.

Empirical study of clustering concentrates on trying various measures for the clustering, and trying out various algorithms and heuristics to compute these clusterings. See bibliographical notes for some relevant references.

Here, we will concentrate on some well defined clustering tasks, including k-center clustering, k-median clustering, and k-means clustering.

3.1 Preliminaries

A clustering problem is usually defined by a set of items, and a distance function defined between these items. While these items might be points in $\mathbb{R}^d$ and the distance function is just the regular Euclidean distance, it is sometime beneficial to consider the more abstract setting of a general metric space.

Definition 3.1.1 A *metric space* is a pair $(\mathcal{X}, \mathbf{d})$ where $\mathcal{X}$ is a set and $\mathbf{d} : \mathcal{X} \times \mathcal{X} \rightarrow [0, \infty)$ is a *metric*, satisfying the following axioms: (i) $\mathbf{d}(x, y) = 0$ iff $x = y$, (ii) $\mathbf{d}(x, y) = \mathbf{d}(y, x)$, and (iii) $\mathbf{d}(x, y) + \mathbf{d}(y, z) \geq \mathbf{d}(x, z)$ (triangle inequality).

For example, $\mathbb{R}^2$ with the regular Euclidean distance is a metric space. In the following, we assume that we are given a *black-box access* to $\mathbf{d}$. Namely, given two points $\mathsf{p}, \mathsf{q} \in \mathcal{X}$, we assume that $\mathbf{d}(\mathsf{p}, \mathsf{q})$ can be computed in constant time.

Thus, given a set of centers $\mathbf{C}$, every point of $\mathbf{P}$ is assigned to its nearest neigbor in $\mathbf{C}$. All the points of $\mathbf{P}$ that are assigned to a center $\overline{c}$ form the *cluster* of $\overline{c}$, denoted by

$$\Pi(\mathbf{C}, \overline{c}) = \left\{ p \in \mathbf{P} \;\middle|\; \mathbf{d}(p, \overline{c}) \le \mathbf{d}(p, \mathbf{C}) \right\}.$$

Namely, the center set $\mathbf{C}$ partition $\mathbf{P}$ into clusters. This spacific scheme of partitioning points by assigning them to their closest center in a given center set is known as *Voronoi partitions*.

3.2 k-Center Clustering

In the *k-center clustering problem*, a set $\mathbf{P} \subseteq \mathcal{X}$ is provided together with a parameter k. We would like to find k points $X \subseteq \mathbf{P}$, such that the maximum distance of a point in $\mathbf{P}$ to the closest point in X is minimized.

As a concrete example, consider the set of points to be a set of cities in a country. Distances between cities represent the time it takes to travel from one point to another. We would like to build k hospitals and minimize the maximum time it takes a patient to arrive to a hospital.

Formally, given a set of centers $\mathbf{C}$, the k-center clustering *price* of clustering $\mathbf{P}$ by $\mathbf{C}$ is denoted by

$$\mathrm{r}_\infty^{\mathbf{C}}(\mathbf{P}) = \max_{\mathsf{p} \in \mathbf{P}} \mathbf{d}(\mathsf{p}, \mathbf{C}),$$

where $\mathbf{d}(\mathsf{p}, \mathbf{C}) = \min_{\overline{c} \in \mathbf{C}} \mathbf{d}(\mathsf{p}, \overline{c})$ denotes the *distance* of p to the set $\mathbf{C}$.

Formally, the *k-center problem* is to find a set $\mathbf{C}$ of k points, such that $\mathrm{r}_\infty^{\mathbf{C}}(\mathbf{P})$ is minimized; namely,

$$\mathrm{r}_\infty^{\mathrm{opt}}(\mathbf{P}, k) = \min_{\mathbf{C}, |\mathbf{C}|=k} \mathrm{r}_\infty^{\mathbf{C}}(\mathbf{P}).$$

We will denote the set of centers realizing the optimal clustering by C_{opt}.

Every point in a clsuter is in distance at most $\mathrm{r}_\infty^{\mathbf{C}}(\mathbf{P})$ from its respective center.

It is known that the k-center clustering is NP-Hard, and it is in fact hard to approximate within a factor of 1, 86 even in two dimensions. Surprisingly, there is a simple and elegant algorithm that achieves 2-approximation.

3.2.1 The Greedy Clustering Algorithm

The greedy algorithm **GreedyKCenter** starts by picking an arbitrary point $\overline{c}_1$ into $\mathbf{C}_1$. Next, we compute for every point $\mathsf{p} \in \mathbf{P}$ its distance $d_1[\mathsf{p}]$ from $\overline{c}_1$. Next, consider the point worst served by $\overline{c}_1$; this is the point realizing $r_1 = \max_{\mathsf{p} \in \mathbf{P}} d_1[\mathsf{p}]$. Let $\overline{c}_2$ denote this point, and add it to the set of centers $\mathbf{C}_2$.

Specifically, in the ith iteration, we compute for each point $\mathsf{p} \in \mathbf{P}$ the quantity $d_{i-1}[\mathsf{p}] = \min_{\overline{c} \in \mathbf{C}_{i-1}} \mathbf{d}(\mathsf{p}, \overline{c})$. We also compute the radius of the clustering

$$r_{i-1} = \max_{\mathsf{p} \in \mathbf{P}} d_{i-1}[\mathsf{p}] = \max_{\mathsf{p} \in \mathbf{P}} \mathbf{d}(\mathsf{p}, \mathbf{C}_{i-1}), \tag{3.1}$$

and the bottleneck point $\overline{c}_i$ that realizes it. Next, we add $\overline{c}_i$ to $\mathbf{C}_{i-1}$ to form the new set $\mathbf{C}_i$. We repeat this process k times.

To make this algorithm slightly faster, observe that

$$d_i[\mathsf{p}] = \mathbf{d}(\mathsf{p}, \mathbf{C}_i) = \min(\mathbf{d}(\mathsf{p}, \mathbf{C}_{i-1}), \mathbf{d}(\mathsf{p}, \overline{c}_i)) = \min(d_{i-1}[\mathsf{p}], \mathbf{d}(\mathsf{p}, \overline{c}_i)).$$

In particular, if we maintain for p a single variable $d[\mathsf{p}]$ with its current distance to the closest center in the current center set, then the above formula boils down to

$$d[\mathsf{p}] \leftarrow \min(d[\mathsf{p}], \mathbf{d}(\mathsf{p}, \overline{c}_i)).$$

Namely, the above algorithm can be implemented using $O(n)$ space, where $n = |\mathbf{P}|$. The ith iteration of choosing the ith center takes $O(n)$ time. Thus, overall this approximation algorithm takes $O(nk)$ time.

Theorem 3.2.1 *Given a set of n points $\mathbf{P} \subseteq \mathcal{X}$, belonging to a metric space $(\mathcal{X}, \mathbf{d})$, the algorithm* **GreedyKCenter** *computes a set $\mathbf{K}$ of k centers, such that $\mathbf{K}$ is a 2-approximation to the optimal k-center clustering of $\mathbf{P}$; namely, $\mathrm{r}_\infty^{\mathbf{K}}(\mathbf{P}) \le 2\mathrm{r}_\infty^{\mathrm{opt}}(\mathbf{P}, k)$. The algorithm takes $O(nk)$ time.*

Proof: The running time follows by the above description, so we concern ourselves only with the approximation quality.

If every cluster of C_{opt} contains exactly one point of $\mathbf{K}$ then the claim follows. Indeed, consider a point $\mathsf{p} \in \mathbf{P}$, and let $\overline{\mathrm{c}}$ be the center it belongs to in C_{opt}. Also, let $\overline{\mathrm{k}}$ be the center of $\mathbf{K}$ that is in $\Pi\left(C_{\mathrm{opt}}, \overline{\mathrm{c}}\right)$. We have that $\mathbf{d}(\mathsf{p}, \overline{\mathrm{c}}) = \mathbf{d}\left(\mathsf{p}, C_{\mathrm{opt}}\right) \le \mathrm{r}_\infty^{\mathrm{opt}} = \mathrm{r}_\infty^{\mathrm{opt}}(\mathbf{P}, k)$. Similarly, observe that $\mathbf{d}\left(\overline{\mathrm{k}}, \overline{\mathrm{c}}\right) = \mathbf{d}\left(\overline{\mathrm{k}}, C_{\mathrm{opt}}\right) \le \mathrm{r}_\infty^{\mathrm{opt}}$. As such, by the triangle inequality, we have that $\mathbf{d}\left(\mathsf{p}, \overline{\mathrm{k}}\right) \le \mathbf{d}(\mathsf{p}, \overline{\mathrm{c}}) + \mathbf{d}\left(\overline{\mathrm{c}}, \overline{\mathrm{k}}\right) \le 2\mathrm{r}_\infty^{\mathrm{opt}}$.

By the pigeon hole principle, the only other possibility is that there are two centers $\overline{\mathrm{k}}$ and $\overline{\mathrm{u}}$ of $\mathbf{K}$ that are both in $\Pi\left(C_{\mathrm{opt}}, \overline{\mathrm{c}}\right)$, for some $\overline{\mathrm{c}} \in C_{\mathrm{opt}}$. Assume, without loss of generality, that $\overline{\mathrm{u}}$ was added later to the center set $\mathbf{K}$ by the algorithm **GreedyKCenter**, say in the ith iteration. But then, since **GreedyKCenter** always chooses the point furthest away from the current set of centers, we have that $\overline{\mathrm{c}} \in \mathbf{C}_{i-1}$ and

$$\mathrm{r}_\infty^{\mathbf{K}}(\mathbf{P}) \le \mathrm{r}_\infty^{\mathbf{C}_{i-1}}(\mathbf{P}) = \mathbf{d}(\overline{\mathrm{u}}, \mathbf{C}_{i-1}) \le \mathbf{d}\left(\overline{\mathrm{u}}, \overline{\mathrm{k}}\right) \le \mathbf{d}(\overline{\mathrm{u}}, \overline{\mathrm{c}}) + \mathbf{d}\left(\overline{\mathrm{c}}, \overline{\mathrm{k}}\right) \le 2\mathrm{r}_\infty^{\mathrm{opt}}.$$

■

3.2.2 The greedy permutation

There is an interesting phenomena associated with **GreedyKCenter**. We can ran it till it exhausts all the points of $\mathbf{P}$ (i.e., $k = n$). Then, this algorithm generates a permutation of $\mathbf{P}$; that is $\mathbf{P} = \langle \overline{\mathrm{c}}_1, \overline{\mathrm{c}}_2, \ldots, \overline{\mathrm{c}}_n \rangle$. We will refer to $\mathbf{P}$ as the *greedy permutation* of $\mathbf{P}$. There is also an associated sequence of radiuses $\langle r_1, r_2, \ldots, r_n \rangle$, where all the points of $\mathbf{P}$ are in distance at most r_i from the points of $\mathbf{C}_i = \langle \overline{\mathrm{c}}_1, \ldots, \overline{\mathrm{c}}_i \rangle$.

Definition 3.2.2 A set $S \subseteq \mathbf{P}$ is a *r-net* for $\mathbf{P}$ if the following two properties hold.

(i) *Covering property*: All the points of $\mathbf{P}$ are in distance at most r from the points of $\mathbf{P}$.

(ii) *Separation property*: For any pair point of points $\mathsf{p}, \mathsf{q} \in S$, we have that $\mathbf{d}(\mathsf{p}, \mathsf{q}) \ge r$.

(One can relax the separation property by requiring that the points of S would be at distance $\Omega(r)$ apart.)

Intuitively, a r-net of a point-set $\mathbf{P}$ is a compact representation of $\mathbf{P}$ in the resolution r. Surprisingly, the greedy permutation of $\mathbf{P}$ provides us with such a representation for all resolutions.

Theorem 3.2.3 *Let $\mathbf{P}$ be a set of n points in a finite metric space, and let its greedy permutation be $\langle \overline{\mathrm{c}}_1, \overline{\mathrm{c}}_2, \ldots, \overline{\mathrm{c}}_n \rangle$ with the associate sequence of radiuses $\langle r_1, r_2, \ldots, r_n \rangle$. For any i, we have that $\mathbf{C}_i = \langle \overline{\mathrm{c}}_1, \ldots, \overline{\mathrm{c}}_i \rangle$ is a r_i-net of $\mathbf{P}$.*

Proof: Note, that by construction $r_k = \mathbf{d}(\overline{\mathrm{c}}_k, \mathbf{C}_{k-1})$, for all $k = 1, \ldots, n$. As such, for $j < k \le n$, we have that $\mathbf{d}(\overline{\mathrm{c}}_j, \overline{\mathrm{c}}_k) \ge r_k$, which implies the required separation property. The covering property follows by definition, see Eq. (3.1). ■

3.3 k-median clustering

In the *k-median clustering problem*, a set $\mathbf{P} \subseteq \mathcal{X}$ is provided together with a parameter k. We would like to find k points $\mathbf{C} \subseteq \mathbf{P}$, such that the sum of distances of points of $\mathbf{P}$ to their closest point in $\mathbf{C}$ is minimized.

Formally, given a set of centers $\mathbf{C}$, the k-center clustering *price* of clustering $\mathbf{P}$ by $\mathbf{C}$ is denoted by

$$\mathrm{r}_1^{\mathbf{C}}(\mathbf{P}) = \max_{\mathsf{p} \in \mathbf{P}} \mathbf{d}(\mathsf{p}, \mathbf{C}),$$

where $\mathbf{d}(\mathsf{p}, \mathbf{C}) = \min_{\overline{c} \in \mathbf{C}} \mathbf{d}(\mathsf{p}, \overline{c})$ denotes the *distance* of p to the set $\mathbf{C}$.

Formally, the *k-center problem* is to find a set $\mathbf{C}$ of k points, such that $\mathrm{r}_1^{\mathbf{C}}(\mathbf{P})$ is minimized; namely,

$$\mathrm{r}_1^{\mathrm{opt}}(\mathbf{P}, k) = \min_{\mathbf{C}, |\mathbf{C}|=k} \mathrm{r}_1^{\mathbf{C}}(\mathbf{P}).$$

We will denote the set of centers realizing the optimal clustering by C_{opt}.

There is a simple and elegant constant factor approximation algorithm for k-median clustering using local search.

A note on notations. Consider the set U of all k-tuples of points of $\mathbf{P}$. Let p_i denote the ith point of $\mathbf{P}$, for $i = 1, \ldots, n$, where $n = |\mathbf{P}|$. for $\mathbf{C} \in U$, consider the n dimensional point

$$\phi(\mathbf{C}) = (\mathbf{d}(p_1, \mathbf{C}), \mathbf{d}(p_2, \mathbf{C}), \ldots, \mathbf{d}(p_n, \mathbf{C})).$$

Clearly, we have that $\mathrm{r}_\infty^{\mathbf{C}}(\mathbf{P}) = \|\phi(\mathbf{C})\|_\infty = \max_i \mathbf{d}(p_i, \mathbf{C})$. And $\mathrm{r}_\infty^{\mathrm{opt}}(\mathbf{P}, k) = \min_{\mathbf{C} \in U} \|\phi(\mathbf{C})\|_\infty$.

Similarly, we have that $\mathrm{r}_1^{\mathbf{C}}(\mathbf{P}) = \|\phi(\mathbf{C})\|_1 = \sum_i \mathbf{d}(p_i, \mathbf{C})$. And $\mathrm{r}_1^{\mathrm{opt}}(\mathbf{P}, k) = \min_{\mathbf{C} \in U} \|\phi(\mathbf{C})\|_1$.

Namely, k-center clustering under this interpeation is just finding the point minimizing the ℓ_∞ norm in a set of points in n dimensions. Similarly, the k-median problem is to find the poitn minizing the norm under the ℓ_1 norm. Since ℓ_1 and ℓ_∞ are equal up to a factor equal to the dimension, we get the following.

Observation 3.3.1 *For any point set* $\mathbf{P}$ *of n points and a parameter k, we have that* $\mathrm{r}_\infty^{\mathrm{opt}}(\mathbf{P}, k) \leq \mathrm{r}_1^{\mathrm{opt}}(\mathbf{P}, k) \leq n \cdot \mathrm{r}_\infty^{\mathrm{opt}}(\mathbf{P}, k)$.

3.3.1 Local Search

A $2n$-approximation. Observation 3.3.1 implies that if we compute a set of centers $\mathbf{C}$ using Theorem 3.2.1 then we have that

$$\mathrm{r}_1^{\mathbf{C}}(\mathbf{P})/2n \leq \mathrm{r}_\infty^{\mathbf{C}}(\mathbf{P})/2 \leq \mathrm{r}_\infty^{\mathrm{opt}}(\mathbf{P}, k) \leq \mathrm{r}_1^{\mathrm{opt}}(\mathbf{P}, k) \leq \mathrm{r}_1^{\mathbf{C}}(\mathbf{P}) \quad \Rightarrow \quad \mathrm{r}_1^{\mathbf{C}}(\mathbf{P}) \leq 2n\mathrm{r}_1^{\mathrm{opt}}(\mathbf{P}, k). \tag{3.2}$$

Namely, $\mathbf{C}$ is a $2n$-approximation to the optimal solution.

Improving it. Let $0 < \tau < 1$ be a parameter to be determined shortly. The local search algorithm **AlgLocalSearchKMed** initially sets the current set of centers $\mathbf{C}_{\mathrm{curr}}$ to be $\mathbf{C}$. Next, at each iteration it checks if the current solution $\mathbf{C}_{\mathrm{curr}}$ can be improved by replacing one of the centers in it by a center from the outside. There are at most $|\mathbf{P}| \cdot |\mathbf{C}_{\mathrm{curr}}| = nk$ choices to consider, as we pick a center $\overline{c} \in \mathbf{C}_{\mathrm{curr}}$ to throw away and a new center to replace it by $\overline{e} \in (\mathbf{P} \setminus \mathbf{C}_{\mathrm{curr}})$. We consider the new candidate set of centers $\mathbf{K} \leftarrow (\mathbf{C}_{\mathrm{curr}} \setminus \{\overline{c}\}) \cup \{\overline{e}\}$. If $\mathrm{r}_1^{\mathbf{K}}(\mathbf{P}) \leq (1-\tau)\mathrm{r}_1^{\mathbf{C}_{\mathrm{curr}}}(\mathbf{P})$ then the algorithm sets $\mathbf{C}_{\mathrm{curr}} \leftarrow \mathbf{K}$ and repeats.

The algorithm **AlgLocalSearchKMed** stops when there is no exchange that would improve the current solution by a factor of (at least) $(1-\tau)$. The final content of the set $\mathbf{C}_{\mathrm{curr}}$ is the required constant factor approximation. Note, that the running time of the algorithm is

$$O\left((nk)^2 \log_{1/(1-\tau)} \frac{\mathrm{r}_1^{\mathbf{C}}(\mathbf{P})}{\mathrm{r}_1^{\mathrm{opt}}(\mathbf{P}, k)}\right) = O\Big((nk)^2 \log_{1+\tau}(2n)\Big) = O\left((nk)^2 \frac{\log n}{\ln(1+\tau)}\right) = O\left((nk)^2 \frac{\log n}{\tau}\right),$$

by (**??**) and since $1/(1-\tau) \geq 1 + \tau$. The final step follows since $1 + \tau \leq \exp(\tau) \leq 1 + 2\tau$, for $\tau < 1/2$ as can be easily verified. Thus, if τ is polynomially small, then the running time would be polynomial.

3.3.2 Proof

The proof of correctness is quite involved and the reader might want to skip it for now.

For the sake of simplicity of exposition, let us assume that the solution returned by the algorithm can not be improved by any swapm, and let $\mathbf{C}$ be this set of cetners. For a center $\overline{c} \in \mathbf{C}$ and $\overline{e} \in \mathbf{P} \setminus \mathbf{C}$ let $\mathbf{C} - \overline{c} + \overline{e}$ denote the set of centers resulting from apply the swap $\overline{c} \rightarrow \overline{e}$ to

$\mathbf{C}$; namely, $\mathbf{C} - \overline{c} + \overline{e} = (\mathbf{C} \setminus \{\overline{c}\}) \cup \{\overline{e}\}$. We are assuming that

$$\forall \overline{c} \in \mathbf{C}, \overline{e} \in \mathbf{P} \setminus \mathbf{C} \qquad \rho(\mathbf{C} - \overline{c} + \overline{e}) \geq \rho(\mathbf{C}),$$

where $\rho(\mathbf{C}) = r_1^{\mathbf{C}}(\mathbf{P})$. The contribution of a point $p \in \mathbf{P}$ to the quantity $\Delta(\overline{c}, \overline{e}) = \rho(\mathbf{C} - \overline{c} + \overline{e}) - \rho(\mathbf{C})$ is $\delta_p = \mathbf{d}(p, \mathbf{C} - \overline{c} + \overline{e}) - \mathbf{d}(p, \mathbf{C})$. Clealry, if p is served by the same center in $\mathbf{C}$ and in $\mathbf{C} - \overline{c} + \overline{e}$ then $\delta_p = 0$. In particular, we have that $\forall p \in \mathbf{P} \setminus \Pi(\mathbf{C}, \overline{c})$ it holds $\delta_p \leq 0$. Thus,

$$\forall \overline{c} \in \mathbf{C}, \overline{e} \in C_{\mathrm{opt}} \quad 0 \leq \Delta(\overline{c}, \overline{e}) \leq \sum_{p \in \Pi(\mathbf{C}, \overline{c}) \cup \Pi(C_{\mathrm{opt}}, \overline{e})} \delta_p = \sum_{p \in \Pi(C_{\mathrm{opt}}, \overline{e})} \delta_p + \sum_{p \in \Pi(\mathbf{C}, \overline{c}) \setminus \Pi(C_{\mathrm{opt}}, \overline{e})} \delta_p \tag{3.3}$$

where $\overline{e} \in C_{\mathrm{opt}}$ and C_{opt} is the optimal set of centers. What Eq. (3.3) gives us is a large family of inequaliteis that all of them hold together. Each inequality is represented by a swap $\overline{c} \rightarrow \overline{e}$. We would like to pick a set of swaps such that these inequalities when added together, would imply that

$$5 r_1^{C_{\mathrm{opt}}}(\mathbf{P}) - r_1^{\mathbf{C}}(\mathbf{P}) \geq 0.$$

This would imply that $5 r_1^{C_{\mathrm{opt}}}(\mathbf{P}) \geq r_1^{\mathbf{C}}(\mathbf{P})$; namely, that the local search algorithm provides a constant factor approximation to optimal clustering. This idea seems to be somewhat mysterious, but to see that there is indeed hope to achieve that, observe that if our set of swaps T has each center of C_{opt} appearing in it exactly once, then when we add up the first term on the right side of Eq. (3.3) then we have that

$$\begin{aligned}
\sum_{\overline{c} \rightarrow \overline{e} \in T} \left(\sum_{p \in \Pi(C_{\mathrm{opt}}, \overline{e})} \delta_p \right) &= \sum_{\overline{c} \rightarrow \overline{e} \in T} \left(\sum_{p \in \Pi(C_{\mathrm{opt}}, \overline{e})} (\mathbf{d}(p, \mathbf{C} - \overline{c} + \overline{e}) - \mathbf{d}(p, \mathbf{C})) \right) \\
&\leq \sum_{\overline{c} \rightarrow \overline{e} \in T} \left(\sum_{p \in \Pi(C_{\mathrm{opt}}, \overline{e})} \Big(\mathbf{d}(p, C_{\mathrm{opt}}) - \mathbf{d}(p, \mathbf{C})\Big) \right) = \sum_{p \in \mathbf{P}} \Big(\mathbf{d}(p, C_{\mathrm{opt}}) - \mathbf{d}(p, \mathbf{C})\Big) \\
&= r_1^{C_{\mathrm{opt}}}(\mathbf{P}) - r_1^{\mathbf{C}}(\mathbf{P}),
\end{aligned}$$

since $\overline{e}$ ranges over the elements of C_{opt} exactly once, and for $\overline{e} \in C_{\mathrm{opt}}$ and $p \in \Pi\left(C_{\mathrm{opt}}, \overline{e}\right)$ we have that $\mathbf{d}(p, \mathbf{C} - \overline{c} + \overline{e}) \leq \mathbf{d}(p, \overline{e}) = \mathbf{d}(p, C_{\mathrm{opt}})$. Thus, we can bound the first term of the right side of Eq. (3.3) as required. We thus turn our attention to bounding the second term.

Namely,

$$0 \leq \rho(\mathbf{C} - \overline{c} + \overline{e}) - \rho(\mathbf{C}) = \sum_{p \in \mathbf{P}} (\mathbf{d}(p, \mathbf{C} - \overline{c} + \overline{e}) - \mathbf{d}(p, \mathbf{C})) = \sum_{p \in \Pi(\mathbf{C}, \overline{c}) \cup \Pi(\mathbf{C} - \overline{c} + \overline{e}, \overline{e})} (\mathbf{d}(p, \mathbf{C} - \overline{c} + \overline{e}) - \mathbf{d}(p, \mathbf{C}))$$

Notice, that the change can be upper bounded as follows: (i) The points the only points that have different contribution to $\rho(\mathbf{C} - \overline{\mathsf{c}} + \mathsf{p})$ and $\rho(\mathbf{C})$ are the one that are in $\Pi\left(\mathbf{C}, \overline{\mathsf{c}}\right)$ and in *uter*

$\Pi_{\mathbf{C}}^{\overline{\mathsf{c}}}$

$[\mathbf{C}, c]$ $\mathbf{P}/\mathbf{C}$ $P_C(c)$

XXXXXXXXXXXX

3.4 *k*-means clustering

3.4.1 Local Search

3.4.2 The *k*-means method

3.5 Bibliographical Notes

In this chapter we introduced the problem of clustering and showed some algorithms that achieve constant factor approximation. A lot more is known about these problems including faster and better clustering algorithms but to discuss them we need more advanced tools than what we currently have at hand.

***k*-center Clustering.** The algorithm **GreedyKCenter** was described by Gonzalez [Gon85], but it was probably known before, as the notion of r-net is much older. The hardness of approximating k-center clustering was shown by Feder and Greene [FG88].

Chapter 4

Well Separated Pairs Decomposition

4.1 WSPD

Let P be a set of n points in $\mathbb{R}^d$, and $1/4 > \varepsilon > 0$ a parameter. Denote by $A \otimes B = \left\{ \{x, y\} \,\middle|\, x \in A,\ y \in B \right\}$. A *well-separated pair decomposition* (WSPD) with parameter ε^{-1} of P is a set of pairs $\{\{A_1, B_1\}, \ldots, \{A_s, B_s\}\}$, such that

1. $A_i, B_i \subset P$ for every i.
2. $A_i \cap B_i = \emptyset$ for every i.
3. $\cup_{i=1}^{s} A_i \otimes B_i = P \otimes P$.
4. $\mathbf{d}(A_i, B_i) \geq \varepsilon^{-1} \cdot \max\{\mathrm{diam}(A_i), \mathrm{diam}(B_i)\}$, where $\mathbf{d}(A_i, B_i) = \min_{p \in A_i, q \in B_i} \|p - q\|$.

Instead of maintaining such a decomposition explicitly, it is convenient to construct a compressed quadtree T of the points of P, and every pair, (A_i, B_i) is just a pair of nodes (v_i, u_i) of T, such that $A_i = P_{v_i}$ and $B_i = P_{u_i}$, where P_v denote the points of P stored in the subtree of v, where v is a node of T. This gives us a compact representation of the distances.

We slightly modify the construction of the compressed quadtree, such that for every nodes $v \in T$, it also stores a *representative* point rep_v, which is a point in P_v. Furthermore, rep_v is one of the representative points of one of the children of v. This can be easily computed in linear time, once the compressed quadtree is computed.

Before presenting the algorithm for computing WSPD, we remind the reader that $\ell(v)$ is $\lg(\varkappa(\square_v))$, where $\square_v$ is the cell (i.e., square, cube or hypercube depending on the dimension) which is the region that the node v corresponds to, and $\varkappa(\square_v)$ is the sidelength of $\square_v$. Since, the root node corresponds to the unit-square/hypercube, $\ell(v)$ is always a non-positive integer number.

For techincal reasons, we induce an aribtrary ordering of the nodes of the given quadtree. Let $\preceq$ denote this ordering.

We compute the compressed quadtree T of P in $O(n \log n)$ time. Next, we compute the WSPD

```
WSPD($u, v, T$)
  Assume $\ell(u) > \ell(v)$ or ( $\ell(u) = \ell(v)$ and $u \preceq v$)
      (otherwise exchange $u \leftrightarrow v$).
  If $8\sqrt{d} \cdot 2^{\ell(u)} \leq \varepsilon \cdot \left\|\mathrm{rep}_u - \mathrm{rep}_v\right\|$ then
    return $\{ \{u, v\} \}$
  else
    Denote by $u_1, \ldots, u_r$ the children of $u$
    return $\bigcup_{i=1}^{r}$ WSPD($u_i, v, T$).
End WSPD
```

Figure 4.1: The algorithm **WSPD** for computing well-separated pairs decomposition.

by calling **WSPD**(u_0, u_0), where u_0 is the root of T and the algorithm **WSPD** is defined recursively in Figure 4.1.

The following lemma follows by an easy packing argument.

Lemma 4.1.1 *Let $\square$ be a cell in a grid G_r in $\mathbb{R}^d$ with sidelength r. Then, the number of cells in G_r such that their distance small $\square$ is smaller than ρ, si smaller than $O\Big((\rho/r)^d\Big)$.*

Lemma 4.1.2 *The WSPD generated by* **WSPD** *is valid. Namely, for any pair $\{u, v\}$ in the WSPD, we have* $\mathrm{diam}(P_u), \mathrm{diam}(P_v) \le \varepsilon \left\|\mathrm{rep}_u - \mathrm{rep}_v\right\|$ *and* $\left\|\mathrm{rep}_u - \mathrm{rep}_v\right\| \le (1+\varepsilon)\left\|x - y\right\|$, *for any $x \in P_u$ and $y \in P_v$.*

Proof: Since T is a compressed quadtree, for any node $u \in T$, we have $\mathrm{diam}(P_u) \le \sqrt{d}2^{\ell(u)}$. In particular, for every output pair $\{u, v\}$,

$$\begin{aligned} \max\{\mathrm{diam}(P_u), \mathrm{diam}(P_v)\} &\le \sqrt{d} \cdot \max\Big\{2^{\ell(u)}, 2^{\ell(v)}\Big\} \le \tfrac{\varepsilon}{4}\left\|\mathrm{rep}_u - \mathrm{rep}_v\right\| \\ &\le \tfrac{\varepsilon}{4}(\mathbf{d}(P_u, P_v) + \mathrm{diam}(P_u) + \mathrm{diam}(P_v)), \end{aligned}$$

so $\max\{\mathrm{diam}(P_u), \mathrm{diam}(P_v)\} \le \frac{\varepsilon}{4(1-\varepsilon/2)}\mathbf{d}(P_u, P_v) \le \varepsilon\mathbf{d}(P_u, P_v)$, since $\varepsilon \le 1$. Similarly, for any $x \in P_u$ and $y \in P_v$, we have

$$\left\|\mathrm{rep}_u - \mathrm{rep}_v\right\| \le \left\|x - y\right\| + \mathrm{diam}(P_u) + \mathrm{diam}(P_v) \le (1+\varepsilon)\left\|x - y\right\|.$$

Finally, by induction, it follows that every pair of points is covered by a pair of subsets $\{P_u, P_v\}$ output by the **WSPD** algorithm. ■

Lemma 4.1.3 *The number of pairs in the computed WSPD is $O(n/\varepsilon^d)$.*

Proof: Let $\{u, v\}$ be an output pair and assume that the call to **WSPD**(u, v) was issued by **WSPD**(u, v'), where $v' = \overline{\mathrm{p}}(v)$ is the parent of v in T. We charge this call to v', and we will prove that each node is charged at most $O(\varepsilon^{-d})$ times.

So fix a node $v' \in V(T)$. It is charged by pairs of the form $\{u, v\}$ in which $\overline{\mathrm{p}}(v) = v'$, and which was issued inside **WSPD**(u, v'). This implies that $\ell(\overline{\mathrm{p}}(u)) \ge \ell(v') \ge \ell(u)$. Since the pair (u, v') was not generated by **WSPD** we conclude that $\left\|\mathrm{rep}_{v'} - \mathrm{rep}_u\right\| \le \rho$, where $\rho = 8\sqrt{d} \cdot 2^{\ell(v')}/\varepsilon$. There are three possible cases:

(i) $\ell(v') = \ell(u)$. But there are at most $O\left(\left(\rho/2^{\ell(v')}\right)^d\right) = O(1/\varepsilon^d)$ nodes that have the same level as v' and their cell are in distance at most ρ from it, by Lemma 4.1.1. Thus, this can happened at most $O(1/\varepsilon^d)$ times.

(ii) $\ell(\overline{\mathrm{p}}(u)) = \ell(v')$. By the same argumentation as at (i), there are at most $O(2^d/\varepsilon^d) = O(1/\varepsilon^d)$ nodes with this property, since every node has at most 2^d children.

(iii) $\ell(\overline{\mathrm{p}}(u)) > \ell(v') > \ell(u)$. Namely, the edge between u and $\overline{\mathrm{p}}(u)$ is a compressed edge in T. But then, consider the node τ in the *uncompressed* tree (on the path connecting u to $\overline{\mathrm{p}}(u)$) which has the same level as v' and its cell contains $\square_u$. We charge u to τ. Clealry, τ will get charged at most once, and we have that $\mathbf{d}(\square_{v'}, \square_\tau) \le \mathbf{d}(\square_{v'}, \square_u) \le \left\|\mathrm{rep}_{v'} - \mathrm{rep}_u\right\| \le \rho$. As such, arguing as above, we have that this happens at most $O(1/\varepsilon^d)$ times.

Thus, v' can be charged $O\Big(\varepsilon^{-d} \cdot |C_{v'}|\Big)$ times, where $C_{v'}$ is the set of children of v'. Since, $|C_{v'}| \le 2^d$, we conclude that v' might be charged at most $O(2^d\varepsilon^{-d}) = O(\varepsilon^{-d})$ times. Thus, the total number of pairs generated by the algorithm is $O(n\varepsilon^{-d})$. ■

Since the running time of **WSPD** is clearly linear in the output size, we have the following result.

Theorem 4.1.4 *For $1 \geq \varepsilon > 0$, one can construct a ε^{-1}-WSPD of size $n\varepsilon^{-d}$, and the construction time is $O\left(n \log n + n\varepsilon^{-d}\right)$.*

Furthermore, for any pair $\{u, v\}$ in the WSPD, we have $\operatorname{diam}(P_u), \operatorname{diam}(P_v) \leq \varepsilon \left\|\mathrm{rep}_u - \mathrm{rep}_v\right\|$ and $\left\|\mathrm{rep}_u - \mathrm{rep}_v\right\| \leq (1+\varepsilon)\left\|x - y\right\|$, for any $x \in P_u$ and $y \in P_v$.

4.2 Applications of WSPD

4.2.1 Spanners

A t-spanner of a set of points $P \subset \mathbb{R}^d$ is a weighted graph G whose vertices is the points of P, and for any $x, y \in P$,

$$\|xy\| \leq d_G(x,y) \leq t \cdot \|xy\|,$$

where d_G the metric of the shortest path on G, see [PS89].

Theorem 4.2.1 *Given an n-point set $P \subseteq \mathbb{R}^d$, and parameter $1 \geq \varepsilon > 0$, one can compute a $(1+\varepsilon)$-spanner of P with $O(n\varepsilon^{-d})$ edges, in $O\left(n \log n + n\varepsilon^{-d}\right)$ time.*

Proof: Let $c \geq 16$ be an arbitrary constant, and set $\delta = \varepsilon / c$. Compute a δ^{-1}-WSPD decomposition using the algorithm of Theorem 4.1.4. For every pair $\{u, v\} \in$ WSPD, add an edge between $\{\mathrm{rep}_u, \mathrm{rep}_v\}$ with weight $\left\|\mathrm{rep}_u \mathrm{rep}_v\right\|$. Let G be the resulting graph, clearly, the resulting shortest path metric d_G dominates $\|\cdot\|$.

The upper bound on the stretch is proved by induction on the length of pairs in the WSPD. Fix a pair $x, y \in P$, by our induction hypothesis, we have for every pair $z, w \in P$ such that $\|zw\| < \|xy\|$, it holds $\mathbf{d}_G(z, w) \leq (1 + c\delta)\|zw\|$.

The pair x, y must appear in some pair $\{u, v\} \in$ WSPD, where $x \in P_u$, and $y \in P_v$. Thus $\left\|\mathrm{rep}_u \mathrm{rep}_v\right\| \leq (1+\delta)\|xy\|$ and $\left\|\mathrm{rep}_u x\right\|, \left\|\mathrm{rep}_v y\right\| \leq \delta \left\|\mathrm{rep}_u \mathrm{rep}_v\right\|$, by Theorem 4.1.4. By the inductive hypothesis

$$\begin{aligned}
\|xy\| &\leq d_G(x,y) \leq d_G(x, \mathrm{rep}_u) + d_G(\mathrm{rep}_u, \mathrm{rep}_v) + d_G(\mathrm{rep}_v, y)\\
&\leq (1+c\delta)\left\|\mathrm{rep}_u x\right\| + \left\|\mathrm{rep}_u \mathrm{rep}_v\right\| + (1+c\delta)\left\|\mathrm{rep}_v y\right\|\\
&\leq 2(1+c\delta)\cdot\delta\cdot\left\|\mathrm{rep}_u \mathrm{rep}_v\right\| + \left\|\mathrm{rep}_u \mathrm{rep}_v\right\|\\
&\leq (1+2\delta+2c\delta^2)\left\|\mathrm{rep}_u \mathrm{rep}_v\right\| \leq (1+2\delta+2c\delta^2)(1+2\delta)\|xy\|\\
&\leq (1+4\delta)(1+\delta)\|xy\| \leq (1+5\delta+4\delta^2)\|xy\|\\
&\leq (1+c\delta)\|xy\| \leq (1+\varepsilon)\|xy\|,
\end{aligned}$$

since $\delta c \leq \varepsilon \leq 1$ and $16\delta \leq 1$ and $c \geq 11$. ■

4.2.2 Approximating the Minimum Spanning Tree and the Diameter of a Point Set

For a graph G, let $G_{\leq r}$ denote the subgraph of G, resulting from removing all edges of weight larger than r from G.

Lemma 4.2.2 *Given a set P of n points in $\mathbb{R}^d$, one can compute a spanning tree T of P, such that $w(T) \leq (1+\varepsilon)w(\mathcal{M})$, where $\mathcal{M}$ is the minimum spanning tree of P, and $w(T)$ is the total weight of the edges of T. This takes $O(n \log n + n\varepsilon^{-d})$ time.*

In fact, for any $r \geq 0$, we have that for any connected C component of $\mathcal{M}_{\leq r}$ is contained in a connected component of $T_{\leq(1+\varepsilon)r}$.

Proof: Compute a $(1+\varepsilon)$-spanner G of P. Let T be the minimum spanning tree of G. Clearly, T is the required $(1+\varepsilon)$-approximate MST. Indeed, for any $u, v \in P$, let $\pi_{u,v}$ denote the shortest path between u and v in G. We have that $G' = (P, \cup_{(u,v)\in\mathcal{M}} \pi_{u,v})$ is connected subgraph of G. Furthermore, $w(G') \leq (1+\varepsilon)w(\mathcal{M})$. It thus follows that $\mathcal{M}(G) \leq w(G') \leq (1+\varepsilon)w(\mathcal{M}(P))$.

The second claim follows by similar argumentation. ■

Lemma 4.2.3 *Given a set P of n points in $\mathbb{R}^d$, one can compute, in $O(n \log n + n\varepsilon^{-d})$ time, a pair $u, v \in P$, such that $\|uv\| \geq (1-\varepsilon)\operatorname{diam}(P)$.*

Proof: Compute a $(\varepsilon^{-1}/4)$-WSPD of P. Then for every pair in the WSPD, compute the distance of the representative point of every pair. Return the pair of representative farthest away from each other. ■

4.2.3 Closest Pair

Let P be a set of points in $\mathbb{R}^d$, and compute a ε^{-1}-WSPD $\mathcal{W}$ of P, for $\varepsilon = 1/4$. Consider the pair of closest points $p, q \in P$, and consider the pair $\{u, v\} \in \mathcal{W}$, such that $p \in P_u$ and $q \in P_v$. If P_u contains an additional point $r \in P_u$, then we have that $\|pr\| \leq \varepsilon(1+\varepsilon)\|uv\| \leq \|uv\|/2$, by Theorem 4.1.4 and since $\varepsilon = 1/4$. Thus, $\|pr\| < \|pq\|$, a contradiction to the choice of p, q as closest pair. Thus, $|P_u| = |P_v| = 1$ and $\mathrm{rep}_u = p$ and $\mathrm{rep}_v = q$. Thus, scan all the pairs of $\mathcal{W}$, and check for all the pairs which connect singletons, what is the distance between their points. Clearly, the closest computed pair, is the closest pair of points.

Theorem 4.2.4 *Given a set P of points in $\mathbb{R}^d$, one can compute the closest pair of points of P in $O(n \log n)$ time.*

We remind the reader that we already saw a linear time (expected) time algorithm for this problem. However, this is a deterministic algorithm, and it can be applied in more abstract settings of finite metric spaces that have small WSPD.

4.3 All Nearest Neighbors

Given a set P of n points in $\mathbb{R}^d$, we would like to compute for each point $p \in P$, its closest neighbor in P. This is harder than it might seem at first, since this is *not* a symmetrical relationship. Indeed, p might be the nearest neighbor to q, but q might have r for a nearest neighbor.

4.3.1 The bounded spread case

Assume P is contained in the unit square, and $\operatorname{diam}(P) \geq 1/4$. Furthermore, let $\Phi = \Phi(P)$ denote the spread of P. Compute a ε^{-1}-WSPD $\mathcal{W}$ of P, for $\varepsilon = 1/4$. Arguing as in the closest pair case, we have that if q is a nearest neighbor to p, then there exists a pair $\{u, v\} \in \mathcal{W}$, such that $P_u = \{p\}$ and $q \in P_v$. Thus, scan all the pairs with singleton as one of their sides, and for each such singleton, record the closest point encountered.

Lemma 4.3.1 *Let P be a set n points in the plane, then one can solve the all nearest neighbor problem, in time $O(n(\log n + \log \Phi(P)))$ time, where Φ is the spread of P.*

Proof: The algorithm is described above. We only remain with the task of analyzing the running time. For a number $i \in \{0, -1, \ldots, -\lfloor \lg \Phi \rfloor - 4\}$, consider the set of pairs W_i, such that $\{u, v\} \in W_i$, if and only if $\{u, v\} \in \mathcal{W}$, and $2^{i-1} \leq \left\|\mathrm{rep}_u \mathrm{rep}_v\right\| \leq 2^i$.

We claim, that for any node u in the compressed quadtree, we have that the number of pairs of W_i that contains u is a constant. As such, the points of P_u are being scanned only constant number of times because of pairs in W_i, and thus every point is being scanned only $O(\log \Phi)$ times overall, yielding the required running time.

Indeed, if $\{u, v\} \in W_i$, then $\Box_{\overline{\mathrm{p}}(v)}$ must have a side length which is $\Omega(\varepsilon \|\mathrm{rep}_u \mathrm{rep}_v\|) = \Omega(\|\mathrm{rep}_u \mathrm{rep}_v\|) = \Omega(2^i)$. On the other hand, the set

$$U(u) = \left\{ \overline{\mathrm{p}}v \;\middle|\; \{u, v\} \in W_i \right\}$$

is a set of nodes which their corresponding cells are disjoint, and all of them are in distances $O(2^i)$ from $\Box_u$. It follows that $|U(u)| = O(1)$. This in turn implies that $\left| \left\{ v \;\middle|\; \{u, v\} \in W_i \right\} \right| = O(1)$. ■

4.3.2 The unbounded spread case

To handle the unbounded case, we need to use some additional geometric properties.

Lemma 4.3.2 *Let u be a node in $\mathcal{QT}(P)$, we partition spaces around* rep_u *into cones of angle $\leq \pi/20$. Let* cone *be one of those cones, and let Q be the set of all points in P which are in distance $\geq 8\,\mathrm{diam}(P_u)$ from* rep_u*, and they all lie inside* cone*. Let q be the closest point in Q to* rep_u*. Then, q is the only point in Q that its nearest neighbor might be in P_u.*

Lemma 4.3.2 implies that we can do a top-down traversal of $\mathcal{QT}(P)$, after computing a ε^{-1}-WSPD of P, for $\varepsilon = 1/16$. For every node u, we maintain its nearest neighbor in its cone, but only of points that are in nodes v that are well separated from u, and that P_v is a singleton. We can do that, by first getting the candidate points from our parent, in addition, we just need to add all the points that lie in a pair $\{u, v\}$ such that $|P_v| = 1$. After we computed this set, we keep only the closest point inside each cone around rep_u. Clearly, this can be done in linear time, since those sets have constant size, and every pair contribute a point at most once to this top-down computation.

Theorem 4.3.3 *Given a set P of n points in $\mathbb{R}^d$, one can solve the all nearest neighbor problem in $O(n \log n)$ time.*

4.4 Bibliographical Notes

Well separated pairs decomposition was defined by Callahan and Kosaraju [CK95]. They defined a different space decomposition tree, known as the *fair split* tree. Here, one compute the axis parallel bounding box of the point-set, and always split along the longest edge by a perpendicular plane in the middle (or near the middle). This splits the point set into two sets, which we construct fail split tree for them recursively. Implementing this in $O(n \log n)$ time requires some cleverness. See [CK95] for details.

Callahan and Kosaraju [CK95] were inspired by the work of Vaidya [Vai86] on all nearest neighbor problem (i.e., compute for each points in P, their nearest neighbor in P). He defined the fair split tree, and show how to compute the all nearest neighbors in $O(n \log n)$ time. However, the first to give an $O(n \log n)$ time algorithm for the all nearest neighbor algorithm was Clarkson [Cla83] (this was part of his PhD thesis).

Section 4.3 is a simplification of the all k-nearest neighbor problem. This computes for every points its k-nearest neighbor in $O(n \log n + nk)$ time. See [CK95] for details.

WSPD can be maintained in polylogarithmic time under insertions and deletions. This is quite surprising when one considers that in the worst case, a point might participate in linear number of pairs, and in fact, a node in the quadtree might participate in linear number of pairs. This is described in detail in Callahan thesis []

Chapter 5

Approximate Nearest Neighbor Search in Low Dimension

"Napoleon has not been conquered by man. He was greater than all of us. But god punished him because he relied on his own intelligence alone, until that prodigious instrument was strained to breaking point. Everything breaks in the end."

– – Carl XIV Johan, King of Sweden

5.1 Introduction

Let P be a set of n points in $\mathbb{R}^d$. We would like to preprocess it, such that given a query point q, one can determine the closet point in P to q quickly. Unfortunately, the exact problem seems to require prohibitive preprocessing time. (Namely, computing the Voronoi diagram of P, and preprocessing it for point-location queries. This requires (roughly) $O\left(n^{\lceil d/2 \rceil}\right)$ time.)

Instead, we will specify a parameter $\varepsilon > 0$, and build a data-structure that answers $(1+\varepsilon)$-approximate nearest neighbor queries.

Definition 5.1.1 For a set $P \subseteq \mathbb{R}^d$, and a point q, we denote by $\widehat{q}$ the closest point in P to q. We denote by $\mathbf{d}_P(q)$ the distances between q and its closest point in P; that is $\mathbf{d}_P(q) = \left\| q\widehat{q} \right\|$.

For a query point q, and a set P of n points in $\mathbb{R}^d$, the point $x \in P$ is an $(1+\varepsilon)$*-approximate nearest neighbor* if $\|qx\| \leq (1+\varepsilon)\mathbf{d}_P(q)$. Alternatively, for any $y \in P$, we have $\|xq\| \leq (1+\varepsilon)\|yq\|$. We will denote this fact by saying that x is $(1+\varepsilon)$-ANN of q.

5.2 Low Quality ANN Search - The Ring Separator Tree

Definition 5.2.1 A binary tree T having the points of P as leaves, is a *t-ring separator tree* for P, if every node $v \in T$, is associated with a ball $\mathbf{b}_v$, such that all the points of $P_{\text{in}}^v = P_v \cap \mathbf{b}_v$ are in one child of T, where $\mathbf{b}_v = \mathbf{b}(p_v, r_v)$ and P_v are the points of P stored in the subtree of v. Furthermore, all the other points of P_v are outside the interior of the enlarged ball $\mathbf{b}(p_v, (1+t)r_v)$, and are stored in the other child of v. Finally, we store a representative point $\text{rep}_v \in P_{\text{in}}^v$ in v.

Lemma 5.2.2 *Given a t-ring separator tree T, one can answer $(1+4/t)$-approximate nearest neighbor queries, in $O(\texttt{depth}(T))$ time.*

Proof: Let q denote the query point, and set v to be the root of T. The algorithm answer the nearest-neighbor query by traversing down T.

During the traversal, we first compute the distance $l = \left\|q\,\mathrm{rep}_v\right\|$. If this is shorter than the nearest neighbor of q encountered so far, we update the closest distance currently stored for q.

If $\|qp_v\| \leq (1 + t/2)r_v$, we continue the search recursively in the child containing P^v_{in}. Otherwise, we continue the search in the subtree containing P^v_{out}. Let π denote the generated search path in T.

As for the quality of approximation, let $\widehat{q}$ denote the nearest neighbor to q in P. And let w denote the last node in the search path π, such that $\widehat{q} \in P_w$. Clearly, if $\widehat{q} \in P^w_{\mathrm{in}}$, but we continued the search in P^w_{out}, then $\left\|\widehat{q}q\right\| \geq (t/2)r_w$, and

$$\left\|q\,\mathrm{rep}_w\right\| \leq \left\|q\widehat{q}\right\| + \left\|\widehat{q}\,\mathrm{rep}_w\right\| \leq \left\|\widehat{q}q\right\| + 2r_w,$$

since $\widehat{q}, \mathrm{rep}_w \in \mathbf{b}_w = \mathbf{b}(p_w, r_w)$. In particular,

$$\frac{\left\|q\,\mathrm{rep}_w\right\|}{\left\|\widehat{q}q\right\|} \leq \frac{\left\|\widehat{q}q\right\| + 2r_w}{\left\|\widehat{q}q\right\|} \leq 1 + \frac{4}{t}.$$

Namely, rep_w is a $(1 + 4/t)$-approximate nearest neighbor to q.

Similarly, if $\widehat{q} \in P^w_{\mathrm{out}}$, but we continued the search in P^w_{in}, then $\left\|\widehat{q}q\right\| \geq (t/2)r_w$ and $\left\|q\,\mathrm{rep}_w\right\| \leq \|qp_w\| + \left\|p_w\,\mathrm{rep}_w\right\| \leq (2 + t/2)r_w$. Thus, rep_w is a $\frac{(2+t/2)r_w}{(t/2)r_w}$-ANN of q. Namely, rep_w is a $(1 + 4/t)$-ANN of q. ■

Lemma 5.2.3 *Given a set P of n points in $\mathbb{R}^d$, one can compute a $(1/n)$-ring separator tree of P in $O(n \log n)$ time.*

Proof: The construction is recursive. Compute the ball $D = \mathbf{b}(p, r)$ that contains $\geq n/c$ points of P, such that $r \leq 2r_{\mathrm{opt}}(P, n/c)$, where c is a constant to be determined shortly. We remind the reader that D can be computed in linear time, by Lemma 1.3.1. Consider the ball D' of radius $2r$ centered at p. The ball D' can be covered by a hypercube S with side length $4r$. Furthermore, partition S into a grid such that every cell is of side length r/L, for $L = \left\lceil 16\sqrt{d}\right\rceil$. Every cell in this grid has diameter $\leq 4r\sqrt{d}/L \leq r/4 \leq r_{\mathrm{opt}}(P, n/c)/2$. Thus, every grid cell can contain at most n/c points, since it can be covered with a ball of radius $r_{\mathrm{opt}}(P, n/c)/4$. There are $M = \left(\frac{4r}{r/L}\right)^d = (4L)^d$ grid cells. Thus, D' contains at most $(4L)^d(n/c)$ points. Specifically, for $c = 2(4L)^d$ the disk D' contains at most $n/2$ points of P.

In particular, there must be a radius r' such that $r \leq r' \leq 2r$, and there is a $h \geq r/n$, such that the ring $\mathbf{b}(p, r' + h) \setminus \mathbf{b}(p, r')$ does not contain any points of P in its interior.

Indeed, sort the points of P inside $D' \setminus D$ by their distances from p. There are $n/2$ numbers in the range of distances $[r, 2r]$. As such, there must be an interval of length $r/(n/2 + 1)$ which is empty. And this empty range, corresponds to the empty ring.

Computing r' and h is done by computing the distance of each point from p, and partitioning the distance range $[r, 2r]$ into $2n$ equal length segments. In each segment, we register the point with minimum and maximum distance from c in this range. This can be done in linear time using the floor function. Next, scan those buckets from left to right. Observe, that the maximum length gap is realized by a maximum of one bucket together with a consecutive sequence of empty buckets, ending by the minimum of a non empty bucket. As such, the maximum length interval can be computed in linear time, and yield r' and h.

Thus, let v be the root of the new tree, set P^v_{in} to be $P \cap \mathbf{b}(p, r')$ and $P^v_{\mathrm{out}} = P \setminus P^v_{\mathrm{in}}$, store $\mathbf{b}_v = \mathbf{b}(p, r')$ and $p_v = p$. Continue the construction recursively on those two sets. Since $\left|P^v_{\mathrm{in}}\right|, \left|P^v_{\mathrm{out}}\right| \geq n/c$, and c is constant. It follows that the construction time of the algorithm is $T(n) = O(n) + T(\left|P^v_{\mathrm{in}}\right|) + T(\left|P^v_{\mathrm{out}}\right|) = O(n \log n)$, and the depth of the resulting tree is $O(\log n)$. ■

Combining the above two lemmas, we get the following result.

Theorem 5.2.4 *Let P be a set of n points in $\mathbb{R}^d$. One can preprocess it in $O(n \log n)$ time, such that given a query point $q \in \mathbb{R}^d$, one can return a $(4n + 1)$-ANN of q in P in $O(\log n)$ time.*

5.3 The bounded spread case

Lemma 5.3.1 *Let P be a set of n points contained inside the unit hypercube in $\mathbb{R}^d$, and let $\mathcal{QT}$ be a quadtree of P, where* $\mathrm{diam}(P) = \Omega(1)$. *Let q be a query point, such that* $\left\|\widehat{q}q\right\| \geq r$. *Then, given a parameter $\varepsilon > 0$, one can return an $(1+\varepsilon)$-ANN to q in $O\left(1/\varepsilon^d + \log(1/r)\right)$ time*

Proof: Let $A_0 = \{\mathrm{root}(T)\}$, and let $r_{\mathrm{curr}} = \left\|q\,\mathrm{rep}_{\mathrm{root}(T)}\right\|$.

In the ith iteration, for $i > 0$, the algorithm expands the nodes of A_i to get A_{i+1}. Formally, for $v \in A_{i-1}$, let C_v be the set of children of v in $\mathcal{QT}$ and $\square_v$ denote the cell (i.e., region) v corresponds to. For every node $w \in C_v$, we compute $r_{\mathrm{curr}} = \min(r_{\mathrm{curr}}, \left\|q\,\mathrm{rep}_w\right\|)$. If $\left\|q\,\mathrm{rep}_w\right\| - \mathrm{diam}(\square_w) < (1-\varepsilon/2)r_{\mathrm{curr}}$, the set P_w might contain points which are closers to q than the current point we have, and we add q to A_i (more precisely, P_w might contain a point which is $(1-\varepsilon/2)$ closer to q than any point we encountered so far). We stop as soon as A_i is an empty set, and we return r_{curr} together with the point of P that realized it, as the ANN to q.

Let us first argue that the algorithm works. Consider the last node w inspected by the algorithm such that $\widehat{q} \in P_w$. Since the algorithm decided to throw this node away, we have, by the triangle inequality, that

$$\left\|q\widehat{q}\right\| \geq \left\|q\,\mathrm{rep}_w\right\| - \mathrm{diam}(\square_w) \geq (1-\varepsilon/2)r_{\mathrm{curr}}.$$

Thus, $\left\|q\widehat{q}\right\|/(1-\varepsilon/2) \geq r_{\mathrm{curr}}$. However, $1/(1-\varepsilon/2) \leq 1+\varepsilon$, for $1 \geq \varepsilon > 0$, as can be easily verified. Thus, $r_{\mathrm{curr}} \leq (1+\varepsilon)\mathbf{d}_P(q)$, and the algorithm returns an $(1+\varepsilon)$-ANN to q.

As for the running time, observe that if a node $w \in \mathcal{QT}$ is considered by the algorithm, and $\mathrm{diam}(\square_w) < (\varepsilon/4)\left\|q\widehat{q}\right\|$ then

$$\left\|q\,\mathrm{rep}_w\right\| - \mathrm{diam}(\square_w) \geq \left\|q\,\mathrm{rep}_w\right\| - (\varepsilon/4)\left\|q\widehat{q}\right\| \geq r_{\mathrm{curr}} - (\varepsilon/4)r_{\mathrm{curr}} \geq (1-\varepsilon/4)r_{\mathrm{curr}},$$

which implies that neither w nor any of its children would be inserted into the sets $A_1, \ldots, A_m$, where m is the depth $\mathcal{QT}$. Thus, no nodes of level $\leq h = \left\lceil \lg(\left\|q\widehat{q}\right\|\varepsilon/4)\right\rceil$ are going to be considered by the algorithm.

Furthermore, in the end of the ith iteration, we have that $r_{\mathrm{curr}} \leq l_i = \left\|q\widehat{q}\right\| + \sqrt{d}2^{-i}$. The only cells of $\mathrm{G}_{2^{-i-1}}$ that might be considered by the algorithm are the ones in distance $\leq l_i$ from q. The number of such cells is

$$n_i = O\left(\left\lceil \frac{l_i}{2^{-i-1}} \right\rceil\right)^d.$$

In particular, as long as $\sqrt{d}2^{-i-1} \geq \left\|q\widehat{q}\right\|$, we have that $l_i = O(2^{-i-1})$ and $n_i = O(1)$. On the other hand, let j be the first iteration for which $\sqrt{d}2^{-j-1} \leq \left\|q\widehat{q}\right\|$. We have for $i \geq j$ that

$$n_i = O\left(\left\lceil \frac{2^{-j-1}}{2^{-i-1}} \right\rceil^d\right).$$

Thus, $n_j, n_{j+1}, \ldots, n_m$ is a geometrically growing series, dominated by the last element. However, we know that we never access nodes of level $\leq h$. Thus, the running time of the algorithm is bounded by

$$\begin{aligned}
O\left(\sum_i |A_i|\right) &= O\left(|h| + \sum_{i=0}^{h} n_i\right) = O\left(|h| + \sum_{i=j}^{h} n_i\right) = O(|h| + n_h) \\
&= O\left(|h| + \left(\left\lceil \frac{\left\|q\widehat{q}\right\|}{2^h} \right\rceil\right)^d\right) \\
&= O\left(|h| + \left(\left\lceil \frac{\left\|q\widehat{q}\right\|}{2^{\left\lceil \lg(\left\|q\widehat{q}\right\|\varepsilon/4)\right\rceil}} \right\rceil\right)^d\right) = O\left(\log\frac{1}{\varepsilon r} + \frac{1}{\varepsilon^d}\right) \\
&= O\left(\log\frac{1}{r} + \log\frac{1}{\varepsilon} + \frac{1}{\varepsilon^d}\right) = O\left(\log\frac{1}{r} + \frac{1}{\varepsilon^d}\right),
\end{aligned}$$

since $\left\|q\widehat{q}\right\| \geq r$. ■

To apply Lemma 5.3.1 for the bounded spread, first realize that if the distance between the closest pair of points of P is $\mu = \mathcal{CP}(P)$, then the algorithm would never search in (the children of) cells that have diameter $\leq \mu/2$, since all such nodes are leafs. As such, we can replace in the above argumentation r by μ.

Lemma 5.3.2 *Let P be a set of n points in $\mathbb{R}^d$, and let $\mathcal{QT}$ be a quadtree of P, where* $\mathrm{diam}(P) = \Omega(1)$. *Given a query point q and $1 \leq \varepsilon > 0$, one can return an $(1+\varepsilon)$-ANN to q in $O\left(1/\varepsilon^d + \log \Phi(P)\right)$ time.*

A less trivial task, is to adapt the algorithm, so that it uses compressed quadtrees. To this end, the algorithm would still handle the nodes by levels. This requires us to keep a heap of integers in the range $0, -1, \ldots, -\lfloor \lg \Phi(P) \rfloor$. This can be easily done by maintaining an array of size $O(\log \Phi(P))$, where each array cell, maintains a linked list of all nodes with this level. Clearly, an insertion/deletion of this data-structure can be handled in constant time by augmenting it with a hash table.

Theorem 5.3.3 *Let P be a set of n points in $\mathbb{R}^d$. One can preprocess P in $O(n \log n)$ time, and using linear space, such that given a query point q and parameter $1 \geq \varepsilon > 0$, one can return an $(1+\varepsilon)$-ANN to q in $O\left(1/\varepsilon^d + \log \Phi(P)\right)$ time.*

If the spread of P is not known, but we are provided with a lower bound $r \leq \left\|q\widehat{q}\right\|$ then the query time is $O(1/\varepsilon^d + \log(\mathrm{diam}(P)/r))$, where $\widehat{q}$ is the nearest neighbor of q in P.

5.4 ANN - general case

The scheme for handling the general case is to use the ring separator tree to get a fast rough approximation. Next, using a compressed quadtree, we would find a constant number of relevant nodes, and apply Theorem 5.3.3 to those nodes. This would yields the required approximation. Before solving this problem, we need a minor extension of the compressed quadtree data-structure.

5.4.1 Extending a compressed quadtree to support cell queries

Let $\widehat{\square}$ be a canonical grid cell (we remind the reader that this is a cell of the grid $G_{2^{-i}}$, for some integer $i \leq 0$). Given a compressed quadtree $\widehat{T}$, we would like to find the *single* node $v \in \widehat{T}$, such that $P \cap \widehat{\square} = P_v$.

To this end, we construct a finger tree over $\widehat{T}$. We recall that the finger tree is constructed by finding a separator node in $\widehat{T}$, which breaks $\widehat{T}$ into a constant number of connected components, creating a corresponding node in the finger tree, and continuing the construction recursively on each connected component. The construction takes $O(n \log n)$ time.

Thus, we are going to perform the search on the finger tree. At every stage, we have a node w', which corresponds to a node w in the compressed quadtree. There are several possibilities:

1. If $\square_w \cap \widehat{\square} = \emptyset$, we continue the search recursively in the child of w' which corresponds to the connected component of $\widehat{T}$ constructed for the region which is the complement of $\square_w$. Let u' denote this child of w'.

2. If $\square_w \subseteq \widehat{\square} \subset \square_{\overline{\mathrm{p}}(w)}$, where $\overline{\mathrm{p}}(w)$ is the parent of w in $\widehat{T}$, then w is the required node, and the algorithm returns it.

3. If $\square_{\overline{\mathrm{p}}(w)} \subseteq \widehat{\square}$, then we continue the search recursively in u'.

4. Otherwise, we need to continue the search in one of the children of w. We can determine which one in constant time.

Since the depth of the finger tree is logarithmic, this takes $O(\log n)$ time overall.

Lemma 5.4.1 *Given a compressed quadtree $\widehat{T}$ of a set P of n points, one can preprocess it in $O(n \log n)$ time, such that given a query canonical cell $\widehat{\Box}$, one can find, in $O(\log n)$ time, the node $w \in \widehat{T}$ such that $\Box_w \subseteq \widehat{\Box}$ and $P \cap \widehat{\Box} = P_w$.*

5.4.2 ANN in Low Dimensions

Let P be a set of n points in $\mathbb{R}^d$ contained in the unit hypercube. We build the compressed quadtree $\widehat{T}$ of P, and the ring separator tree $\mathcal{T}_R$ of P. Furthermore, we preprocess $\widehat{T}$ for cell queries, using Lemma 5.4.1.

Given a query point q, using the ring tree $\mathcal{T}_R$, we compute a point $u \in P$, such that $\mathbf{d}_P(q) \leq \|uq\| \leq (4n+1)\mathbf{d}_P(q)$. Let $R = \|uq\|$ and $r = \|uq\|/(4n+1)$. Clearly, $r \leq \mathbf{d}_P(q) \leq R$. Next, compute $l = \lceil \lg R \rceil$, and let C be the set of cells of G_{2^l} that are in distance $\leq R$ from q. Clearly, since $R \leq 2^l$, it follows that $\widehat{q} \in \bigcup_{\Box \in C} \Box$. For each cell $\Box \in C$, we compute the node $v \in \widehat{T}$ such that $P \cap \Box = P_v$. This can be done in $O(\log n)$ time, using Lemma 5.4.1. Let V be the resulting set of nodes of P.

For each node of $v \in V$, we now apply the algorithm of Theorem 5.3.3 to the compressed quadtree rooted at v. Since $|V| = O(1)$, and $\operatorname{diam}(P_v) = O(R)$, for all $v \in V$, the query time is

$$\begin{aligned}
\sum_{v \in V} O\left(\frac{1}{\varepsilon^d} + \log \frac{\operatorname{diam}(P_v)}{r}\right) &= O\left(\frac{1}{\varepsilon^d} + \sum_{v \in V} \log \frac{\operatorname{diam}(P_v)}{r}\right) = O\left(\frac{1}{\varepsilon^d} + \sum_{v \in V} \log \frac{R}{r}\right) \\
&= O\left(\frac{1}{\varepsilon^d} + \log n\right).
\end{aligned}$$

As for the correctness of the algorithm, notice that there is a node $w \in V$, such that $\widehat{q} \in P_w$. As such, when we apply the algorithm of Theorem 5.3.3 to w, it would return us a $(1+\varepsilon)$-ANN to q.

Theorem 5.4.2 *Let P be a set of n points in $\mathbb{R}^d$. One can construct a data-structure of linear size, in $O(n \log n)$ time, such that given a query point $q \in \mathbb{R}^d$, and a parameter $1 \geq \varepsilon > 0$, one can compute a $(1+\varepsilon)$-ANN to q in $O(1/\varepsilon^d + \log n)$ time.*

5.5 Bibliographical notes

The presentation of the ring separator tree follows the recent work of Har-Peled and Mendel [**?**]. Ring separator trees are probably an old idea. A more elaborate but similar data-structure is described by Indyk and Motwani [IM98]. Of course, the property that "thick" ring separators exist, is inherently low dimensional, as the regular simplex in $n-1$ dimensions show. One option is to allow the rings to contain points, and replicate the points inside the ring in both subtrees. As such, the size of the resulting tree is not necessarily linear. However, careful implementation yields linear (or small) size, see Exercise 5.6.1 for more details. This and several additional ideas are used in the construction of the cover tree of Indyk and Motwani [IM98].

Section 5.3 is a simplification of Arya *et al.* [AMN+98] work. Section 5.4 is also inspired to a certain extent by Arya *et al.* work, although it is essentially a simplification of Har-Peled and Mendel [**?**] data-structure to the case of compressed quadtrees. In particular, we believe that the data-structure presented is conceptually simpler than previously published work.

There is a huge literature on approximate nearest neighbor search, both in low and high dimensions in the theory, learning and database communities. The reason for this huge work lies in the importance of this problem, special input distributions, different computation models (i.e., I/O-efficient algorithms), search in high-dimensions, and practical efficiency.

Liner space. In the low dimensions, the seminal work of Arya *et al.* [AMN⁺98], mentioned above, was the first to offer linear size data-structure, with logarithmic query time, such that the approximation quality is specified with the query. The query time of Arya *et al.* is slightly worse than the running time of Theorem 5.4.2, since they maintain a heap of cells, always handling the cell closest to the query point. This results in running time $O(\varepsilon^{-d} \log n)$. It can be further improved to $O(1/\varepsilon^d \log(1/\varepsilon) + \log n)$ by observing that this heap has only very few del-min, and many insertions. This rather nice observation is due to Duncan [Dun99].

Instead of having a separate finger tree, Arya *et al.* rebalance the compressed quadtree directly. This results in nodes, which correspond to cells that have the shape of an annulus (i.e., the region formed by the difference between two canonical grid cells).

Duncan [Dun99] and some other authors offered data-structure (called the BAR-tree) with similar query time, but it is seems to be inferior, in practice, to Arya *et al.* work, for the reason that while the regions the nodes correspond to are convex, they have higher descriptive complexity, and it is harder to compute the distance of the query point to a cell.

Faster query time. One can improve the query time if one is willing to specify ε during the construction of the data-structure, resulting in a trade off between space for query time. In particular, Clarkson [Cla94] showed that one can construct a data-structure of (roughly) size $O(n/\varepsilon^{(d-1)/2})$, and query time $O(\varepsilon^{-(d-1)/2} \log n)$. Chan simplified and cleaned up this result [Cha98] and presented also some other results.

Details on Faster Query Time. A set of points Q is $\sqrt{\varepsilon}$*-far* from a query point q, if the $\|pc_Q\| \geq \mathrm{diam}(Q)/\sqrt{\varepsilon}$, where c_Q is some point of Q. It is easy to verify that if we partition space around c_Q into cones with central angle $O(\sqrt{\varepsilon})$ (this requires $O(1/\varepsilon^{(d-1)/2})$ cones), then the most extreme point of Q in such a cone cone, furthest away from c_Q, is the $(1+\varepsilon)$-approximate nearest neighbor for any query point inside cone which is $\sqrt{\varepsilon}$-far. Furthermore, by careful implementation (i.e., grid in the angles space), we can decide, in constant time, which cone the query point lies in. Thus, using $O(1/\varepsilon^{(d-1)/2})$ space, we can answer $(1+\varepsilon)$-ANN queries for Q, if the query point is $\sqrt{\varepsilon}$-far.
We next, construct this data-structure for every set P_v, for $v \in \widehat{T}(P)$. This results in a data-structure of size $O(n/\varepsilon^{(d-1)/2})$. Given a query point q, we use the algorithm of Theorem 5.4.2, and stop as soon as for a node v, P_v is $\sqrt{\varepsilon}$-far, and then we use the secondary data-structure for P_v. It is easy to verify that the algorithm would stop as soon as $\mathrm{diam}(\square_v) = O(\sqrt{\varepsilon}\mathbf{d}_P(q)$. As such, the number of nodes visited would be $O(\log n + 1/\varepsilon^{d/2})$, and identical query time.
Note, that we omitted the construction time (which requires some additional work to be done efficiently), and our query time is slightly worse than the best known. The interested reader can check out the work by Chan [Cha98], which is somewhat more complicated than what is outlined here.

The first to achieve $O(\log(n/\varepsilon))$ query time (using near linear space), was Har-Peled [Har01], using space roughly $O(n\varepsilon^{-d} \log^2 n)$. This was later simplified and improved by Arya and Malamatos [AM02], which present a data-structure with the same query time, and of size $O(n/\varepsilon^d)$. Those data-structure relies on the notion of computing approximate Voronoi diagrams and performing point location queries in those diagrams. By extending the notion of approximate Voronoi diagrams, Arya, Mount and Malamatos [AMM02] showed that one can answer $(1+\varepsilon)$-ANN queries in $O(\log(n/\varepsilon))$ time, using $O(n/\varepsilon^{(d-1)})$ space. On the other end of the spectrum, they showed that one can construct a data-structure of size $O(n)$ and query time $O(\log n + 1/\varepsilon^{(d-1)/2})$ (note, that for this data-structure $\varepsilon > 0$ has to be specified in advance). In particular, the later result breaks a space/query time tradeoff that all other results suffers from (i.e., the query time multiplied by the construction time has dependency of $1/\varepsilon^d$ on ε).

Practical Considerations Arya *et al.* [AMN⁺98] implemented their algorithm. For most inputs, it is essentially a *kd*-tree. The code of their library was carefully optimized and is very efficient. In particular, in practice, I would expect it to beat most of the algorithms mentioned above. The code of their implementation is available online as a library [AM98].

Higher Dimensions. All our results have exponential dependency on the dimension, in query and preprocessing time (although the space can be probably be made subexponential with careful implementation). Getting a subexponential algorithms requires a completely different techniques, and would be discussed in detail at some other point.

Stronger computation models. If one assume that the points have integer coordinates, in the range $[1, U]$, then approximate nearest-neighbor queries can be answered in (roughly) $O(\log \log U + 1/\varepsilon^d)$ time [AEIS99], or even $O(\log \log(U/\varepsilon))$ time [Har01]. The algorithm of Har-Peled [Har01] relies on computing a compressed quadtree of height $O(\log(U/\varepsilon))$, and performing fast point-location query in it. This only requires using the floor function and hashing (note, that the algorithm of Theorem 5.4.2 uses the floor function and hashing during the construction, but it is not used during the query). In fact, if one is allowed to slightly blowup the space (by a factor U^δ, where $\delta > 0$ is an arbitrary constant), the ANN query time can be improved to constant [HM04].

By shifting quadtrees, and creating $d + 1$ quadtrees, one can argue that the approximate nearest neighbor must lie in the same cell (and of the "right" size) of the query point in one of those quadtrees. Next, one can map the points into a real number, by using the natural space filling curve associated with each quadtree. This results in $d + 1$ lists of points. One can argue that a constant approximate neighbor must be adjacent to the query point in one of those lists. This can be later improved into $(1 + \varepsilon)$-ANN by spreading $1/\varepsilon^d$ points. This simple algorithm is due to Chan [Cha02].

The reader might wonder why we bothered with a considerably more involved algorithm. There are several reasons: (i) This algorithm requires the numbers to be integers of limited length (i.e., $O(\log U)$ bits), and (ii) it requires shuffling of bits on those integers (i.e., for computing the inverse of the space filling curve) in constant time, and (iii) the assumption is that one can combine d such integers into a single integer and perform xor on their bits in constant time. The last two assumptions are not reasonable when the input is made out of floating point numbers.

Further research. At least (and only) in low dimensions, the ANN problem seems to be essentially solved both in theory and practice (such proclamations are inherently dangerous, and should be taken with considerable amount of healthy skepticism). Indeed, for $\varepsilon > 1/\log^{1/d} n$, the current data structure of Theorem 5.4.2 provide logarithmic query time. Thus, ε has to be quite small for the query time to become bad enough that one would wish to speed it up.

Main directions for further research seems to be working on this problem in high dimensions, and solving it in other computation models.

Surveys. A survey on approximate nearest neighbor search in high dimensions is by Indyk [Ind04]. In low dimensions, there is a survey by Arya and Mount [AM04].

5.6 Exercises

Exercise 5.6.1 [10 Points]

Let P be a set of n points in $\mathbb{R}^d$. Show how to build a ring tree, of linear size, that can answer $O(\log n)$-ANN queries in $O(\log n)$ time. [**Hint:** Show, that there is always a ring containing $O(n/\log n)$ points, such that it is of width w, and its interior radius is $O(w \log n)$. Next, build a ring tree, replicating the points in both children of this ring node. Argue that the size of the resulting tree is linear, and prove the claimed bound on the query time and quality of approximation.]

Chapter 6

Approximate Nearest Neighbor via Point-Location among Balls

Today I know that everything watches, that nothing goes unseen, and that even wallpaper has a better memory than ours. It isn't God in His heaven that sees all. A kitchen chair, a coat-hanger a half-filled ash tray, or the wood replica of a woman name Niobe, can perfectly well serve as an unforgetting witness to every one of our acts.
– – The tin drum, Gunter Grass

6.1 Hierarchical Representation of Points

In the following, it would be convenient to carry out our discussion in a more generalized settings than just low dimensional Euclidean space.

Definition 6.1.1 A *metric space* $\mathcal{M}$ is a pair $(X, \mathbf{d})$ where X is a set and $\mathbf{d} : X \times X \to [0, \infty)$ is a *metric*, satisfying the following axioms: (i) $\mathbf{d}(x, y) = 0$ iff $x = y$, (ii) $\mathbf{d}(x, y) = \mathbf{d}(y, x)$, and (iii) $\mathbf{d}(x, y) + \mathbf{d}(y, z) \geq \mathbf{d}(x, z)$ (triangle inequality).

For example, $\mathbb{R}^2$ with the regular euclidean distance is a metric space.

In the following, we are going to assume that we are provided with a black box, such that given two points $x, y \in X$, we can compute the distances $\mathbf{d}(x, y)$ in constant time.

6.1.1 Low Quality Approximation by HST

We will use the following special type of metric spaces:

Definition 6.1.2 Let P be a set of elements, and T a tree having the elements for P as leaves. The tree T defines a *hierarchically well-separated tree* (HST) over the points of P, if to each vertex $u \in T$ there is associated a label $\Delta_u \geq 0$, such that $\Delta_u = 0$ if and only if u is a leaf of T. The labels are such that if a vertex u is a child of a vertex v then $\Delta_u \leq \Delta_v$. The distance between two leaves $x, y \in T$ is defined as $\Delta_{\mathrm{lca}(x,y)}$, where $\mathrm{lca}(x, y)$ is the least common ancestor of x and y in T.

If every internal node of T has exactly two children, we will refer to it as being a *binary HST* (BHST).

For convenience, we will assume that the underlying tree is binary (any HST can be converted to binary HST in linear time, while retaining the underlying distances). We will also associate with every vertex $u \in T$, an arbitrary leaf rep_u of the subtree rooted at u. We also require that $\mathrm{rep}_u \in \left\{ \mathrm{rep}_v \,\middle|\, v \text{ is a child of } u \right\}$.

A metric N is said to t-approximate the metric $\mathcal{M}$, if they are on the same set of points, and $\mathbf{d}_{\mathcal{M}}(u, v) \leq d_N(u, v) \leq t \cdot \mathbf{d}_{\mathcal{M}}(u, v)$, for any $u, v \in \mathcal{M}$.

It is not hard to see that any n-point metric is $(n-1)$-approximated by some HST.

Lemma 6.1.3 *Given a weighted connected graph G on n vertices and m edges, it is possible to construct, in $O(n \log n + m)$ time, a binary HST $\mathcal{H}$ that $(n-1)$-approximates the shortest path metric on G.*

Proof: Compute the minimum spanning tree of G in $O(n \log n + m)$ time, and let T denote this tree.

Sort the edges of T in non-decreasing order, and add them to the graph one by one. The HST is built bottom up. At each point we have a collection of HSTs, each corresponds to a connected component of the current graph. When an added edge merges two connected components, we merge the two corresponding HSTs into one by adding a new common root for the two HST, and labeling this root with the edge's weight times $n-1$. This algorithm is only slight variation on Kruskal algorithm, and has the same running time.

We next estimate the approximation factor. Let x, y be two vertices of G. Denote by e the first edge that was added in the process above that made x and y in the same connected component C. Note that at that point of time e is the heaviest edge in C, so $w(e) \leq d_G(x,y) \leq (|C|-1)\, w(e) \leq (n-1)\, w(e)$. Since $d_H(x,y) = (n-1)\, w(e)$, we are done. ■

Corollary 6.1.4 *For a set P of n points in $\mathbb{R}^d$, one can construct, in $O(n \log n)$ time, a BHST $\mathcal{H}$ that $(2n-2)$-approximates the distances of points in P. That is, for any $p, q \in P$, we have $\mathbf{d}_{\mathcal{H}}(p,q)/(2n-2) \leq \|pq\| \leq \mathbf{d}_{\mathcal{H}}(p,q)$.*

Proof: We remind the reader, that in $\mathbb{R}^d$, one can compute a 2-spanner for P of size $O(n)$, in $O(n \log n)$ time (see Theorem 4.2.1). Let G be this spanner, and apply Lemma 6.1.3 to this spanner. Let $\mathcal{H}$ be the resulting metric. For any $p, q \in P$, we have $\|pq\| \leq \mathbf{d}_{\mathcal{H}}(p,q) \leq (n-1)\mathbf{d}_G(p,q) \leq 2(n-1)\|pq\|$. ■

Corollary 6.1.4 is unique to $\mathbb{R}^d$ since for general metric spaces, no HST can be computed in subquadratic time, see Exercise 6.5.1.

Corollary 6.1.5 *For a set P of n points in a metric space $\mathcal{M}$, one can compute a HST $\mathcal{H}$ that $(n-1)$-approximates the metric $\mathbf{d}_{\mathcal{M}}$.*

6.2 ANN using Point-Location Among Balls

In the following, let P be a set of n points in $\mathcal{M}$, where $\mathcal{M}$ is a metric space.

Definition 6.2.1 For a set of balls $\mathcal{B}$ such that $\bigcup_{b \in \mathcal{B}} b = \mathcal{M}$ (i.e., one of the balls might be of infinite radius and it covers the whole space $\mathcal{M}$), and a query point $q \in \mathcal{M}$, the *target ball of q in* $\mathcal{B}$, denoted by $\odot_{\mathcal{B}}(q)$, is the smallest ball of $\mathcal{B}$ that contains q (if several equal radius balls contain q we resolve this in an arbitrary fashion).

Our objective, is to show that $(1+\varepsilon)$-ANN queries can be reduced to target ball queries, among a near linear size set of balls. But let us first start from a "silly" result, to get some intuition about the problem.

Lemma 6.2.2 *Let $\mathcal{B} = \cup_{i=-\infty}^{\infty} \mathcal{B}(P, (1+\varepsilon)^i)$, where $\mathcal{B}(P, r) = \cup_{p \in P} \mathbf{b}(p, r)$. For $q \in \mathbb{R}^d$, let $\mathbf{b} = \odot_{\mathcal{B}}(q)$, and let $p \in P$ be the center of* $\mathbf{b}$. *Then, p is $(1+\varepsilon)$-ANN to q.*

Proof: Let $\widehat{q}$ be the nearest neighbor to q in P, and let $r = \mathbf{d}_P(q)$. Let i be such that $(1+\varepsilon)^i < r \leq (1+\varepsilon)^{i+1}$. Clearly, radius($\mathbf{b}$) $> (1+\varepsilon)^i$. On the other hand, $p \in \mathbf{b}(\widehat{q}, (1+\varepsilon)^{i+1})$. It follows that, $\|qp\| \leq (1+\varepsilon)^{i+1} \leq (1+\varepsilon)\mathbf{d}_P(q)$. Implying that p is $(1+\varepsilon)$-ANN to P. ■

Remark 6.2.3 Here is an intuitive construction of a set of balls of polynomial size, such that target queries answer $(1 + \varepsilon)$-ANN correctly. Indeed, consider two points $u, v \in P$. As far as correctness, we care if the ANN returns either u or v, for a query point q, only if $\mathbf{d}_P(()q) \in [\mathbf{d}_{\mathcal{M}}(u, v)/4, 2\mathbf{d}_{\mathcal{M}}(u, v)/\varepsilon]$ (for shorter distances, either u or v are the unique ANN, and for longer distances either one of them is ANN for the set $\{u, v\}$).

Next, consider a range of distances $[(1 + eps)^i, (1 + \varepsilon)^{i+1}]$ to be active, if there is $u, v \in P$, such that $\varepsilon(1 + \varepsilon)^i \le \mathbf{d}_{\mathcal{M}}(u, v) \le 4(1 + \varepsilon)^{i+1}/\varepsilon$. Clearly, the number of active intervals is $O(n^2\varepsilon^{-1} \log(1/\varepsilon))$ (one can prove a better bound). Generate a ball for each point of P, for each active range. Clearly, the resulting number of balls is polynomial, and can be used to resolve ANN queries.

Getting the number of balls to be near linear, requires to be more careful, and the details are provided below.

6.2.1 Handling a Range of Distances

Definition 6.2.4 For a real number $r > 0$, a *near-neighbor data structure*, denoted by $\mathrm{NNbr} = \mathrm{NNbr}(P, r)$, is a data-structure, such that given a query point q, it can decide whether $\mathbf{d}_P(q) \le r$, or $\mathbf{d}_P(q) > r$. If $\mathbf{d}_P(q) \le r$, it also returns as a witness a point $p \in P$, such that $\mathbf{d}(p, q) \le r$.

The data-structure $\mathrm{NNbr}(P, r)$ can be realized by just a set n balls around the points of P of radius r, and performing target ball queries on this set. For the time being, the reader can consider $\mathrm{NNbr}(P, r)$ as just being this set of n balls.

Definition 6.2.5 One can in fact, resolve ANN queries on a range of distances, $[a, b]$, by building NNbr data structures, with exponential jumps on this range. Formally, let $\mathcal{N}_i = \mathrm{NNbr}(P, r_i)$, where $r_i = (1 + \varepsilon)^i a$, for $i = 0, \ldots, M - 1$, where $M = \lceil \log_{1+\varepsilon}(b/a) \rceil$. And let $\mathcal{N}_M = \mathrm{NNbr}(P, M)$, where $r_M = b$. We will denote this set of data-structures by $\widehat{\mathcal{I}}(P, a, b, \varepsilon) = \{\mathcal{N}_0, \ldots, \mathcal{N}_M\}$. We refer to $\widehat{\mathcal{I}}$ is *interval near-neighbor* data structure.

Lemma 6.2.6 *Given P as above, and parameters $a \le b$ and $\varepsilon > 0$. We have: (i) $\widehat{\mathcal{I}}(P, a, b, \varepsilon)$ is made out of $O(\varepsilon^{-1} \log(b/a))$ NNbr data structures, and (ii) $\widehat{\mathcal{I}}(P, a, b, \varepsilon)$ contains $O(\varepsilon^{-1} n \log(b/a))$ balls overall.*

Furthermore, one can decide if either of the following options holds: (i) $\mathbf{d}_P(q) < a$, (ii) $\mathbf{d}_P(q) > b$, (iii) or return a number r and a point $p \in P$, such that $\|pq\| \le \mathbf{d}_P(q) \le (1 + \varepsilon)\|pq\|$. This requires two NNbr *queries if (i) or (ii) holds, and $O(\log(\varepsilon^{-1} \log(b/a)))$ otherwise.*

Proof: Given a query point q, we first check if $\mathbf{d}_P(q) \le a$, by querying $\mathcal{N}_0$, and if so, the algorithm returns "$\mathbf{d}_P(q) \le a$". Otherwise, we check if $\mathbf{d}_P(q) > b$, by querying $\mathcal{N}_M$, and if so, the algorithm returns "$\mathbf{d}_P(q) > b$".

Otherwise, let $X_i = 1$ if and only if $\mathbf{d}_P(q) \le r_i$, for $i = 0, \ldots, m$. We can determine the value of X_i by performing a query in the data-structure $\mathcal{N}_i$. Clearly, $X_0, X_1, \ldots, X_M$ is a sequence of zeros, followed by a sequence of ones. As such, we can find the i, such that $X_i = 0$ and $X_{i+1} = 1$, by performing a binary search. This would require $O(\log M)$ queries. In this case, we have that $r_i < \mathbf{d}_P(q) \le r_{i+1} \le (1 + \varepsilon)r_i$. Namely, the ball in $\mathrm{NNbr}(P, r_{i+1})$ covering q, corresponds to $(1 + \varepsilon)$-ANN to q.

To get the state bounds, observe that by the Taylor's expansion $\ln(1 + x) = x - x^2/2 + x^3/3 + \cdots \ge x/2$, for $x \ge 1$. Thus,

$$M = \lceil \log_{1+\varepsilon}(b/a) \rceil = O\left(\frac{\ln(b/a)}{\ln(1 + \varepsilon)}\right) = O(\log(b/a)/\varepsilon),$$

since $1 \ge \varepsilon > 0$. ■

Corollary 6.2.7 *Let P be a set of n points in $\mathcal{M}$, and let $a < b$ be real numbers. For a query point $q \in \mathcal{M}$, such that $\mathbf{d}_P(()q) \in [a, b]$, the target query over the set of balls $\mathrm{NNbr}(P, a, b, \varepsilon)$ returns a ball centered at $(1 + \varepsilon)$-ANN to q.*

Lemma 6.2.6 implies that we can "cheaply" resolve $(1+\varepsilon)$-ANN over intervals which are not too long.

Definition 6.2.8 For a set P of n points in a metric space $\mathcal{M}$, let $\mathcal{U}_{\text{balls}}(P, r) = \bigcup_{p \in P} \mathbf{b}(p, r)$ denote the *union of balls of radius r* around the points of P.

Lemma 6.2.9 *Let Q be a set of m points in $\mathcal{M}$ and $r > 0$ a real number, such that $\mathcal{U}_{\text{balls}}(Q, r)$ is a connected set. Then: (i) Any two points $p, q \in Q$ are in distance $\leq 2r(m-1)$ from each other. (ii) For $q \in \mathcal{M}$ a query point, such that $\mathbf{d}_Q(q) > 2mr/\delta$, then any point of Q is $(1+\delta)$-ANN of q.*

Proof: (i) Since $\mathcal{U}_{\text{balls}}(Q, r)$ is a connected set, there is a path of length $\leq (m-1)2r$ between any two points x, y of P. Indeed, consider the graph G, which connects two vertices $u, v \in P$, if $\mathbf{d}_{\mathcal{M}}(u, v) \leq 2r$. Since $\mathcal{U}_{\text{balls}}(Q, r)$ is a connected set, it follows that G is connected. As such, there is a path of length $m-1$ between x and y in G. This corresponds to a path of length $\leq (m-1)2r$ connecting x to y in $\mathcal{M}$, and this path lies inside $\mathcal{U}_{\text{balls}}(Q, r)$.

(ii) For any $p \in Q$, we have

$$\frac{2mr}{\delta} \leq \mathbf{d}_{\mathcal{M}}(q, p) \leq \mathbf{d}_{\mathcal{M}}(q, \widehat{q}) + \mathbf{d}_{\mathcal{M}}(\widehat{q}, p) \leq \mathbf{d}_{\mathcal{M}}(q, \widehat{q}) + 2mr \leq (1+\delta)\mathbf{d}_{\mathcal{M}}(q, \widehat{q}),$$

where $\widehat{q}$ is the nearest-neighbor of q in Q. ■

Lemma 6.2.9 implies that for faraway query points, a cluster points Q which are close together can be treated as a single point.

6.2.2 The General Case

Theorem 6.2.10 *Given a set P of n points in $\mathcal{M}$, then one can construct data-structures $\mathcal{D}$ that answers $(1+\varepsilon)$-ANN queries, by performing $O(\log(n/\varepsilon))$* NNbr *queries. The total number of balls stored at $\mathcal{D}$ is $O(n\varepsilon^{-1}\log(n/\varepsilon))$.*

Let $\mathcal{B}$ be the set of all the balls stored at $\mathcal{D}$. Then a target query on $\mathcal{B}$ answers $(1+\varepsilon)$-ANN query.

Proof: We are going to build a tree $\mathcal{D}$, such that each node v would have an interval near-neighbor data-structure $\widehat{\mathcal{I}}_v$ associated with it. As we traverse down the tree, we will use those data-structure to decide to what child to continue the search into.

Compute the minimum value $r > 0$ such that $\mathcal{U}_{\text{balls}}(P, r)$ is made out of $\lceil n/2 \rceil$ connected components. We set $\widehat{\mathcal{I}}_{\text{root}(T)} = \widehat{\mathcal{I}}(P, r, R, \varepsilon/4)$, where $R = 2\overline{c}\mu n r/\varepsilon$, μ is a global parameter and $\overline{c} > 1$ is an appropriate constant, both to be determined shortly. For each connected component $\mathcal{C}$ of $\mathcal{U}_{\text{balls}}(P, r)$, we build recursively a tree for $\mathcal{C} \cap P$ (i.e., the points corresponding to $\mathcal{C}$), and hung it on root(T). Furthermore, from each such connected component $\mathcal{C}$, we pick one representative point $q \in \mathcal{C} \cap P$, and let Q be this set of points. We also build (recursively) a tree for Q, and hang it on root(T). We will refer to the child of root(T) corresponding to Q, as the *outer child* of root(T).

Given a query point $q \in \mathcal{M}$, use $\widehat{\mathcal{I}}_{\text{root}(T)} = \widehat{\mathcal{I}}(P, r, R, \varepsilon/4)$ to determine, if $\mathbf{d}_P(q) \leq r$. If so, we continue the search recursively in the relevant child built for the connected component of $\mathcal{U}_{\text{balls}}(P, r)$ containing q (we know which connected component it is, because $\widehat{\mathcal{I}}_{\text{root}(T)}$ also returns a point of P in distance $\leq r$ from q). If $\mathbf{d}_P(q) \in [r, R]$, then we will find its $(1+\varepsilon)$-ANN from $\widehat{\mathcal{I}}_{\text{root}(T)}$. 'Otherwise, $\mathbf{d}_P(q) > R$, and we continue the search recursively in the outer child of root(T).

Observe, that in any case, we continue the search on a set of balls of size $\leq n/2 + 1$. As such, after number of steps $\leq \log_{3/2} n$ the search halts.

correctness. If during the search the algorithm traverse from a node v down to one of the connected components which is a child of $\mathcal{U}_{\mathrm{balls}}(P_v, r_v)$, then no error is introduced, where P_v is the point set used in constructing v, and r_v is the value of r used in the construction. If the query is resolved by $\widehat{\mathcal{I}}_v$, then a $(1+\varepsilon/4)$ error is introduced into the quality of approximation. If the algorithm continues the search in the outer child of v, then an error of $1 + \delta_v$ is introduced in the answer, where $\delta_v = \varepsilon/(\bar{c}\mu)$, by Lemma 6.2.9 (ii). Thus, the overall quality of the ANN returned in the worst case is

$$t \;\le\; \left(1 + \frac{\varepsilon}{4}\right) \prod_{i=1}^{\log_{3/2} n} \left(1 + \frac{\varepsilon}{\bar{c}\mu}\right) \le \exp\!\left(\frac{\varepsilon}{4}\right) \prod_{i=1}^{\log_{3/2} n} \exp\!\left(\frac{c\varepsilon}{\bar{c}\mu}\right),$$

since $x \le e^x$ for $x \le 1$. Thus, setting $\mu = \left\lceil \log_{3/2} n \right\rceil$ and $\bar{c}$ to be a sufficiently large constant, we have $t \le \exp\!\left(\varepsilon/4 + \sum_{i=1}^{\log_{3/2} n} \varepsilon/\bar{c}\mu\right) \le \exp(\varepsilon/2) \le 1 + \varepsilon$, since $\varepsilon < 1/2$. We used the fact that $e^x \le (1 + 2x)$ for $x \le 1/2$, as can be easily verified.

Number of queries. As the search algorithm proceeds down the tree $\mathcal{D}$, at most two NNbr queries are performed at each node. At last node of the traversal, the algorithm performs $O(\log(\varepsilon^{-1} \log(n/\varepsilon))) = O(\log(n/\varepsilon))$ queries, by Lemma 6.2.6

Number of balls. We need a new interpretation of the construction algorithm. In particular, let $\mathcal{H}$ be the HST constructed for P using the exact distances. It is easy to observe, that a connected component of $\mathcal{U}_{\mathrm{balls}}(P, r)$ is represented by a node v and its subtree in $\mathcal{H}$. In particular, the recursive construction for each connected component, is essentially calling the algorithm recursively on a subtree of $\mathcal{H}$. Let V be the set of nodes of $\mathcal{H}$ which represent connected components of $\mathcal{U}_{\mathrm{balls}}(P, r)$.

Similarly, the outer recursive call, can be charged to the upper tree of $\mathcal{H}$, having the nodes of V for leafs. Indeed, the outer set of points, is the set $\mathrm{rep}(V) = \left\{ \mathrm{rep}_v \;\middle|\; v \in V \right\}$. Let $\widehat{L}$ be this collection of subtrees of $\mathcal{H}$. Clearly, those subtrees are not disjoint in their vertices, but they are disjoint in their edges. The total number of edges is $O(n)$, and $\left|\widehat{L}\right| \ge n/2$.

Namely, we can interpret the algorithm (somewhat counter intuitively) as working on $\mathcal{H}$. At every stage, we break the current HST into subtrees, and recursively continue the construction on those connected subtrees.

In particular, for a node $v \in T$, let n_v be the number of children of v. Clearly, $|P_v| = O(n_v)$, and since we can charge each such child to the fact that we are disconnecting the edges of $\mathcal{H}$ from each other, we have that $\sum_{vin\mathcal{D}} n_v = O(n)$.

At a node $v \in \mathcal{D}$, we have that the data-structure $\widehat{\mathcal{I}}_v$ requires storing $m_v = O(\varepsilon^{-1} |P_v| \log(R_v/r_v)) = O(\varepsilon^{-1} n_v \log(\mu n_v/\varepsilon))$ balls. In particular, $m_v = O(\varepsilon^{-1} n_v \log(\mu n_v/\varepsilon))$. We conclude, that the overall number of balls stored in $\mathcal{D}$ is

$$\sum_{v \in \mathcal{D}} O\!\left(\frac{n_v}{\varepsilon} \log \frac{\mu n_v}{\varepsilon}\right) = O\!\left(\frac{n}{\varepsilon} \log \frac{n \log n}{\varepsilon}\right) = O\!\left(\frac{n}{\varepsilon} \log \frac{n}{\varepsilon}\right).$$

A single target query. The claim that a target query on $\mathcal{B}$ answers $(1 + \varepsilon)$-ANN query on P, follows by an inductive proof on the algorithm execution. Indeed, if the algorithm is at node v, and let $\mathcal{B}_v$ be the set of balls stored in the subtree of v. We claim, that if the algorithm continue the search in w, then all the balls of $U = \mathcal{B}_v \setminus \mathcal{B}_w$ are not relevant for the target query.

Indeed, if w is the outer child of v, then all the balls in U are too small, and none of them contains q. Otherwise, $q \in \mathcal{U}_{\mathrm{balls}}(P_v, r_V)$. As such, all the balls of $\mathcal{B}_v$ that are bigger than the balls of $\mathcal{U}_{\mathrm{balls}}(P_v, r_v)$, and as such they can be ignored. Furthermore, all the other balls stored in other children of v, are of radius $\le r_v$, and are not in the same connected component as $\mathcal{U}_{\mathrm{balls}}(P_w, r_v)$ in $\mathcal{U}_{\mathrm{balls}}(P_v, r_v)$, as such, none of them is relevant for the target query.

The only other case, is when the algorithm stop the search at v. But at this case, we have $r_v \leq \mathbf{d}_P(q) \leq R_v$, and then all the balls in the children of v are either too big (i.e., the balls stored at the outer child are of radius $\geq R_v$), or too small (i.e., the balls stored at the regular children are of radius $< r_v$). Thus, only the balls of $\widehat{\mathcal{I}}(P_v, r_v, R_v, \varepsilon/4)$ are relevant, and there we know that the returend ball is a $(1+\varepsilon/4)$-ANN by Corollary 6.2.7.

Thus, the point returned by the target query on $\mathcal{B}$, is identical to running the search algorithm on $\mathcal{D}$, and as such, by the above prove, the result is correct. ■

6.2.2.1 Efficient Construction

Theorem 6.2.10 does not provide any bounds on the construction time, since it requires quadratic time in the worst case.

Lemma 6.2.11 *Given a set P of n points in $\mathcal{M}$ and $\mathcal{H}$ be a HST of P that t-approximates $\mathcal{M}$. Then one can construct data-structures $\mathcal{D}$ that answers $(1+\varepsilon)$-ANN queries, by performing $O(\log(n/\varepsilon))$* NNbr *queries. The total number of balls stored at $\mathcal{D}$ is $O(n\varepsilon^{-1}\log(tn/\varepsilon))$.*

The construction time is $O(n\varepsilon^{-1}\log(tn/\varepsilon))$.

Proof: We reimplement the algorithm of Theorem 6.2.10, by doing the decomposition directly on the HST $\mathcal{H}$. Indeed, let $U = \left\{ \Delta_v \,\middle|\, v \in \mathcal{H}, \text{and } v \text{ is an internal node} \right\}$. Let ℓ be the median value of U. Let V be the set of all the nodes v of $\mathcal{H}$, such that $v \in V$, if $\Delta_v \leq \ell$ and $\Delta_{\overline{\mathrm{p}}(v)} > \ell$. Next, build an $\widehat{\mathcal{I}} = \widehat{\mathcal{I}}(P, r, R)$, where

$$r = \ell/(2tn) \text{ and } R = (2\overline{c}nr\log n)/\varepsilon, \tag{6.1}$$

where $\overline{c}$ is a large enough constant. As in the algorithm of Theorem 6.2.10, the set V breaks $\mathcal{H}$ into $\lceil n/2 \rceil + 1$ subtrees, and we continue the construction on each such connected component. In particular, for the new root node we create, set $r_{\mathrm{root}(\mathcal{D})} = r$, $R_{\mathrm{root}(\mathcal{D})} = R$, and $\ell_{\mathrm{root}(\mathcal{D})} = \ell$.

Observe, that for a query point $q \in \mathcal{M}$, if $\mathbf{d}_Q(p) \leq r$, then $\widehat{\mathcal{I}}$ would return a ball, which in turn would correspond to a point stored in one of the subtrees rooted at a node $v \in V$. Let $\mathcal{C}$ be the connected component of $\mathcal{U}_{\mathrm{balls}}(P, r)$ that contains q, and let $Q = P \cap \mathcal{C}$. It is easy to verify that $Q \subseteq P_v$. Indeed, since $\mathcal{H}$ t-approximates $\mathbf{d}_{\mathcal{M}}$, and Q is a connected component of $\mathcal{U}_{\mathrm{balls}}(P, r)$, it follows that Q must be in the same connected component of $\mathcal{H}$, when considering distances $\leq t \cdot r < \ell$. But such a connected component of $\mathcal{H}$, is no more than a node $v \in \mathcal{H}$, and the points of P stored in the subtree v in $\mathcal{H}$. However, such a node v is either in V, or one of its ancestors is in V.

For a node v, the number of balls of $\widehat{\mathcal{I}}_V$ is $O(\varepsilon^{-1}n_v\log((\log n)n_v y/\varepsilon))$. Thus, the overall number of balls in the data-structure is as claimed.

As for the construction time, it is dominated by the size of $\widehat{\mathcal{I}}$ data-structures, and as such the same bound holds. ■

Using Corollary 6.1.4 with Lemma 6.2.11 we get the following.

Theorem 6.2.12 *Let P be a set of n points in $\mathbb{R}^d$, and $\varepsilon > 0$ a parameter. One can compute an a set of $O(n\varepsilon^{-1}\log(tn/\varepsilon))$ balls (the same bound holds for the running time), such that a ANN can be resolved by a target ball query on this set of balls.*

Alternatively, one can construct a data-structure where $(1+\varepsilon)$-ANN can be resolved by $O(\log(n/\varepsilon))$ NNbr *queries.*

6.3 ANN using Point-Location Among Approximate Balls

While the results of Theorem 6.2.10 and Theorem 6.2.12 might be surprising, they can not be used immediately to solve ANN, since they reduce the ANN problem into answering near neighbor queries, which seems to be also a hard problem to solve efficiently.

The key observation is, that we do not need to use exact balls. We are allowed to slightly deform the balls. This approximate near neighbor problem is considerably easier.

Definition 6.3.1 For a ball $\mathbf{b} = \mathbf{b}(p, r)$, a set $\mathbf{b}_\approx$ is an $(1 + \varepsilon)$-*approximation to* $\mathbf{b}$, if $\mathbf{b} \subseteq \mathbf{b}_\approx \subseteq \mathbf{b}(p, (1 + \varepsilon)r)$.

For a set of balls $\mathcal{B}$, the set $\mathcal{B}_\approx$ is an $(1 + \varepsilon)$-*approximation* to $\mathcal{B}$, if for any ball $\mathbf{b} \in \mathcal{B}$ there is a corresponding $(1 + \varepsilon)$-approximation $\mathbf{b}_\approx \in \mathcal{B}_\approx$. For a set $\mathbf{b}_\approx \in \mathcal{B}_\approx$, let $\mathbf{b} \in \mathcal{B}$ denote the ball corresponding to $\mathbf{b}_\approx$, $r_\mathbf{b}$ be the radius of $\mathbf{b}$, and let $p_\mathbf{b} \in P$ denote the center of $\mathbf{b}$.

For a query point $q \in \mathcal{M}$, the *target set* of $\mathcal{B}_\approx$ in q, is the set $\mathbf{b}_\approx$ of $\mathcal{B}_\approx$ that contains q and has the smallest radius $r_\mathbf{b}$.

Luckily, the interval near-neighbor data-structure still works in this settings.

Lemma 6.3.2 *Let $\mathcal{I}_\approx = \mathcal{I}_\approx(P, r, R, \varepsilon/16)$ be a $(1 + \varepsilon/16)$-approximation to $\widehat{\mathcal{I}}(P, r, R, \varepsilon/16)$. For a query point, $q \in \mathcal{M}$, if $\mathcal{I}_\approx$ returns a ball centered at $p \in P$ of radius α, and $\alpha \in [r, R]$ then p is $(1 + \varepsilon/4)$-ANN to q.*

Proof: The data-structure of $\mathcal{I}_\approx$ returns p as an ANN to q, if and only if there are two consecutive indices, such that q is inside the union of the approximate balls of $\mathcal{N}_{i+1}$ but not inside the balls of $\mathcal{N}_i$. Thus, $r(1 + \varepsilon/16)^i \leq \mathbf{d}_P(q) \leq \mathbf{d}(p, q) \leq r(1 + \varepsilon/16)^{i+1}(1 + \varepsilon/16) \leq (1 + \varepsilon/4)r$. Thus p is indeed $(1 + \varepsilon/4)$-ANN. ■

Lemma 6.3.3 *Let P be a set of n points in a metric space $\mathcal{M}$, and let $\mathcal{B}$ be a set of balls with centers at P, computed by the algorithm of Lemma 6.2.11, such that one can answer $(1 + \varepsilon/16)$-ANN queries on P, by performing a single target query in $\mathcal{B}$.*

Let $\mathcal{B}_\approx$ be a $(1 + \varepsilon/16)$-approximation to $\mathcal{B}$. A target query on $\mathcal{B}_\approx$, for a query point q, returns a $(1 + \varepsilon)$-ANN to q in P.

Proof: Let $\mathcal{D}$ be the tree computed by Lemma 6.2.11. We prove the correctness of the algorithm by an inductive proof over the height of $\mathcal{D}$, similar in nature to the proofs of Lemma 6.2.11 and Theorem 6.2.10.

Indeed, for a node $v \in \mathcal{D}$, let $\mathcal{I}_{\approx v} = \mathcal{I}_\approx(P_v, r_v, R_v, \varepsilon/16)$ be the set of $(1 + \varepsilon/16)$-approximate balls of $\mathcal{B}_\approx$ that corresponds to the set of balls stored in $\widehat{\mathcal{I}}_v = \widehat{\mathcal{I}}(P_v, r_v, R_v, \varepsilon/4)$. If the algorithm stops at v, we know by Lemma 6.3.2 that we returned a $(1 + \varepsilon/4)$-ANN to q in P_v.

Otherwise, if we continued the search into the outer child of v using $\widehat{\mathcal{I}}_v$, then we would also continue the search into the this node when using $\mathcal{I}_{\approx v}$. As such, by induction the result is correct.

Otherwise, we continue the search into a node w, where $q \in \mathcal{U}_{\text{balls}}(P_w, (1 + \varepsilon/16)r_v)$. We observe that because of the factor 2 slackness of Eq. (6.1), we are guaranteed to continue the search in the right connected component of $\mathcal{U}_{\text{balls}}(P_v, \ell_v)$.

Thus, the quality of approximation argument of Lemma 6.2.11 and Theorem 6.2.10 still holds, and require easy adaption to this case (we omit the straightforward by tedios details). We conclude, that the result returned is correct. ■

6.4 Bibliographical notes

Finite metric spaces would receive more attention later in the course. HSTs, despite their simplicity are a powerful tool in giving a compact (but approximate) representation of metric spaces.

Indyk and Motwani observed that ANN can be reduced to small number of near neighbor queries (i.e., this is the PLEB data-structure of [IM98]). In particular, the elegant argument in Remark 6.2.3 is due to Indyk (personal communications). Indyk and Motwani also provided a rather involved reduction showing that a near linear number of balls is sufficient. A considerably simpler reduction was provided by Har-Peled [Har01], and we loosely follow his exposition in Section 6.2, although our bound on the number of balls is better by a logarithmic factor than the bounded provided by Har-Peled. This improvement was pointed out by Sabharwal *et al.* [SSS02]. They also provide a more involved construction, which achieves a linear dependency on n.

Open problems. The argument showing that one can use approximate near-neighbor data-structure instead of exact (i.e., Lemma 6.3.3), is tedious and far from being elegant; see Exercise 6.5.2. A natural question for further research, is to try and give a simple concrete condition on the set of balls, such that using approximate near neighbor data-structure still give the correct result. Curretly, it seems that what we need is somekind of separability property. However, it would be nice to give a simpelr direct condition.

As mentioned above, Sabharwal *et al.* [SSS02] showed a reduction from ANN to linear number of balls (ignoring the dependency on ε). However, it seems like a simpler construction should work.

6.5 Exercises

Exercise 6.5.1 [10 Points] Show, that by adversarial argument, that for any $t > 1$, we have: Any algorithm computing a HST $\mathcal{H}$ for n points in a metric space $\mathcal{M}$, that t approximates $\mathbf{d}_{\mathcal{M}}$, must in the worst case inspect all $\binom{n}{2}$ distances in the metric. Thus, computing a HST requires quadratic time in the worst case.

Exercise 6.5.2 Lemma 6.3.3 is not elegant. A more natural conjecture would be the following:

> **Conjecture 6.5.3** *Let P be a set of n points in the plane, and let $\mathcal{B}$ be a set of balls such that $(1+\varepsilon/\overline{c})$-ANN, let $\mathcal{B}_{\approx}$ be a $(1+\varepsilon/\overline{c})$-approximation to $\mathcal{B}$. Then a target query on $\mathcal{B}_{\approx}$, answers $(1+\varepsilon)$-ANN on P, for $\overline{c}$ large enough constant.*

Give a counter example to the above conjecture, showing that it is incorrect.

Chapter 7

Approximate Voronoi Diagrams

"She had given him a smile, first because that was more or less what she was there for, and then because she had never seen him before and she had a prejudice in favor of people she did not know."
– – The roots of heaven, Romain Gary

7.1 Introduction

A Voronoi diagram of a point set $P \subseteq \mathbb{R}^d$ is a partition of space into regions, such that the cell of $p \in P$, is $V(p, P) = \left\{ x \,\middle|\, \|xp\| \leq \|xp'\| \text{ for all } p' \in P \right\}$. Vornoi diagrams are a powerful tool and have numerous applications [Aur91].

One problem with Vornoi diagrams is that their descriptive complexity is $O(n^{\lceil d/2 \rceil})$ in $\mathbb{R}^d$, in the worst case. See Figure 7.1 for an exmaple in three dimensions. It is a natrual question to ask, whether one can reduce the complexity to linear (or near linear) by allow some approximation.

Definition 7.1.1 (Approximate Voronoi Diagram.) Given a set P of n points in $\mathbb{R}^d$, and parameter $\varepsilon > 0$, an $(1+\varepsilon)$-*approximated Voronoi diagram* of P, is a partition $\mathcal{V}$ of space into regions, such that for any region $\varphi \in \mathcal{V}$, there is an associated point $\mathrm{rep}_\varphi \in P$, such that for any $x \in \varphi$, we have that rep_φ is a $(1+\varepsilon)$-ANN for x in P. We will refer to $\mathcal{V}$ as $(1+\varepsilon)$-*AVD*.

7.2 Fast ANN in $\mathbb{R}^d$

In the following, assume P is a set of points contained in the hypercube $[0.5 - \varepsilon/d, 0.5 + \varepsilon/d]^d$. This can be easily guaranteed by affine linear transformation T which preserves relative distances (i.e., computing the ANN for q on P, is equivalent to computing the ANN for $T(q)$ on $T(P)$).

In particular, if the query point is outside the unit hypercube$\left[0, 1\right]^d$ then any point of P is $(1+\varepsilon)$-ANN, and we are done. Thus, we need to answer ANN only for points inside the unit hypercube.

We are going to use the reduction we saw between ANN and target queries among balls. In particular, one can compute the set of (exact) balls of Lemma 6.3.3 using the algorithm of Theorem 6.2.12. Let $\mathcal{B}$ be this set of balls. This takes $O(n\varepsilon^{-1}\log(n/\varepsilon))$ time. Next, we approximate each ball $\mathbf{b} \in \mathcal{B}$ of radius r, by the set of grid cells of G_{2^i} that intersects it, where $\sqrt{d}2^i \leq (\varepsilon/16)r$; namely, $i = \left\lfloor \lg(\varepsilon r/\sqrt{d}) \right\rfloor$. In particular, let $\mathbf{b}_\approx$ denote the region corresponding to the grid cells intersected by $\mathbf{b}$. Clearly, $\mathbf{b}_\approx$ is an approximate ball of $\mathbf{b}$.

Let $\mathcal{B}_\approx$ be the resulting set of approximate balls for $\mathcal{B}$, and let C' be the associated (multi) set of grid cells formed by those balls. The multi-set C' is a collection of canonical grid cells from different resolutions.

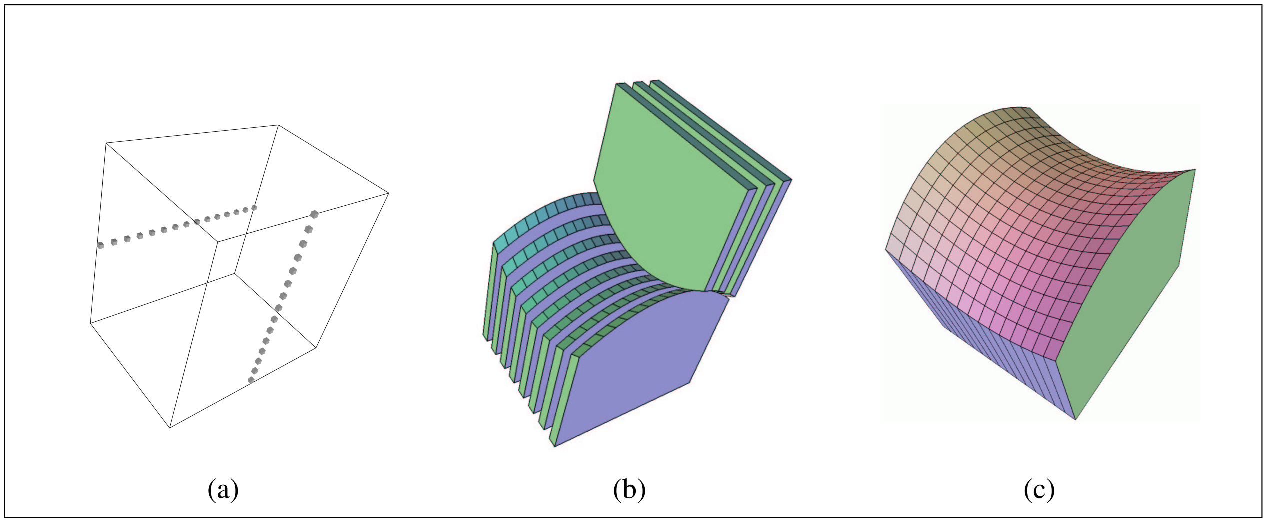


Figure 7.1: (a) The point-set in 3D inducting a Voronoi diagram of quadratic complexity. (b) Some cells in this Voronoi diagram. Note that the cells are thin and flat, and every cell from the lower part touches the cells on the upper part. (c) The contact surface between the two parts of the Voronoi diagram has quadratic complexity, and thus the Voronoi diagram itself has quadratic complexity.

We create a set out of C', by picking from all the instances of the same cell $\square \in C'$, the instance associated with the smallest ball of $\mathcal{B}$. Let C be the resulting set of canonical grid cells.

Thus, by Lemma 6.3.3, answering $(1+\varepsilon)$-ANN is no more than performing a target query on $\mathcal{B}_\approx$. This in turn, just finding the smallest canonical grid cell of C which contains the query point q. In previous lecture, we showed that one can construct a compressed quadtree $\widehat{T}$ that has all the nodes of C appear as nodes of $\widehat{T}$. The construction of this compressed quadtree takes $O(|C| \log |C|)$ time, by Lemma 2.2.4.

By performing a point-location query in $\widehat{T}$, we can compute the smallest grid cell of C that contains the query point in $O(\log |C|)$ time. Specificly, the query returns a leaf v of $\widehat{T}$. We need to find, along the path of v to the root, the smallest ball associated with those nodes. This information can be propogated down the compressed quadtree during the preprocessing stage, and as such, once we have v at hand, one can answer the ANN query in constant time. We conclude:

Theorem 7.2.1 *Let P be a set of n points in $\mathbb{R}^d$. One can build a compressed quadtree $\widehat{T}$, in $O(n\varepsilon^{-d} \log^2(n/\varepsilon))$ time, of size $O(n\varepsilon^{-d} \log(n/\varepsilon))$, such that $(1+\varepsilon)$-ANN query on P, can be answered by a single point-location query in $\widehat{T}$. Such a point-location query takes $O(\log(n/\varepsilon))$ time.*

Proof: The construction is described above. We only need to prove the bounds on the running time. It takes $O(n\varepsilon^{-1} \log(n/\varepsilon))$ time to compute $\mathcal{B}$. For every such ball, we generate $O(1/\varepsilon^d)$ canonical grid cells that cover it. Let C be the resulting set of grid cells (after filtering multiple instance of the same grid cell). Naively, the size of C is bounded by $O\Big(|\mathcal{B}| / \varepsilon^d\Big)$. However, it is easy to verify that $\mathcal{B}$ has a large number of balls of similar size centered at the same point (because the set $\widehat{\mathcal{I}}$ has a lot of such balls). In particular, if we have the set of balls $\widehat{\mathcal{I}}(\{p\}, r, 2r, \varepsilon/16)$, it requires only $O(1/\varepsilon^d)$ canonical grid cells to approximate it. Thus, we can bound $|C|$ by $N = O\Big(|\mathcal{B}| / \varepsilon^{d-1}\Big) = O(n\varepsilon^{-d} \log(n/\varepsilon))$. We can also compute C in this time, by careful implementation (we omit the straightforward and tedious details).

Constructing $\widehat{T}$ for C takes $O(N \log N) = O(n\varepsilon^{-d} \log^2(n/\varepsilon))$ time (Lemma 2.2.4). The resulting compressed quadtree is of size $O(N)$. Point location queries at $\widehat{T}$ takes $O(\log N) = O(\log(n/\varepsilon))$ time. Given a query point, and the leaf v, such that $q \in \square_v$, we need to find the first ancestor above v that has a point associated with it. This can be done in constant ti1me, by preprocessing $\widehat{T}$, by propagating down the compressed

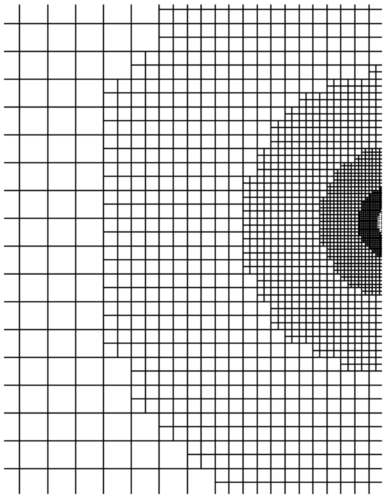


Figure 7.2: Exponential grid.

quadtree the nodes they are associated with. ■

7.3 A Direct Construction of AVD

Intuitively, constructing an approximate Voronoi diagram, can be interpret as a meshing problem. Indeed, consider the function $\mathbf{d}_P(q)$, for $q \in \mathbb{R}^d$,and a cell $\square \subseteq \mathbb{R}^d$ such that $\mathrm{diam}(\square) \leq (\varepsilon/4) \min_{q \in \square} \mathbf{d}_P(q)$. Since for any $x, y \in \mathbb{R}^d$, we have $\mathbf{d}_P(y) \leq \mathbf{d}_P(x) + \|xy\|$. It follows that

$$\forall x, y \in \square \quad \mathbf{d}_P(x) \leq (1 + \varepsilon/4)\mathbf{d}_P(y). \tag{7.1}$$

Namely, the distance function $\mathbf{d}_P(\cdot)$ is essentially a "constant" in such a cell. Of course, if there is only a unique nearest neighbor to all the points in $\square$, we do not need this condition.

Namely, we need to partition space into cells, such that for each cell, either it has a unique nearest neighbor (no problem), or alternatively, Eq. (7.1) holds in the cell. Unfortunately, forcing Eq. (7.1) globally is too expensive. Alternatively, we sill force Eq. (7.1) only in "critical" regions where there are two points which are close to being ANN to a cell.

The general idea, as before is to generate a set of canonical grid cells. We are guaranteed, that when generating the compressed quadtree for the point-set, and considering the space partition induced by the leafs, then necessarily those cells would be smaller than the input grid cells.

Definition 7.3.1 (Exponential Grid.) For a point $p \in \mathbb{R}^d$, and parameters r, R and $\varepsilon > 0$, let $\mathrm{G}_E(p, r, R, \varepsilon)$ denote an *exponential grid* centered at p.

let $\mathbf{b}_i = \mathbf{b}(p, r_i)$, for $i = 0, \ldots, \lceil \lg R/r \rceil$, where $r_i = r2^i$. Next, let G'_i be the set of cells of the canonical grid G_{α_i} that intersects $\mathbf{b}_i$, where $\alpha_i = 2^{\lfloor \lg(\varepsilon r_i/(16d)) \rfloor}$. Clearly, $\left|G'_i\right| = O(1/\varepsilon^d)$. We remove from G'_i all the canonical cells completely covered by cells of G_{i-1}. Similarly, for cells the that are partially covered in G'_i by cells in G'_{i-1}, we replace them by the cells covering them in $\mathrm{G}_{\alpha_{i-1}}$. Let G_i be the resulting set of canonical grid cells. And let $\mathrm{G}_E(p, r, R, \varepsilon) = \cup_i G_i$. We have $|\mathrm{G}_E(p, r, R, \varepsilon)| = O(\varepsilon^{-d} \log(R/r))$. Furthermore, it can be computed in linear time in its size; see Figure 7.2.

Let P be a set of n points, and let $0 < \varepsilon < 1/2$. As before, we assume that $P \subseteq [0.5 - \varepsilon/d, 0.5 + \varepsilon/d]^d$.

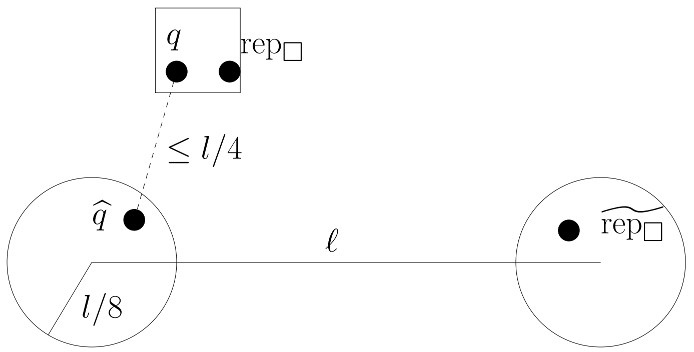


Figure 7.3: Illustration of the proof of Claim 7.3.2.

Compute a $(1/(8d))^{-1}$-WSPD $\mathcal{W}$ of P. Note that $|\mathcal{W}| = O(n)$. For every pair $X = \{u, v\} \in \mathcal{W}$, let $\ell_{uv} = \|uv\|$, and consider the set of canonical cells

$$W(u, v) = \mathrm{G}_E(\mathrm{rep}_u, \ell_{uv}/4, 4\ell_{uv}/\varepsilon, \varepsilon) \cup \mathrm{G}_E(\mathrm{rep}_v, \ell_{uv}/4, 4\ell_{uv}/\varepsilon, \varepsilon).$$

For a query point q, if $\mathbf{d}_P(q) \in [\ell_{uv}/4, \ell_{uv}/\varepsilon]$, and the nearest neighbor of q is in $P_u \cup P_v$, then the cell $\square \in W(u, v)$ containing q is of the right size, it comply with Eq. (7.1), and as such any $(1 + \varepsilon/4)$-ANN to any point of $\square$ is a $(1 + \varepsilon)$-ANN to q.

Thus, let $W = \cup_{\{u,v\}\in\mathcal{W}} W(u, v) \cup [0, 1]^{\mathsf{d}}$. Let $\widehat{T}$ be a compressed quadtree constructed so that it contains all the canonical nodes of W as nodes (we remind the reader that this can be done in $O(|W| \log |W|)$ time; see Lemma 2.2.4). Next, let U be the space decomposition induced by the leafs of $\widehat{T}$.

For each cell $\square \in U$, take an arbitrary point inside it $\mathrm{rep}_\square$, and compute a $(1 + \varepsilon/4)$-ANN to p. Let $\widetilde{\mathrm{rep}_\square} \in P$ be this ANN, and store it together with $\square$.

Claim 7.3.2 *The set U is a $(1 + \varepsilon)$-AVD.*

Proof: Let q be am arbitrary query point, and let $\square \in U$ be the cell containing q. Let $\mathrm{rep}_\square \in \square$ be its representative, and let $\widetilde{\mathrm{rep}_\square} \in P$ be the $(1+\varepsilon/4)$-ANN to $\mathrm{rep}_\square$ stored at $\square$. Also, let $\widehat{q}$ be the nearest neighbor of q in P. If $\widetilde{\mathrm{rep}_\square} = \widehat{q}$ then we are done. Otherwise, consider the pair $\{u, v\} \in \mathcal{W}$ such that $\widetilde{\mathrm{rep}_\square} \in P_u$ and $\widehat{q} \in P_v$. Finally, let $\ell = \left\|\widetilde{\mathrm{rep}_\square}\,\widehat{q}\right\|$.

If $\left\|q\widehat{q}\right\| > \ell/\varepsilon$ then $\widetilde{\mathrm{rep}_\square}$ is $(1 + \varepsilon)$-ANN since $\left\|q\,\widetilde{\mathrm{rep}_\square}\right\| \leq \left\|q\widehat{q}\right\| + \left\|\widehat{q}\,\widetilde{\mathrm{rep}_\square}\right\| \leq (1 + \varepsilon)\left\|q\widehat{q}\right\|$.

If $\left\|q\widehat{q}\right\| < \ell/4$, then by the construction of $W(u, v)$, we have that $\mathrm{diam}(\square) \leq (\varepsilon/16)\left\|\mathrm{rep}_u\,\mathrm{rep}_v\right\| \leq \varepsilon\ell/8$. This holds by the construction of the WSPD, where $\mathrm{rep}_u, \mathrm{rep}_v$ are the representative from the WSPD construction of P_u and P_v, respecitvely. See Figure 7.3. But then,

$$\left\|\widehat{q}\,\mathrm{rep}_\square\right\| \leq \left\|q\widehat{q}\right\| + \mathrm{diam}(\square) \leq \frac{\ell}{4} + \frac{\ell\varepsilon}{8} < \frac{5}{16}\ell,$$

for $\varepsilon \leq 1/2$. On the other hand,

$$\begin{aligned} \left\|\mathrm{rep}_\square\,\widetilde{\mathrm{rep}_\square}\right\| &\geq \left\|\mathrm{rep}_u\mathrm{rep}_v\right\| - \left\|\mathrm{rep}_v\widehat{q}\right\| - \left\|\widehat{q}q\right\| - \mathrm{diam}(\square) - \left\|\mathrm{rep}_u\widetilde{\mathrm{rep}_\square}\right\| \\ &\geq \ell - \ell/8 - \ell/4 - \varepsilon\ell/8 - \ell/8 \geq (7/16)\ell. \end{aligned}$$

But then, $(1 + \varepsilon/4)\left\|\mathrm{rep}_\square\widehat{q}\right\| \leq (5/16)(9/8)\ell < \ell/2 \leq \left\|\mathrm{rep}_\square\,\widetilde{\mathrm{rep}_\square}\right\|$. A contradiction, because $\widetilde{\mathrm{rep}_\square}$ is not $(1 + \varepsilon/4)$-ANN in P for $\mathrm{rep}_\square$. See Figure 7.3.

If $\left\| q\widetilde{q} \right\| \in [\ell/4, \ell/\varepsilon]$, then by the construction of $W(u,v)$, we have that $\mathrm{diam}(\square) \leq (\varepsilon/4)\mathbf{d}_{P_u \cup P_v}(q)$. We have $\mathbf{d}_P(z) \leq (1+\varepsilon/4)\mathbf{d}_P(q)$ and

$$\begin{aligned} \left\| q\, \widetilde{\mathrm{rep}_\square} \right\| &\leq \left\| q\, \mathrm{rep}_\square \right\| + \left\| \mathrm{rep}_\square\, \widetilde{\mathrm{rep}_\square} \right\| \leq \mathrm{diam}(\square) + (1+\varepsilon/4)\mathbf{d}_P(\mathrm{rep}_\square) \\ &\leq (\varepsilon/4)\mathbf{d}_P(q) + (1+\varepsilon/4)(1+\varepsilon/4)\mathbf{d}_P(q) \leq (1+\varepsilon)\mathbf{d}_P(q), \end{aligned}$$

as required. ■

Theorem 7.3.3 *Given a set P of n points in $\mathbb{R}^d$ one can compute, in $O(n/\varepsilon^d \log(1/\varepsilon)(\varepsilon^{-d} + \log(n/\varepsilon)))$ time, a $(1+\varepsilon)$-AVD of P. The AVD is of complexity $O(n\varepsilon^{-d}\log(1/\varepsilon))$.*

Proof: The total number of cubes in W is $O(n\varepsilon^{-d}\log(1/\varepsilon))$, and W can also be computed in this time, as described above. For each node of W we need to perform a $(1+\varepsilon/4)$-ANN query on P. After $O(n\log n)$ preprocessing, such queries can be answered in $O(\log n + 1/\varepsilon^d)$ time (Theorem 5.4.2). Thus, we can answer those queries, and built the overall data-structure, in the time stated in the theorem. ■

7.4 Bibliographical notes

The realization that point-location among approximate balls is sufficient for ANN, was realized by Indyk and Motwani [IM98]. The idea of first building a global set of balls, and using a compressed quadtree on the associated set of approximate balls, is due to Har-Peled [Har01]. The space used by the data-structure of Theorem 7.2.1 was further improved (by a logarithmic factor) by Sabharwal *et al.* [SSS02] and Arya and Malamatos [AM02]. As mentioned before, Sabharwal *et al.* improved the number of balls needed (for the general metric case), while Arya and Malamatos showed a direct construction for the low-dimensional euclidean space using $O(n/\varepsilon^d)$ balls.

The direct construction (Section 7.3) of AVD is due to Arya and Malamatos [AM02]. The reader might wonder why we went through the excruciating pain of first approximating the metric by balls, and then using it to construct AVD. The reduction to point location among approximate balls, would prove to be useful when dealing with proximity in high dimensions. Of course, once we have the fact about ANN via point-location among approximate balls, the AVD construction is relatively easy.

Chapter 8

The Johnson-Lindenstrauss Lemma

8.1 The Brunn-Minkowski inequality

Definition 8.1.1 For two sets A and B in $\mathbb{R}^n$, let $A + B$ denote the *Minkowski sum* of A and B. Formally,

$$A + B = \left\{ a + b \,\middle|\, a \in A, b \in B \right\}.$$

It is easy to verify that if A', B' are translated copies of A, B respectively, then $A' + B'$ is a translated copy of $A + B$.

Theorem 8.1.2 (Brunn-Minkowski inequality) *Let A and B be two non-empty compact sets in* $\mathbb{R}^n$. *Then*

$$\mathrm{Vol}(A + B)^{1/n} \geq \mathrm{Vol}(A)^{1/n} + \mathrm{Vol}(B)^{1/n}$$

Definition 8.1.3 A set $A \subseteq \mathbb{R}^n$ is a *brick set* if it is the union of finitely many (close) axis parallel boxes with disjoint interiors.

It is intuitively clear, by limit arguments, that proving the claim for brick sets will imply Theorem 8.1.2.

Lemma 8.1.4 (Brunn-Minkowski inequality for Brick Sets) *Let A and B be two non-empty brick sets in* $\mathbb{R}^n$. *Then*

$$\mathrm{Vol}(A + B)^{1/n} \geq \mathrm{Vol}(A)^{1/n} + \mathrm{Vol}(B)^{1/n}$$

Proof: By induction on the number k of bricks in A and B. If $k = 2$ then A and B are just bricks, with dimensions $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$, respectively. In this case, the dimensions of $A+B$ are $a_1+b_1, \ldots, a_n+b_n$, as can be easily verified. Thus, we need to prove that $\left(\prod_{i=1}^n a_i\right)^{1/n} + \left(\prod_{i=1}^n b_i\right)^{1/n} \leq \left(\prod_{i=1}^n (a_i + b_i)\right)^{1/n}$. Dividing the left side by the right side, we have

$$\left(\prod_{i=1}^n \frac{a_i}{a_i + b_i}\right)^{1/n} + \left(\prod_{i=1}^n \frac{b_i}{a_i + b_i}\right)^{1/n} \leq \frac{1}{n}\sum_{i=1}^n \frac{a_i}{a_i + b_i} + \frac{1}{n}\sum_{i=1}^n \frac{b_i}{a_i + b_i} = 1,$$

by the generalized arithmetic-geometric mean inequality[②], and the claim follows for this case.

[②]Here is a proof of this generalized form: Let $x_1, \ldots, x_n$ be n positive real numbers. Consider the quantity $R = x_1 x_2 \cdots x_n$. If we fix the sum of the n numbers to be equal α, then R is maximized when all the x_is are equal. Thus, $\sqrt[n]{x_1 x_2 \cdots x_n} \leq \sqrt[n]{(\alpha/n)^n} = \alpha/n = (x_1 + \cdots + x_n)/n$.

Now let $k > 2$ and suppose that the Brunn-Minkowski inequality holds for any pair of brick sets with fewer than k bricks (together). Let A, B be a pair of sets having k bricks together, and A has at least two (disjoint) bricks. However, this implies that there is an axis parallel hyperplane h that separates between the interior of one brick of A and the interior of another brick of A (the hyperplane h might intersect other bricks of A). Assume that h is the hyperplane $x_1 = 0$ (this can be achieved by translation and renaming of coordinates).

Let $\overline{A^+} = A \cap h^+$ and $\overline{A^-} = A \cap h^-$, where h^+ and h^- are the two open half spaces induced by h. Let A^+ and A^- be the closure of $\overline{A^+}$ and $\overline{A^-}$, respectively. Clearly, A^+ and A^- are both brick sets with (at least) one fewer brick than A.

Next, observe that the claim is translation invariant, and as such, let us translate B so that its volume is split by h in the same ratio A's volume is being split. Denote the two parts of B by B^+ and B^-, respectively. Let $\rho = \mathrm{Vol}(A^+)/\,\mathrm{Vol}(A) = \mathrm{Vol}(B^+)/\,\mathrm{Vol}(B)$ (if $\mathrm{Vol}(A) = 0$ or $\mathrm{Vol}(B) = 0$ the claim trivially holds).

Observe, that $A^+ + B^+ \subseteq A + B$, and it lies on one side of h, and similarly $A^- + B^- \subseteq A + B$ and it lies on the other side of h. Thus, by induction, we have

$$\begin{aligned}
\mathrm{Vol}(A+B) &\geq \mathrm{Vol}(A^+ + B^+) + \mathrm{Vol}(A^- + B^-)\\
&\geq \left(\mathrm{Vol}(A^+)^{1/n} + \mathrm{Vol}(B^+)^{1/n}\right)^n + \left(\mathrm{Vol}(A^-)^{1/n} + \mathrm{Vol}(B^-)^{1/n}\right)^n\\
&= \left[\rho^{1/n}\,\mathrm{Vol}(A)^{1/n} + \rho^{1/n}\,\mathrm{Vol}(B)^{1/n}\right]^n\\
&\quad + \left[(1-\rho)^{1/n}\,\mathrm{Vol}(A)^{1/n} + (1-\rho)^{1/n}\,\mathrm{Vol}(B)^{1/n}\right]^n\\
&= (\rho + (1-\rho))\left[\mathrm{Vol}(A)^{1/n} + \mathrm{Vol}(B)^{1/n}\right]^n\\
&= \left[\mathrm{Vol}(A)^{1/n} + \mathrm{Vol}(B)^{1/n}\right]^n,
\end{aligned}$$

establishing the claim. ■

Proof of Theorem 8.1.2: Let $A_1 \subseteq A_2 \subseteq \cdots A_i \subseteq$ be a sequence of brick sets, such that $\bigcup_i A_i = A$, and similarly let $B_1 \subseteq B_2 \subseteq \cdots B_i \subseteq \cdots$ be a sequence of brick sets, such that $\bigcup_i B_i = B$. It is well known fact in measure theory, that $\lim_{i\to\infty} \mathrm{Vol}(B_i) = \mathrm{Vol}(B)$ and $\lim_{i\to\infty}\ Vol(B_i) = \mathrm{Vol}(B)$.

We claim that $\lim_{i\to\infty} \mathrm{Vol}(A_i + B_i) = \mathrm{Vol}(A + B)$. Indeed, consider any point $z \in A + B$, and let $u \in A$ and $v \in B$ be such that $u + v = z$. By definition, there exists i, such that for all $j > i$ we have $u \in A_j$, $v \in B_j$, and as such $z \in A_i + B_i$. Thus, $\cup_i (A_i + B_i) = A + B$.

Furthermore, for any $i > 0$, since A_i and B_i are brick sets, we have

$$\mathrm{Vol}(A_i + B_i)^{1/n} \geq \mathrm{Vol}(A_i)^{1/n} + \mathrm{Vol}(B_i)^{1/n},$$

by Lemma 8.1.4. Thus,

$$\begin{aligned}
\mathrm{Vol}(A+B) &= \lim_{i\to\infty} \mathrm{Vol}(A_i + B_i)^{1/n} \geq \lim_{i\to\infty}\left(\mathrm{Vol}(A_i)^{1/n} + \mathrm{Vol}(B_i)^{1/n}\right)\\
&= \mathrm{Vol}(A)^{1/n} + \mathrm{Vol}(B)^{1/n}.
\end{aligned}$$

■

Theorem 8.1.5 (Brunn-Minkowski for slice volumes.) *Let $\mathcal{P}$ be a convex set in $\mathbb{R}^{n+1}$, and let $A = \mathcal{P} \cap (x_1 = a)$, $B = \mathcal{P} \cap (x_1 = b)$ and $C = \mathcal{P} \cap (x_1 = c)$ be three slices of A, for $a < b < c$. We have* $\mathrm{Vol}(B) \geq \min(\mathrm{Vol}(A), \mathrm{Vol}(C))$.

In fact, consider the function

$$v(t) = \left(\mathrm{Vol}(\mathcal{P} \cap (x_1 = t))\right)^{1/n},$$

and let $I = [t_{min}, t_{max}]$ be the interval where the hyperplane $x_1 = t$ intersects $\mathcal{P}$. Then, $v(t)$ is concave in I.

Proof: If a or c are outside (t_{min}, t_{max}), then $\mathrm{Vol}(A) = 0$ or $\mathrm{Vol}(C) = 0$, respectively, and then the claim trivially holds.

Otherwise, let $\alpha = (b - a)/(c - a)$. We have that $b = (1 - \alpha) \cdot a + \alpha \cdot c$, and by the convexity of $\mathcal{P}$, we have $(1 - \alpha)A + \alpha C \subseteq B$. Thus, by Theorem 8.1.2 we have

$$\begin{aligned} v(b) = \mathrm{Vol}(B)^{1/n} &\geq \mathrm{Vol}((1-\alpha)A + \alpha C)^{1/n} \geq \mathrm{Vol}((1-\alpha)A)^{1/n} + \mathrm{Vol}(\alpha C)^{1/n} \\ &= (1-\alpha) \cdot \mathrm{Vol}(A)^{1/n} + \alpha \cdot \mathrm{Vol}(C)^{1/n} \\ &\geq (1-\alpha)v(a) + \alpha \cdot v(c). \end{aligned}$$

Namely, $v(\cdot)$ is concave on $[t_{min}, t_{max}]$. In particular, $v(b) \geq \min(v(a), v(c))$, which in turn implies that $\mathrm{Vol}(B) = v(b)^n \geq \min(\mathrm{Vol}(A), \mathrm{Vol}(B))$. ■

Corollary 8.1.6 *For A and B compact sets in* $\mathbb{R}^n$*, we have* $\mathrm{Vol}((A + B)/2) \geq \sqrt{\mathrm{Vol}(A)\,\mathrm{Vol}(B)}$.

Proof: $\mathrm{Vol}((A + B)/2)^{1/n} = \mathrm{Vol}(A/2 + B/2) \geq \mathrm{Vol}(A/2)^{1/n} + \mathrm{Vol}(B/2)^{1/n} = (\mathrm{Vol}(A) + \mathrm{Vol}(B))/2 \geq \sqrt{\mathrm{Vol}\,A\,\mathrm{Vol}\,B}$ by Theorem 8.1.2, and since $(a + b)/2 \geq \sqrt{ab}$ for any $a, b \geq 0$. ■

8.2 Measure Concentration on the Sphere

Let $\mathbb{S}^{(n-1)}$ be the unit sphere in $\mathbb{R}^n$. We assume there is a uniform probability measure defined over $\mathbb{S}^{(n-1)}$, such that its total measure is 1. Surprisingly, most of the mass of this measure is near the equator. In fact, as the dimension increases, the width of the strip around the equator $(x_1 = 0) \cap \mathbb{S}^{(n-1)}$ containing, say, 90% of the measure is of width $\approx 1/\sqrt{n}$. Counter intuitively, this is true for any equator. We are going to show that in fact a stronger result holds: The mass is concentrated close to the boundary of any set $A \subseteq \mathbb{S}^{(n-1)}$ such that $\mathbf{Pr}[A] = 1/2$.

Theorem 8.2.1 (Measure concentration on the sphere.) *Let* $A \subseteq \mathbb{S}^{(n-1)}$ *be a measurable set with* $\mathbf{Pr}[A] \geq 1/2$*, and let* A_t *denote the set of points of* $\mathbb{S}^{(n-1)}$ *in distance at most t from A. Then* $1 - \mathbf{Pr}[A_t] \leq 2\exp\left(-tn^2/2\right)$

Proof: We will prove a slightly weaker bound, with $-tn^2/4$ in the exponent. Let $\widehat{A} = \left\{\alpha x \,\middle|\, x \in A, \alpha in [0,1]\right\} \subseteq \mathbf{b}^n$, where $\mathbf{b}^n$ is the unit ball in $\mathbb{R}^n$. We have that $\mathbf{Pr}[A] = \mu\left(\widehat{A}\right)$, where $\mu\left(\widehat{A}\right) = \mathrm{Vol}(\widehat{A}) / \mathrm{Vol}(\mathbf{b}^n)$.

Let $B = \mathbb{S}^{(n-1)} \setminus A_t$. We have that $\|a - b\| \geq t$ for all $a \in A$ and $b \in B$.

Lemma 8.2.2 *For any* $\widehat{a} \in \widehat{A}$ *and* $\widehat{b} \in \widehat{B}$*, we have* $\left\|\frac{a+b}{2}\right\| \leq 1 - t^2/8$.

Proof: Let $\widehat{a} = \alpha a$ and $\widehat{b} = \beta b$, where $a \in A$ and $binB$.

$$\left\|\frac{a+b}{2}\right\| \leq \sqrt{1 - \frac{t^2}{4}} \leq 1 - \frac{t^2}{8}.$$

When passing to $\widehat{a}$ and $\widehat{b}$, we can assume that $\beta = 1$. As such

$$\begin{aligned} \left\|\frac{\widehat{a} + \widehat{b}}{2}\right\| &= \left\|\frac{\alpha a + b}{2}\right\| \leq \left\|\frac{\alpha(a+b)}{2}\right\| + \left\|(1-\alpha)\frac{b}{2}\right\| \\ &\leq \alpha(1 - t^2/8) + (1-\alpha)(1 - 1/2) \leq 1 - \frac{t^2}{8}. \end{aligned}$$

■

By the Lemma, the set $(\widehat{A}+\widehat{B})/2$ is contained in a ball of radius $\leq 1 - t^2/8$ around the origin. Applying the Brunn-Minkowski inequality in the form of Corollary 8.1.6, we have

$$\left(1-\frac{t^2}{8}\right)^n \geq \mu\left(\frac{\widehat{A}+\widehat{B}}{2}\right) \geq \sqrt{\mu\left(\widehat{A}\right)\mu\left(\widehat{B}\right)} = \sqrt{\mathbf{Pr}[A]\,\mathbf{Pr}[B]} \geq \sqrt{\mathbf{Pr}[B]/2}.$$

Thus, $\mathbf{Pr}[B] \leq 2(1-t^2/8)^{2n} \leq 2\exp(-2nt^2/8)$, since $1 - x \leq \exp(-x)$, for $x \geq 0$. ∎

8.3 Concentration of Lipshitz Functions

Consider a function $f : \mathbb{S}^{(n-1)} \to \mathbb{R}$. Furthermore, imagine that we have a probability density define over the sphere. Let $\mathbf{Pr}[f \leq t] = \mathbf{Pr}\left[\left\{x \in S^{n-1} \;\middle|\; f(x) \leq t\right\}\right]$. We define the *median* of f, denoted by $\mathrm{med}(f)$, to be the $\sup t$, such that $\mathbf{Pr}[f \leq t] \leq 1/2$.

Lemma 8.3.1 *Let* $\mathbf{Pr}[f < \mathrm{med}(f)] \leq 1/2$ *and* $\mathbf{Pr}[f > \mathrm{med}(f)] \leq 1/2$.

Proof: Since $\bigcup_{k\geq 1}(-\infty, \mathrm{med}(f) - 1/k] = (-\infty, \mathrm{med}(f))$, we have

$$\mathbf{Pr}[f < \mathrm{med}(f)] \quad = \quad \sup_{k\geq 1} \mathbf{Pr}\left[f \leq \mathrm{med}(f) - \frac{1}{k}\right] \leq \frac{1}{2}.$$

∎

Definition 8.3.2 A function $f : A \to B$ is $\overline{c}$*-Lipschitz* if, for any $x, y \in \mathbb{S}^{(n-1)}$, we have$\|f(x) - f(y)\| \leq \overline{c}\|xy\|$.

Theorem 8.3.3 (Lévy's Lemma.) *Let* $f : \mathbb{S}^{(n-1)} \to \mathbb{R}$ *be* 1*-Lipschitz. Then for all* $t \in [0, 1]$,

$$\mathbf{Pr}[f > \mathrm{med}(f) + t] \leq 2\exp\left(-t^2 n/2\right) \text{ and } \mathbf{Pr}[f < \mathrm{med}(f) - t] \leq 2\exp\left(-t^2 n/2\right).$$

Proof: We prove only the first inequality, the second follows by symmetry. Let $A = \left\{x \in \mathbb{S}^{(n-1)} \;\middle|\; f(x) \leq \mathrm{med}(f)\right\}$. By Lemma 8.3.1, we have $\mathbf{Pr}[A] \geq 1/2$. Since f is 1-Lipschitz, we have $f(x) \leq \mathrm{med}(f) + t$, for $x \in A_t$. Thus, by Theorem 8.2.1, we get $\mathbf{Pr}[f > \mathrm{med}(f) + t] \leq 1 - \mathbf{Pr}[A_t] \leq 2\exp\left(-t^2 n/2\right)$. ∎

8.4 The Johnson-Lindenstrauss Lemma

Lemma 8.4.1 *For a unit vector* $x \in \mathbb{S}^{(n-1)}$, *let*

$$f(x) = \sqrt{\sum_{i=1}^{k} x_1^2 + x_2^2 + \cdots + x_k^2}$$

be the length of the projection of x *into the subspace formed by the first* k *coordinates. Let* x *be a vector randomly chosen with uniform distribution from* $\mathbb{S}^{(n-1)}$. *Then* $f(x)$ *is sharply concentrated. Namely, there exists* $m = m(n, k)$ *such that*

$$\mathbf{Pr}[f(x) \geq m + t] \leq 2\exp(-t^2 n/2) \text{ and } \mathbf{Pr}[f(x) \leq m - t] \leq 2\exp(-t^2 n/2).$$

Furthermore, for $k \geq 10 \ln n$, *we have* $m \geq \frac{1}{2}\sqrt{k/n}$.

Proof: The orthogonal projection $p : \ell_2^n \to \ell_2^k$ given by $p(x_1, \ldots, x_n) = (x_1, \ldots, x_k)$ is 1-Lipshitz, and so f is 1-Lipshitz, since for any x, y we have $|f(x) - f(y)| \leq \|\|f(x)0\| - \|f(y)0\|\| \leq \|f(x)f(y)\| \leq \|xy\|$. Theorem 8.3.3 (i.e., Lévy's lemma) gives the tail estimate with $m$med(f). Thus, we only need to prove the lower bound on m.

For a random $x \in \mathbb{S}^{(n-1)}$, we have $\mathbf{E}\Big[\|x\|^2\Big] = 1$. By linearity of expectations, and symmetry, we have $1 = \mathbf{E}\Big[\|x\|^2\Big] = \mathbf{E}\Big[\sum_{i=1}^n x_i^2\Big] = \sum_{i=1}^n \mathbf{E}\Big[x_i^2\Big] = n\,\mathbf{E}\Big[x_j^2\Big]$, for any $1 \leq j \leq n$. Thus, $\mathbf{E}\Big[x_j^2\Big] = 1/n$, for $j = 1, \ldots, n$. Thus, $\mathbf{E}\Big[(f(x))^2\Big] = k/n$. We next use the fact that f is concentrated, to show that f^2 is also relatively concentrated.

For any $t \geq 0$, we have

$$\frac{k}{n} = \mathbf{E}\Big[f^2\Big] \leq \mathbf{Pr}[f \leq m + t]\,(m+t)^2 + \mathbf{Pr}[f \geq m + t] \cdot 1 \leq 1 \cdot (m+t)^2 + 2\exp(-t^2 n/2),$$

since $f(x) \leq 1$, for any $x \in \mathbb{S}^{(n-1)}$. Let $t = \sqrt{k/5n}$. Since $k \geq 10 \ln n$, we have that $2\exp(-t^2 n/2) \leq 2/n$. We get that $\frac{k}{n} \leq (m + k/5n)^2 + 2/n$. Implying that $\sqrt{(k-2)/n} \leq m + k/5n$, which in turn implies that $m \geq \sqrt{(k-2)/n} - k/5n \geq \frac{1}{2}\sqrt{k/n}$. ■

At this point, we would like to flip Lemma 8.4.1 around, and instead of randomly picking a point and projecting it down to the first k-dimensional space, we would like x to be fixed, and randomly pick the k-dimensional subspace. However, we need to pick this k-dimensional space carefully, so that if we rotate this random subspace, by a transformation T, so that it occupies the first k dimensions, then the point $T(x)$ is uniformly distributed on the hypersphere.

To this end, we would like to randomly pick a random rotation of $\mathbb{R}^n$. This is an orthonormal matrix with determinant 1. We can generate such a matrix, by randomly picking a vector $e_1 \in \mathbb{S}^{(n-1)}$. Next, we set e_1 is the first column of our rotation matrix, and generate the other $n-1$ columns, by generating recursively $n-1$ orthonormal vectors in the space orthogonal to e_1.

Generating a random vector from the unit hypersphere, and a random rotation. At this point, the reader might wonder how do we pick a point uniformly from the unit hypersphere. The idea is to pick a point from the multi-dimensional normal distribution $N^d(0,1)$, and normalizing it to have length 1. Since the multi-dimensional normal distribution has the density function

$$(2\pi)^{-n/2} \exp\Big(-(x_1^2 + x_2^2 + \cdots + x_n^2)/2\Big),$$

which is symmetric (i.e., all the points in distance r from the origin has the same distribution), it follows that this indeed randomly generates a point randomly and uniformly on $\mathbb{S}^{(n-1)}$.

Generating a vector with multi-dimensional normal distribution, is no more than picking each coordinate according to the normal distribution. Given a source of random numbers according to the uniform distribution, this can be done using a $O(1)$ computations, using the Box-Muller transformation [BM58].

Since projecting down n-dimensional normal distribution to the lower dimensional space yields a normal distribution, it follows that generating a random projection, is no more than randomly picking n vectors according to the multidimensional normal distribution $v_1, \ldots, v_n$. Then, we orthonormalize them, using Graham-Schmidt, where $\widehat{v_1} = v_1/\|v_1\|$, and $\widehat{v_i}$ is the normalized vector of $v_i - w_i$, where w_i is the projection of v_i to the space spanned by $v_1, \ldots, v_{i-1}$.

Taking those vectors as columns of a matrix, generates a matrix A, with determinant either 1 or -1. We multiply one of the vectors by -1 if the determinant is -1. The resulting matrix is a random rotation matrix.

Theorem 8.4.2 (Johnson-Lindenstrauss lemma.) *Let X be an n-point set in a Euclidean space, and let $\varepsilon \in (0,1]$ be given. Then there exists a $(1+\varepsilon)$-embedding of X into $\mathbb{R}^k$, where $k = O(\varepsilon^{-2} \log n)$.*

Proof: Let $X \subseteq \mathbb{R}^n$ (if X lies in higher dimensions, we can consider it to be lying in the span of its points, if it is in lower dimensions, we can add zero coordinates). Let $k = 200\varepsilon^{-2} \ln n$. Assume $k < n$, and let L be a random k-dimensional linear subspace of $\mathbb{R}^n$. Let $p : \mathbb{R}^n \to L$ be the orthogonal projection operator. Let m be the number around which$\|p(x)\|$ is concentrated, for $x \in \mathbb{S}^{(n-1)}$, as in Lemma 8.4.1.

Fix two points $x, y \in \mathbb{R}^n$, we prove that

$$\left(1 - \frac{\varepsilon}{3}\right) m\|x - y\| \leq \|p(x) - p(y)\| \leq \left(1 + \frac{\varepsilon}{3}\right) m\|x - y\|$$

holds with probability $\geq 1 - n^{-2}$. Since there are $\binom{n}{2}$ pairs of points in X, it follows that with constant probability this holds for all pair of points of X. In such a case, the mapping p is D-embedding of X into $\mathbb{R}^k$ with $D \leq \frac{1+\varepsilon/3}{1-\varepsilon/3} \leq 1 + \varepsilon$, for $\varepsilon \leq 1$.

Let $u = x - y$, we have $p(u) = p(x) - p(y)$ since $p(\cdot)$ is a linear operator. Thus, the condition becomes $\left(1 - \frac{\varepsilon}{3}\right) m\|u\| \leq \|p(u)\| \leq \left(1 + \frac{\varepsilon}{3}\right) m\|u\|$. Since this condition is scale independent, we can assume $\|u\| = 1$. Namely, we need to show that

$$\left|\|p(u)\| - m\right| \leq \frac{\varepsilon}{3} m.$$

Lemma 8.4.1 (exchanging the random space with the random vector) implies, for $t = \frac{\varepsilon}{3} m$, that this is bounded by

$$4\exp(-t^2 n/2) = 4\exp\left(\frac{-\varepsilon^2 m^2 n}{18}\right) \leq 4\exp\left(-\frac{\varepsilon^2 k}{72}\right) < n^{-2},$$

since $m \geq \frac{1}{2}\sqrt{k/n}$. ■

8.5 An alternative proof of the Johnson-Lindenstrauss lemma

8.5.1 Some Probability

Definition 8.5.1 Let $N(0,1)$ denote the one dimensional *normal distribution*. This distribution has density $n(x) = e^{-x^2/2}/\sqrt{2\pi}$.

Let $N^d(0,1)$ denote the d-dimensional Gaussian distribution, induced by picking each coordinate independently from the standard normal distribution $N(0,1)$.

Let $\mathrm{Exp}(\lambda)$ denote the *exponential distribution*, with parameter λ. The density function of the exponential distribution is $f(x) = \lambda \exp(-\lambda x)$.

Let $\Gamma_{\lambda,k}$ denote the *gamma distribution*, with parameters λ and k. The density function of this distribution is $g_{\lambda,k}(x) = \lambda \frac{(\lambda x)^{k-1}}{(k-1)!} \exp(-\lambda x)$. The cumulative distribution function of $\Gamma_{\lambda,k}$ is $G_{\lambda,k}(x) = 1 - \exp(-\lambda x)\left(1 + \frac{\lambda x}{1!} + \cdots + \frac{(\lambda x)^i}{i!} + \cdots + \frac{(\lambda x)^{k-1}}{(k-1)!}\right)$. As we prove below, gamma distribution is how much time one has to wait till k experiments succeed, where an experiment duration distributes according to the exponential distribution.

A random variable X has the *Poisson distribution*, with parameter $\eta > 0$, which is a discrete distribution, if $\mathbf{Pr}[X = i] = e^{-\eta} \frac{\eta^i}{i!}$.

Lemma 8.5.2 *The following properties hold for the d dimensional Gaussian distribution $N^d(0,1)$:*

(i) The distribution $N^d(0,1)$ is centrally symmetric around the origin.

(ii) If $X \sim N^d(0,1)$ and u is a unit vector, then $X \cdot u \sim N(0,1)$.

(iii) If $X, Y \sim N(0,1)$ are two independent variables, then $Z = X^2 + Y^2$ follows the exponential distribution with parameter $\lambda = \frac{1}{2}$.

(iv) Given k independent variables $X_1, \ldots, X_k$ distributed according to the exponential distribution with parameter λ, then $Y = X_1 + \cdots + X_k$ is distributed according to the Gamma distribution $\Gamma_{\lambda,k}(x)$.

Proof: (i) Let $x = (x_1, \ldots, x_d)$ be a point picked from the Gaussian distribution. The density $\phi_d(x) = \phi(x_1)\phi(x_2)\cdot\phi(x_d)$, where $\phi(x_i)$ is the normal distribution density function, which is $\phi(x_i) = \exp(-x_i^2/2)/\sqrt{2\pi}$. Thus $\phi_d(x) = (2\pi)^{-n/2}\exp(-(x_1^2 \cdots + x_d^2)/2)$. Consider any two points $x, y \in \mathbb{R}^n$, such that $r = \|x\| = \|y\|$. Clearly, $\phi_d(x) = \phi_d(y)$. Namely, any two points of the same distance from the origin, have the same density (i.e., "probability"). As such, the distribution $N^d(0,1)$ is centrally symmetric around the origin.

(ii) Consider $e_1 = (1, 0, \ldots, 0) \in \mathbb{R}^n$. Clearly, $x \cdot e_1 = x_1$, which is distributed $N(0,1)$. Now, by the symmetry of of $N^d(0,1)$, this implies that $x \cdot u$ is distributed $N(0,1)$. Formally, let R be a rotation matrix that maps u to e_1. We know that Rx is distributed $N^d(0,1)$ (since $N^d(0,1)$ is centrally symmetric). Thus $x \cdot u$ has the same distribute as $Rx \cdot Ru$, which has the same distribution as $x \cdot e_1$, which is $N(0,1)$.

(iii) If $X, Y \sim N(0,1)$, and consider the integral of the density function

$$A = \int_{x=-\infty}^{\infty} \int_{y=-\infty}^{\infty} \frac{1}{2\pi} \exp\left(-\frac{x^2+y^2}{2}\right) dx\, dy.$$

We would like to change the integration variables to $x(r,\alpha) = \sqrt{r}\sin\alpha$ and $y(r,\alpha) = \sqrt{r}\cos\alpha$. The Jacobian of this change of variables is

$$I(r,\alpha) = \begin{vmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \alpha} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \alpha} \end{vmatrix} = \begin{vmatrix} \frac{\sin\alpha}{2\sqrt{r}} & \sqrt{r}\cos\alpha \\ \frac{\cos\alpha}{2\sqrt{r}} & -\sqrt{r}\sin\alpha \end{vmatrix} = -\frac{1}{2}\left(\sin^2\alpha + \cos^2\alpha\right) = -\frac{1}{2}.$$

As such, we have

$$\begin{aligned} \mathbf{Pr}[Z = z] &= \int_{x^2+y^2=\alpha} \frac{1}{2\pi}\exp\left(-\frac{x^2+y^2}{2}\right) \\ &= \int_{\alpha=0}^{2\pi} \frac{1}{2\pi}\exp\left(-\frac{x(\sqrt{z},\alpha)^2 + y(\sqrt{z},\alpha)^2}{2}\right) \cdot |I(r,\alpha)| \\ &= \frac{1}{2\pi}\cdot\frac{1}{2}\cdot\int_{\alpha=0}^{2\pi}\exp\left(-\frac{z}{2}\right) = \frac{1}{2}\exp\left(-\frac{z}{2}\right). \end{aligned}$$

As such, Z has an exponential distribution with $\lambda = 1/2$.

(iv) For $k = 1$ the claim is trivial. Otherwise, let $g_{k-1}(x) = \lambda\frac{(\lambda x)^{k-2}}{(k-2)!}\exp(-\lambda x)$. Observe that

$$\begin{aligned} g_k(t) &= \int_0^t g_{k-1}(t-x)g_1(x)\,dx = \int_0^t \left(\lambda\frac{(\lambda(t-x))^{k-2}}{(k-2)!}\exp(-\lambda(t-x))\right)(\lambda\exp(-\lambda x))\,dx \\ &= \int_0^t \lambda^2\frac{(\lambda(t-x))^{k-2}}{(k-2)!}\exp(-\lambda t)\,dx \\ &= \lambda\exp(-\lambda t)\int_0^t \lambda\frac{(\lambda x)^{k-2}}{(k-2)!}\,dx = \lambda\exp(-\lambda t)\frac{(\lambda t)^{k-1}}{(k-1)!} = g_k(x). \end{aligned}$$

■

8.5.2 Proof of the Johnson-Lindenstrauss Lemma

Lemma 8.5.3 *Let u be a unit vector in $\mathbb{R}^d$. For any even positive integer k, let $U_1, \ldots, U_k$ be random vectors chosen independently from the d-dimensional Gaussian distribution $N^d(0,1)$. For $X_i = u \cdot U_i$, define $W = W(u) = (X_1, \ldots, X_k)$ and $L = L(u) = \|W\|^2$. Then, for any $\beta > 1$, we have:*

1. $\mathbf{E}[L] = k$.

2. $\mathbf{Pr}[L \geq \beta k] \leq \frac{k+3}{2}\exp\left(-\frac{k}{2}(\beta - (1 + \ln\beta))\right)$.

3. $\mathbf{Pr}[L \leq k/\beta] < O(k) \times \exp(-\frac{k}{2}(\beta^{-1} - (1 - \ln\beta)))$.

Proof: By Lemma 8.5.2 (ii) each X_i is distributed as $N(0,1)$, and $X_1,\ldots,X_k$ are independent. Define $Y_i = X_{2i-1}^2 + X_{2i}^2$, for $i = 1,\ldots,\tau$, where $\tau = k/2$. By Lemma 8.5.2 (iii) Y_i follows the exponential distribution with parameter $\lambda = 1/2$. Let $L = \sum_{i=1}^{\tau} Y_i$. By Lemma 8.5.2 (iv), the variable L follows the Gamma distribution $(k/2, 1/2)$, its expectation $\mathbf{E}[L] = \sum_{i=1}^{k/2} \mathbf{E}[Y_i] = \tau \times 2 = k$.

Now, let $\eta = \lambda\beta k$, we have

$$\mathbf{Pr}[L \geq \beta k] = 1 - \mathbf{Pr}[L \leq \beta k] = 1 - G_{\lambda,\tau}(\beta k) = \sum_{i=0}^{\tau} e^{-\eta}\frac{\eta^i}{i!} \leq (\tau+1)e^{-\eta}\frac{\eta^\tau}{\tau!},$$

since $\eta = \beta\tau > \tau$, as $\beta > 1$. Now, since $\tau! \geq (\tau/e)^\tau$, as can be easily verified[②], and thus

$$\begin{aligned}\mathbf{Pr}[L \geq \beta k] &\leq (\tau+1)e^{-\eta}\frac{\eta^\tau}{\tau^\tau/e^\tau} = (\tau+1)e^{-\eta}\left(\frac{e\eta}{\tau}\right)^\tau = (\tau+1)e^{-\beta\tau}\left(\frac{e\beta\tau}{\tau}\right)^\tau \\ &= (\tau+1)e^{-\beta\tau}\cdot\exp(\tau\ln(e\beta)) = (\tau+1)\exp(-\tau(\beta-(1+\ln\beta))) \\ &\leq \frac{k+3}{2}\exp\left(-\frac{k}{2}(\beta-(1+\ln\beta))\right).\end{aligned}$$

Arguing in a similar fashion, we have, for a large constant $\rho \gg 1$

$$\begin{aligned}\mathbf{Pr}[L \leq k/\beta] &= \sum_{i=\tau}^{\infty} e^{-\tau/\beta}\frac{(\tau/\beta)^i}{i!} \leq e^{-\tau/\beta}\sum_{i=\tau}^{\infty}\left(\frac{e\tau}{i\beta}\right)^i \\ &= e^{-\tau/\beta}\left[\sum_{i=\tau}^{\rho e\tau/\beta}\left(\frac{e\tau}{i\beta}\right)^i + \sum_{i=\rho e\tau/\beta+1}^{\infty}\left(\frac{e\tau}{i\beta}\right)^i\right]\end{aligned}$$

The second sum is very small for $\rho \gg 1$ and we bound only the first one. As the sequence $(\frac{e\tau}{i\beta})^i$ is decreasing for $i \geq \tau/\beta$, we can bound the first sum by

$$\left(\frac{\rho e\tau}{\beta}\right)e^{-\tau/\beta}\left(\frac{e}{\beta}\right)^\tau = O(\tau)\exp\left(-\tau(\beta^{-1}-(1-\ln\beta))\right).$$

Since $\tau = k/2$, we obtain the desired result. ■

Next, we show how to interpret the above inequalities in a somewhat more intuitive way. Let $\beta = 1+\varepsilon$, $\varepsilon > 0$. From Taylor expansion we know that $\ln\beta \leq \varepsilon - \varepsilon^2/2 + \varepsilon^3/3$. By plugging it into the upper bound for $\mathbf{Pr}[L \geq \beta k]$ we get

$$\begin{aligned}\mathbf{Pr}[L \geq \beta k] &\leq O(k)\times\exp(-\frac{k}{2}(\beta-1-\varepsilon+\varepsilon^2/2-\varepsilon^3/3)) \\ &\leq O(k)\times\exp(-\frac{k}{2}(\varepsilon^2/2-\varepsilon^3/3))\end{aligned}$$

On the other hand, we also know that $\ln\beta \geq \varepsilon - \varepsilon^2/2$. Therefore

$$\begin{aligned}\mathbf{Pr}[L \leq k/\beta] &\leq O(k)\times\exp(-\frac{k}{2}(\beta^{-1}-1+\varepsilon-\varepsilon^2/2)) \\ &\leq O(k)\times\exp(-\frac{k}{2}(\frac{1}{1+\varepsilon}-1+\varepsilon-\varepsilon^2/2)) \\ &\leq O(k)\times\exp(-\frac{k}{2}(\frac{\varepsilon^2}{1+\varepsilon}-\varepsilon^2/2)) \\ &\leq O(k)\times\exp(-\frac{k}{2}\cdot\frac{\varepsilon^2-\varepsilon^3}{2(1+\varepsilon)})\end{aligned}$$

[②]Indeed, $\ln\tau! = \sum_{i=1}^{\tau}\ln i \geq \int_{x=1}^{n}\ln x\,dx = \Big[x\ln x - x\Big]_{x=1}^{n} = n\ln n - n + 1 \geq n\ln n - n = \ln((n/e)^n)$.

Thus, the probability that a given unit vector gets distorted by more than $(1+\varepsilon)$ in any direction[3] grows roughly as $\exp(-k\varepsilon^2/4)$, for small $\varepsilon > 0$. Therefore, if we are given a set P of n points in l_2, we can set k to roughly $8\ln(n)/\varepsilon^2$ and make sure that with non-zero probability we obtain projection which does not distort distances[4] between *any* two different points from P by more than $(1+\varepsilon)$ in each direction. This is (up to the constants) the statement of *Johnson-Lindenstrauss* lemma [JL84, FM88]. We mention that our constant 8 yields a slight improvement over the earlier constant 9 obtained in [FM88].

8.6 Bibliographical notes

Our presentation follows Matoušek [Mat02]. The Brunn-Minkowski inequality is a powerful inequality which is widely used in mathematics. A nice survey of this inequality and its applications is provided by Gardner [Gar02]. Gardner says: "In a sea of mathematics, the Brunn-Minkowski inequality appears like an octopus, tentacles reaching far and wide, its shape and color changing as it roams from one area to the next." However, Gardner is careful in claiming that the Brunn-Minkowski inequality is one of the most powerful inequalities in mathematics since as a wit put it "the most powerful inequality is $x^2 \geq 0$, since all inequalities are in some sense equivalent to it."

A striking application of the Brunn-Minkowski inequality is the proof that in any partial ordering of n elements, there is a single comparison that knowing its result, reduces the number of linear extensions that are consistent with the partial ordering, by a constant fraction. This immediately implies (the uninteresting result) that one can sort n elements in $O(n \log n)$ comparisons. More interestingly, it implies that if there are m linear extensions of the current partial ordering, we can *always* sort it using $O(\log m)$ comparisons. A nice exposition of this surprising result is provided by Matoušek [Mat02, Section 12.3].

The probability review of Section 8.5.1 can be found in Feller [Fel71]. The alternative proof of the Johnson-Lindenstrauss lemma of Section 8.5.2 is due to Indyk and Motwani [IM98]. It exposes the fact that the Johnson-Lindenstrauss lemma is no more than yet another instance of the concentration of mass phenomena (i.e., like the Chernoff inequality). The alternative proof is provided since it is conceptually simpler (although the computations are more involved), and it is technically easier to use. Another alternative proof is provided by Dasgupta and Gupta [DG99].

Interestingly, it is enough to pick each entry in the dimension reducing matrix randomly out of $-1, 0, 1$. This requires more involved proof [Ach01]. This is useful when one care about storing this dimension reduction transformation efficiently.

Magen [Mag01] observed that in fact the JL lemma preserves angles, and in fact can be used to preserve any "k dimensional angle", by projecting down to dimension $O(k\varepsilon^{-2}\log n)$. In particular, Exercise 8.7.1 is taken from there.

Dimension reduction is crucial in learning, AI, databases, etc. One common technique that is being used in practice is to do PCA (i.e., principal component analysis) and take the first few main axises. Other techniques include independent component analysis, and MDS (multidimensional scaling). MDS tries to embed points from high dimensions into low dimension ($d = 2$ or 3), which preserving some properties. Theoretically, dimension reduction into really low dimensions is hopeless, as the distortion in the worst case is $\Omega(n^{1/(k-1)})$, if k is the target dimension [Mat90].

8.7 Exercises

Exercise 8.7.1 [10 Points] Show that the Johnson-Lindenstrauss lemma also $(1\pm\varepsilon)$-preserves angles among triples of points of P (you might need to increase the target dimension however by a constant factor). **[Hint:**

[3] Note that this implies distortion $(1+\varepsilon)^2$ if we require the mapping to be a contraction.

[4] In fact, this statement holds even for the *square* of the distances.

For every angle, construct a equilateral triangle that its edges are being preserved by the projection (add the vertices of those triangles [conceptually] to the point set being embedded). Argue, that this implies that the angle is being preserved.**]**

Chapter 9

ANN in High Dimensions

9.1 ANN on the Hypercube

9.1.1 Hypercube and Hamming distance

Definition 9.1.1 The set of points $\mathcal{H}^d = \{0,1\}^d$ is the d-dimensional hypercube. A point $p = (p_1, \ldots, p_d) \in \mathcal{H}^d$ can be interpreted, naturally, as a binary string $p_1 p_2 \ldots p_d$. The *Hamming distance* $\mathbf{d}_H(p,q)$ between $p, q \in \mathcal{H}^d$, is the number of coordinates where p and q disagree.

It is easy to verify that the Hamming distance comply with the triangle inequality, and is as such a metric.

As we saw in previous lectures, all we need to solve $(1+\varepsilon)$-ANN, is it is enough to efficiently solve the *approximate near neighbor* problem. Namely, given a set P of n points in $\mathcal{H}^d$, and radius $r > 0$ and parameter $\varepsilon > 0$, we want to decide for a query point q whether $\mathbf{d}_H(q,P) \le r$ or $\mathbf{d}_H(q,P) \ge (1+\varepsilon)r$.

Definition 9.1.2 For a set P of points, a data-structure $\mathrm{NNbr}_{\approx}(P, r, (1+\varepsilon)r)$ solves the *approximate near neighbor* problem, if given a query point q, the data-structure works as follows.

- If $\mathbf{d}(q,P) \le r$ then $\mathrm{NNbr}_{\approx}$ outputs a point $p \in P$ such that $\mathbf{d}(p,q) \le (1+\varepsilon)r$.
- If $\mathbf{d}(q,P) \ge (1+\varepsilon)r$, in this case $\mathrm{NNbr}_{\approx}$ outputs that "$\mathbf{d}(q,P) \ge r$".
- If $r \le \mathbf{d}(q,P) \le (1+\varepsilon)r$, either of the above answers is acceptable.

Given such a data-structure $\mathrm{NNbr}_{\approx}(P, r, (1+\varepsilon)r)$, one can construct a data-structure that answers ANN using $O(\log(n/\varepsilon))$ queries.

9.1.2 Constructing NNbr for the Hamming cube

Let $P = \{p_1, \ldots, p_n\}$ be a subset of vertices of the hypercube in d dimensions. Let $r, \varepsilon > 0$ be two prespecified parameters. We are interested in building an $\mathrm{NNbr}_{\approx}$ for balls of radius r in the Hamming distance.

Definition 9.1.3 Let U be a (small) positive integer. A family $\mathcal{F} = \{h : S \to [0, U]\}$ of functions, is an (r, R, α, β)-sensitive if for any $u, q \in S$, we have:

- If $u \in \mathbf{b}(q,r)$ then $\mathbf{Pr}[h(u) = h(v)] \ge \alpha$.
- If $u \notin \mathbf{b}(q,R)$ then $\mathbf{Pr}[h(u) = h(v)] \le \beta$,

where h is randomly picked from $\mathcal{F}$, $r < R$, and $\alpha > \beta$.

Intuitively, if we can construct a (r, R, α, β)-sensitive family, then we can distinguish between two points which are close together, and two points which are far away from each other. Of course, the probabilities α and β might be very close to each other, and we need a way to do amplification.

Lemma 9.1.4 *For the hypercube $\mathcal{H}^d = \{0,1\}^d$, and a point $b = (b_1, \ldots, b_d) \in \mathcal{H}^d$, let $\mathcal{F}$ be the set of functions*

$$\left\{ h_i(b) = b_i \;\middle|\; b = (b_1, \ldots, b_d) \in \mathcal{H}^d, \text{ for } i = 1, \ldots, d \right\}.$$

Then for any r, ε, the family $\mathcal{F}$ is $\left(r, (1+\varepsilon)r, 1 - \frac{r}{d}, 1 - \frac{r(1+\varepsilon)}{d}\right)$-sensitive.

Proof: If $u, v \in \{0,1\}^d$ are in distance smaller than r from each other (under the Hamming distance), then they differ in at most r coordinates. The probability that $h \in \mathcal{F}$ would project into a coordinate that u and v agree on is $\geq 1 - r/d$.

Similarly, if $\mathbf{d}_H(u, v) \geq (1+\varepsilon)r$ then the probability that h would map into a coordinate that u and v agree on is $\leq 1 - (1+\varepsilon)r/d$. ■

Let k be a parameter to be specified shortly. Let

$$\mathcal{G}(\mathcal{F}) = \left\{ g : \{0,1\}^d \to \{0,1\}^k \;\middle|\; g(u) = (h^1(u), \ldots, h^k(u)), \text{ for } h^1, \ldots, h^k \in \mathcal{F} \right\}.$$

Intuitively, $\mathcal{G}$ is a family that extends $\mathcal{F}$ by probing into k coordinates instead of only one coordinate.

9.1.3 Construction the near-neighbor data-structure

Let τ be (yet another) parameter to be specified shortly. We pick $g_1, \ldots, g_\tau$ functions randomly and uniformly from $\mathcal{G}$. For each point $u \in P$ compute $g_1(u), \ldots, g_\tau(u)$. We construct a hash table H_i to store all the values of $g_i(p_1), \ldots, g_i(p_n)$, for $i = 1, \ldots, \tau$.

Given a query point $q \in \mathcal{H}^d$, we compute $p_1(q), \ldots, p_\tau(q)$, and retrieve all the points stored in those buckets in the hash tables $H_1, \ldots, H_\tau$, respectively. For every point retrieved, we compute its distance to q, and if this distance is $\leq (1+\varepsilon)r$, we return it. If we encounter more than 4τ points we abort, and return '*fail*'. If no "close" point is encountered, the search returns '*fail*'.

We choose k and τ so that with constant probability (say larger than half) we have the following two properties:

(1) If there is a point $u \in P$, such that $\mathbf{d}_H(u, q) \leq r$, then $g_j(u) = g_j(q)$ for some j.

(2) Otherwise (i.e., $\mathbf{d}_H(u, q) \geq (1+\varepsilon)r$), the total number of points colliding with q in the τ hash tables, is smaller than 4τ.

Given a query point, $q \in \mathcal{H}^d$, we need to perform τ probes into τ hash tables, and retrieve at most 4τ results. Overall this takes $O(d^{O(1)}\tau)$ time.

Lemma 9.1.5 *If there is a $(r, (1+\varepsilon)r, \alpha, \beta)$-sensitive family $\mathcal{F}$ of functions for the hypercube, then there exists a* $\mathrm{NNbr}_{\approx}(P, r, (1+\varepsilon)r)$ *which uses $O\left(dn + n^{1+\rho}\right)$ space and $O(n^\rho)$ hash probes for each query, where*

$$\rho = \frac{\ln 1/\alpha}{\ln 1/\beta}.$$

This data-structure succeeds with constant probability.

Proof: It suffices to ensure that properties (1) and (2) holds with probability larger than half.

Set $k = \log_{1/\beta} n = \frac{\ln n}{\ln(1/\beta)}$, then the probability that for a random hash function $g \in \mathcal{G}(\mathcal{F})$, we have $g(p) = g(q)$ for $p \in P \setminus \mathbf{b}(q, (1+\varepsilon)r)$ is at most

$$\mathbf{Pr}[g(p') = g(q)] \le \beta^k \le \exp\Big(\ln(\beta) \cdot \tfrac{\ln n}{\ln(1/\beta)}\Big) \le \frac{1}{n}.$$

Thus, the expected number of elements from $P \setminus \mathbf{b}(q, (1+\varepsilon)r)$ colliding with q in the jth hash table H_j is bounded by one. In particular, the overall expected number of such collisions in $H_1, \ldots, H_\tau$ is bounded by τ. By the Markov inequality we have that the probability that the collusions number exceeds 4τ is less than $1/4$; therefore the probability that the property (2) holds is $\ge 3/4$.

Next, for a point $p \in \mathbf{b}(q, r)$, consider the probability of $g_j(p) = g_j(q)$, for a fixed j. Clearly, it is bounded from below by

$$\ge \alpha^k = \alpha^{\log_{1/\beta} n} = n^{-\frac{\ln 1/\alpha}{\ln 1/\beta}} = n^{-\rho}.$$

Thus the probability that such a g_j exists is at least $1 - (1 - n^{-\rho})^\tau$. By setting $\tau = 2n^\rho$ we get property (1) holds with probability $\ge 1 - 1/e^2 > 4/5$. The claim follows. ■

Claim 9.1.6 *For $x \in [0, 1)$ and $t \ge 1$ such that $1 - tx > 0$ we have* $\dfrac{\ln(1-x)}{\ln(1-tx)} \le \dfrac{1}{t}$.

Proof: Since $\ln(1 - tx) < 0$, it follows that the claim is equivalent to $t \ln(1 - x) \ge \ln(1 - tx)$. This in turn is equivalent to

$$g(x) \equiv (1 - tx) - (1 - x)^t \le 0.$$

This is trivially true for $x = 0$. Furthermore, taking the derivative, we see $g'(x) = -t + t(1 - x)^{t-1}$, which is non-positive for $x \in [0, 1)$ and $t \ge 1$. Therefore, g is non-increasing in the region in which we are interested, and so $g(x) \le 0$ for all values in this interval. ■

Lemma 9.1.7 *There exists a* $\mathrm{NNbr}_\approx(r, (1+\varepsilon)r)$ *which uses* $O\Big(dn + n^{1+1/(1+\varepsilon)}\Big)$ *space and* $O(n^{1/(1+\varepsilon)})$ *hash probes for each query. The probability of success (i.e., there is a point $u \in P$ such that $\mathbf{d}_H(u, q) \le r$, and we return a point $v \in P$ such that $\|uv\| \le (1+\varepsilon)r$) is a constant.*

Proof: By Lemma 9.1.4, we have a $(r, (1+\varepsilon)r, \alpha, \beta)$-sensitive family of hash functions, where $\alpha = 1 - \frac{r}{d}$ and $\beta = 1 - \frac{r(1+\varepsilon)}{d}$. As such

$$\rho = \frac{\ln 1/\alpha}{\ln 1/\beta} = \frac{\ln \alpha}{\ln \beta} = \frac{\ln \frac{d-r}{d}}{\ln \frac{d-(1+\varepsilon)r}{d}} = \frac{\ln\Big(1 - \frac{r}{d}\Big)}{\ln\Big(1 - (1+\varepsilon)\frac{r}{d}\Big)} \le \frac{1}{1+\varepsilon},$$

by Claim 9.1.6. ■

By building $O(\log n)$ structures of Lemma 9.1.7, we can do probability amplification and get a correct result with High probability.

Theorem 9.1.8 *Given a set P of n points on the hypercube $\mathcal{H}^d$, parameters $\varepsilon > 0$ and $r > 0$, one can build a* $\mathrm{NNbr}_\approx = \mathrm{NNbr}_\approx(P, r, (1+\varepsilon)r)$*, such that given a query point q, one can decide if:*

- $\mathbf{b}(q, r) \cap P \ne \emptyset$*, then* $\mathrm{NNbr}_\approx$ *returns a point $u \in P$, such that* $\mathbf{d}_H(u, q) \le (1+\varepsilon)r$.
- $\mathbf{b}(q, (1+\varepsilon)r) \cap P = \emptyset$ *then* $\mathrm{NNbr}_\approx$ *returns that no point is in distance $\le r$ from q.*

In any other case, any of the answers is correct. The query time is $O(dn^{1/(1+\varepsilon)} \log n)$ and the space used is $O\Big(dn + n^{1+1/(1+\varepsilon)} \log n\Big)$. The result returned is correct with high probability.

Proof: Note, that every point can be stored only once. Any other reference to it in the data-structure can be implemented with a pointer. Thus, the $O(dn)$ requirement on the space. The other term follows by repeating the space requirement of Lemma 9.1.7 $O(\log n)$ times. ■

In the hypercube case, we can just build $M = O(\varepsilon^{-1} \log n)$ such data-structures such that $(1+\varepsilon)$-ANN can be answered using binary search on those data-structures, which corresponds to radiuses $r_1, \ldots, r_M$, where $r_i = (1+\varepsilon)^i$.

Theorem 9.1.9 *Given a set P of n points on the hypercube $\mathcal{H}^d$, parameters $\varepsilon > 0$ and $r > 0$, one can build ANN data-structure using $O\Big((d + n^{1/(1+\varepsilon)})\varepsilon^{-1} n \log^2 n\Big)$ space, such that given a query point q, one can returns an ANN in P (under the Hamming distance) in $O(dn^{1/(1+\varepsilon)} \log(\varepsilon^{-1} \log n))$ time. The result returned is correct with high probability.*

9.2 LSH and ANN on Euclidean Space

9.2.1 Preliminaries

Lemma 9.2.1 *Let $X = (X_1, \ldots, X_d)$ be a vector of d independent variables which have distribution $N(0,1)$, and let $v = (v_1, \ldots, v_d) \in \mathbb{R}^d$. We have that $v \cdot X = \sum_i v_i X_i$ is distributed as $\|v\| Z$, where $Z \sim N(0,1)$.*

Proof: If $\|v\| = 1$ then this holds by the symmetry of the normal distribution. Indeed, let $e_1 = (1, 0, \ldots, 0)$. By the symmetry of the d-dimensional normal distribution, we have that $v \cdot X \sim e_1 \cdot X = X_1 \sim N(0,1)$.

Otherwise, $v \cdot X / \|V\| \sim N(0,1)$, and as such $v \cdot X \sim N\Big(0, \|v\|^2\Big)$, which is indeed the distribution of $\|v\| Z$. ■

A d-dimensional distribution that has the property of Lemma 9.2.1, is called a 2-*stable distribution*.

9.2.2 Locality Sensitive Hashing

Let q, r be two points in $\mathbb{R}^d$. We want to perform an experiment to decide if $\|\mathsf{q} - \mathsf{r}\| \leq 1$ or $\|\mathsf{q} - \mathsf{r}\| \geq \eta$, where $\eta = 1 + \varepsilon$. We will randomly choose a vector $\vec{v}$ from the d-dimensional normal distribution $N^d(0,1)$ (which is 2-stable). Next, let r be a parameter, and let t be a random number chosen uniformly from the interval $[0, r]$. For $p \in \mathbb{R}^d$, and consider the random hash function

$$h(p) = \left\lfloor \frac{p \cdot \vec{v} + t}{r} \right\rfloor. \tag{9.1}$$

If p and q are in distance η from each other, and when we project to $\vec{v}$, the distance between the projection is t, then the probability that they get the same hash value is $1 - t/r$, since this is the probability that the random sliding will not separate them. As such, we have that the probability of collusion is

$$\alpha(\eta) = \mathbf{Pr}\big[h(p) = h(q)\big] = \int_{t=0}^{r} \mathbf{Pr}\Big[\big|p \cdot \vec{v} - q \cdot \vec{v}\big| = t\Big]\left(1 - \frac{t}{r}\right) dt.$$

However, since $\vec{v}$ is chosen from a 2-stable distribution, we have that $p \cdot \vec{v} - q \cdot \vec{v} = (p-q) \cdot \vec{v} \sim N(0, \|pq\|^2)$. Since we are considering the absolute value of the variable, we need to multiply this by two. Thus, we have

$$\alpha(\eta, r) = \int_{t=0}^{r} \frac{2}{\sqrt{2\pi}\eta} \exp\left(-\frac{t^2}{2\eta^2}\right)\left(1 - \frac{t}{r}\right) dt.$$

Intuitively, we care about the difference $\alpha(1+\varepsilon, r) - \alpha(1, r)$, and we would like to maximize it as much as possible (by choosing the right value of r). Unfortunately, this integral is unfriendly, and we have to resort to numerical computation.

In fact, if are going to use this hashing scheme for constructing locality sensitive hashing, like in Lemma 9.1.5, then we care about the ratio

$$\rho(1+\varepsilon) = \min_r \frac{\log(1/\alpha(1))}{\log(1/\alpha(1+\varepsilon))}.$$

The following is verified using numerical computations on a computer,

Lemma 9.2.2 ([DNIM04]) *One can choose r, such that* $\rho(1+\varepsilon) \le \frac{1}{1+\varepsilon}$.

Lemma 9.2.2 implies that the hash functions defined by Eq. (9.1) are $(1, 1+\varepsilon, \alpha', \beta')$-sensitive, and furthermore, $\rho = \frac{\log(1/\alpha')}{\log(1/\beta')} \le \frac{1}{1+\varepsilon}$, for some values of α' and β'. As such, we can use this hashing family to construct $\mathrm{NNbr}_{\approx}$ for the set P of points in $\mathbb{R}^d$. Following the same argumentation of Theorem 9.1.8, we have the following.

Theorem 9.2.3 *Given a set P of n points in* $\mathbb{R}^d$*, parameters* $\varepsilon > 0$ *and* $r > 0$*, one can build a* $\mathrm{NNbr}_{\approx} = \mathrm{NNbr}_{\approx}(P, r, (1+\varepsilon)r)$*, such that given a query point q, one can decide if:*

- $\mathbf{b}(q, r) \cap P \ne \emptyset$*, then* $\mathrm{NNbr}_{\approx}$ *returns a point* $u \in P$*, such that* $\mathbf{d}_H(u, q) \le (1+\varepsilon)r$.
- $\mathbf{b}(q, (1+\varepsilon)r) \cap P = \emptyset$ *then* $\mathrm{NNbr}_{\approx}$ *returns that no point is in distance* $\le r$ *from q.*

In any other case, any of the answers is correct. The query time is $O(dn^{1/(1+\varepsilon)} \log n)$ *and the space used is* $O\left(dn + n^{1/(1+\varepsilon)} n \log n\right)$*. The result returned is correct with high probability.*

9.2.3 ANN in High Dimensional Euclidean Space

Unlike the hypercube case, where we could just do direct binary search on the distances. Here we need to use the reduction from ANN to near-neighbor queries. We will need the following result (which follows from what we had seen in previous lectures).

Theorem 9.2.4 *Given a set P of n points in* $\mathbb{R}^d$*, then one can construct data-structures* $\mathcal{D}$ *that answers* $(1+\varepsilon)$*-ANN queries, by performing* $O(\log(n/\varepsilon))$ $\mathrm{NNbr}_{\approx}$ *queries. The total number of points stored at* $\mathrm{NNbr}_{\approx}$ *data-structures of* $\mathcal{D}$ *is* $O(n\varepsilon^{-1} \log(n/\varepsilon))$.

Constructing the data-structure of Theorem 9.2.4 requires building a low quality HST. Unfortunately, the previous construction seen for HST are exponential in the dimension, or take quadratic time. We next present a faster scheme.

9.2.3.1 Low quality HST in high dimensional Euclidean space

Lemma 9.2.5 *Let P be a set of n in* $\mathbb{R}^d$*. One can compute a nd-HST of P in* $O(nd \log^2 n)$ *time (note, that the constant hidden by the O notation does not depend on d).*

Proof: Our construction is based on a recursive decomposition of the point-set. In each stage, we split the point-set into two subsets. We recursively compute a *nd*-HST for each point-set, and we merge the two trees into a single tree, by creating a new vertex, assigning it an appropriate value, and hung the two subtrees from this node. To carry this out, we try to separate the set into two subsets that are furthest away from each other.

Let $R = R(P)$ be the minimum axis parallel box containing P, and let $\nu = l(P) = \sum_{i=1}^{d} \|I_i(R)\|$, where $I_i(R)$ is the projection of R to the ith dimension.

Clearly, one can find an axis parallel strip H of width $\geq \nu/((n-1)d)$, such that there is at least one point of P on each of its sides, and there is no points of P inside H. Indeed, to find this strip, project the point-set into the ith dimension, and find the longest interval between two consecutive points. Repeat this process for $i = 1, \ldots, d$, and use the longest interval encountered. Clearly, the strip H corresponding to this interval is of width $\geq \nu/((n-1)d)$. On the other hand, $\text{diam}(P) \leq \nu$.

Now recursively continue the construction of two trees T^+, T^-, for P^+, P^-, respectively, where P^+, P^- is the splitting of P into two sets by H. We hung T^+ and T^- on the root node v, and set $\Delta_v = \nu$. We claim that the resulting tree T is a nd-HST. To this end, observe that $\text{diam}(P) \leq \Delta_v$, and for a point $p \in P^-$ and a point $q \in P^+$, we have $\|pq\| \geq \nu/((n-1)d)$, which implies the claim.

To construct this efficiently, we use an efficient search trees to store the points according to their order in each coordinate. Let $D_1, \ldots, D_d$ be those trees, where D_i store the points of P in ascending order according to the ith axis, for $i = 1, \ldots, d$. We modify them, such that for every node $v \in D_i$, we know what is the largest empty interval along the ith axis for the points P_v (i.e., the points stored in the subtree of v in D_i). Thus, finding the largest strip to split along, can be done in $O(d \log n)$ time. Now, we need to split the d trees into two families of d trees. Assume we split according to the first axis. We can split D_1 in $O(\log n)$ time using the splitting operation provided by the search tree (Treaps for example can do this split in $O(\log n)$ time). Let assume that this split P into two sets L and R, where $|L| < |R|$.

We still need to split the other $d-1$ search trees. This is going to be done by deleting all the points of L from those trees, and building $d-1$ new search trees for L. This takes $O(|L|\, d \log n)$ time. We charge this work to the points of L.

Since in every split, only the points in the smaller portion of the split get charged, it follows that every point can be charged at most $O(\log n)$ time during this construction algorithm. Thus, the overall construction time is $O(dn \log^2 n)$ time. ■

9.2.3.2 The overall result

Plugging Theorem 9.2.3 into Theorem 9.2.4, we have:

Theorem 9.2.6 *Given a set P of n points in $\mathbb{R}^d$, parameters $\varepsilon > 0$ and $r > 0$, one can build ANN data-structure using*

$$O\Big(dn + n^{1+1/(1+\varepsilon)} \varepsilon^{-2} \log^3(n/\varepsilon)\Big)$$

space, such that given a query point q, one can returns an $(1+\varepsilon)$-ANN in P in

$$O\left(dn^{1/(1+\varepsilon)} (\log n) \log \frac{n}{\varepsilon}\right)$$

time. The result returned is correct with high probability.

The construction time is $O\Big(dn^{1+1/(1+\varepsilon)} \varepsilon^{-2} \log^3(n/\varepsilon)\Big)$.

Proof: We compute the low quality HST using Lemma 9.2.5. This takes $O(nd \log^2 n)$ time. Using this HST, we can construct the data-structure $\mathcal{D}$ of Theorem 9.2.4, where we do not compute the $\text{NNbr}_{\approx}$ data-structures. We next traverse the tree $\mathcal{D}$, and construct the $\text{NNbr}_{\approx}$ data-structures using Theorem 9.2.3.

We only need to prove the bound on the space. Observe, that we need to store each point only once, since other place can refer to the point by a pointer. Thus, this is the $O(nd)$ space requirement. The other term comes from plugging the bound of Theorem 9.2.4 into the bound of Theorem 9.2.3. ■

9.3 Bibliographical notes

Section 9.1 follows the exposition of Indyk and Motwani [IM98]. The fact that one can perform approximate nearest neighbor in high dimensions in time and space polynomial in the dimension is quite surprising, One can reduce the approximate near-neighbor in euclidean space to the same question on the hypercube (we show the details below). This implies together with the reduction from ANN to approximate near-neighbor (seen in previous lectures) that one can answer ANN in high dimensional euclidean space with similar performance. Kushilevitz, Ostrovsky and Rabani [KOR00] offered an alternative data-structure with somewhat inferior performance.

The value of the results showed in this write-up depend to large extent on the reader perspective. Indeed, for small value of $\varepsilon > 0$, the query time $O(dn^{1/(1+\varepsilon)})$ is very close to linear dependency on n, and is almost equivalent to just scanning the points. Thus, from low dimension perspective, where ε is assumed to be small, this result is slightly sublinear. On the other hand, if one is willing to pick ε to be large (say 10), then the result is clearly better than the naive algorithm, suggesting running time for an ANN query which takes (roughly) $n^{1/11}$.

The idea of doing locality sensitive hashing directly on the Euclidean space, as done in Section 9.2 is not shocking after seeing the Johnson-Lindenstrauss lemma. It is taken from a recent paper of Datar *et al.* [DNIM04]. In particular, the current analysis which relies on computerized estimates is far from being satisfactory. It would be nice to have a simpler and more elegant scheme for this case. This is an open problem for further research. Another open problem is to improve the performance of the LSH scheme.

The low-quality high-dimensional HST construction of Lemma 9.2.5, is taken from [Har01]. The running time of this lemma can be further improved to $O(dn \log n)$ by more careful and involved implementation, see [CK95] for details.

From approximate near-neighbor in $\mathbb{R}^d$ to approximate near-neighbor on the hypercube. The reduction is quite involved, and we only sketch the details. Let P Be a set of n points in $\mathbb{R}^d$. We first reduce the dimension to $k = O(\varepsilon^{-2} \log n)$ using the Johnson-Lindenstrauss lemma. Next, we embed this space into $\ell_1^{k'}$ (this is the space $\mathbb{R}^k$, where distances are the L_1 metric instead of the regular L_2 metric), where $k' = O(k/\varepsilon^2)$. This can be done with distortion $(1 + \varepsilon)$.

Let Q the resulting set of points in $\mathbb{R}^{k'}$. We want to solve $\mathrm{NNbr}_{\approx}$ on this set of points, for radius r. As a first step, we partition the space into cells by taking a grid with sidelength (say) $k'r$, and randomly translating it, clipping the points inside each grid cell. It is now sufficient to solve the $\mathrm{NNbr}_{\approx}$ inside this grid cell (which has bounded diameter as a function of r), since with small probability that the result would be correct. We amplify the probability by repeating this polylogarithmic number of times.

Thus, we can assume that P is contained inside a cube of side length $\leq k'nr$, and it is in $\mathbb{R}^{k'}$, and the distance metric is the L_1 metric. We next, snap the points of P to a grid of sidelength (say) $\varepsilon r/k'$. Thus, every point of P now has an integer coordinate, which is bounded by a polynomial in $\log n$ and $1/\varepsilon$. Next, we write the coordinates of the points of P using unary notation. (Thus, a point $(2, 5)$ would be written as $(010, 101)$ assuming the number of bits for each coordinates is 3.) It is now easy to verify that the hamming distance on the resulting strings, is equivalent to the L_1 distance between the points.

Thus, we can solve the near-neighbor problem for points in $\mathbb{R}^d$ by solving it on the hypercube under the Hamming distance.

See Indyk and Motwani [IM98] for more details.

This relationship indicates that the ANN on the hypercube is "equivalent" to the ANN in Euclidean space. In particular, making progress on the ANN on the hypercube would probably lead to similar progress on the Euclidean ANN problem.

We had only scratched the surface of proximity problems in high dimensions. The interested reader is referred to the survey by Indyk [Ind04] for more information.

Chapter 10

Approximating a Convex Body by An Ellipsoid

10.1 Some Linear Algebra

In the following, we cover some material from linear algebra. Proofs of those facts can be found in any text on linear algebra, for example [Leo98].

For a matrix A, let A^T denote the transposed matrix. We remind the reader that for two matrices A and B, we have $(AB)^T = B^T A^T$. Furthermore, for any three matrices A, B and C, we have $(AB)C = A(BC)$.

A matrix $A \in \mathbb{R}^{n\times n}$ is *symmetric* if $A^T = A$. All the eigenvalues of a symmetric matrix are real numbers. A matrix A is *positive definite* if $x^T A x > 0$, for all $x \in \mathbb{R}^n$. Among other things this implies that A is non-singular. If A is symmetric then it is positive definite if and only if all its eigenvalues are positive numbers.

In particular, if A is symmetric positive definite then $\det(A) > 0$.

For two vectors $u, v \in \mathbb{R}^n$, let $\langle u, v\rangle = u^T v$ denote their dot product.

10.2 Ellipsoids

Definition 10.2.1 Let $\mathbf{b} = \left\{ x \,\middle|\, \|x\| \le 1 \right\}$ be the unit ball. Let $a \in \mathbb{R}^n$ be a vector and let $T : \mathbb{R}^n \to \mathbb{R}^n$ be an invertible linear transformation. The set

$$\mathcal{E} = T(\mathbf{b}) + a$$

is called an *ellipsoid* and a is its *center*.

Alternatively, we can write

$$\mathcal{E} = \left\{ x \in \mathbb{R}^n \,\middle|\, \left\| T^{-1}(x-a) \right\| \le 1 \right\}.$$

However,

$$\left\| T^{-1}(x-a) \right\| = \left\langle T^{-1}(x-a), T^{-1}(x-a) \right\rangle = (T^{-1}(x-a))^T T^{-1}(x-a) = (x-a)^T \left(T^{-1}\right)^T T^{-1}(x-a).$$

In particular, let $Q = \left(T^{-1}\right)^T T^{-1}$. Observe that Q is symmetric, and that it is positive definite. Thus,

$$\mathcal{E} = \left\{ x \in \mathbb{R}^n \,\middle|\, (x-a)^T Q (x-a) \le 1 \right\}.$$

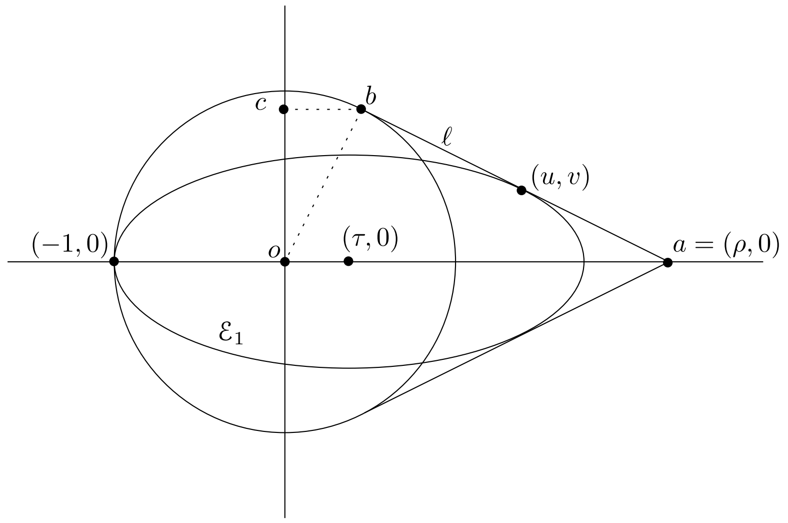


Figure 10.1: Ellipse

If we change the basis of $\mathbb{R}^n$ to be the set of unit eigenvectors of $\mathcal{E}$, then Q becomes a diagonal matrix, and we have that

$$\mathcal{E} = \left\{ (y_1, \ldots, y_n) \in \mathbb{R}^n \;\middle|\; \lambda_1 (y_1 - a_1)^2 + \cdots + \lambda_n (y_n - a_n)^2 \leq 1 \right\},$$

where $a = (a_1, \ldots, a_n)$ and $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of Q. In particular, this implies that the point $(a_1, \ldots, a_i + 1/\sqrt{\lambda_i}, \ldots, a_n) \in \partial\mathcal{E}$, for $i = 1, \ldots, n$. In particular,

$$\mathrm{Vol}(\mathcal{E}) = \frac{\mathrm{Vol}(\mathbf{b})}{\sqrt{\lambda_1} \cdots \sqrt{\lambda_n}} = \frac{\mathrm{Vol}(\mathbf{b})}{\sqrt{\det(Q)}}.$$

For a convex body P (i.e., a convex and bounded set), let $\mathcal{E}$ be a largest volume ellipsoid contained inside P. One can show that $\mathcal{E}$ is unique. Namely, there is a single maximum volume ellipsoid inside P.

Theorem 10.2.2 *Let $K \subset \mathbb{R}^n$ be a convex body, and let $\mathcal{E} \subseteq K$ be its maximum volume ellipsoid. Suppose that $\mathcal{E}$ is centered at the origin, then $K \subseteq n\,\mathcal{E} = \left\{ nx \;\middle|\; x \in \mathcal{E} \right\}$.*

Proof: Applying a linear transformation, we can assume that $\mathcal{E}$ is the unit ball $\mathbf{b}$. And assume for the sake of contradiction that there is a point $x \in K$, such that$\|x\| > n$. Consider the set C which is the convex-hull of $\{x\} \cup \mathbf{b}$. Since K is convex, we have that $C \subseteq K$.

We will reach a contradiction, by finding an ellipsoid $\mathcal{E}_1$ which has volume larger than $\mathbf{b}$, which is enclosed inside C.

By rotation of space, we can assume that $x = (\rho, 0, \ldots, 0)$, for $\rho > n$. We consider ellipsoids of the form

$$\mathcal{E}_1 = \left\{ (y_1, \ldots, y_n) \;\middle|\; \frac{(y_1 - \tau)^2}{\alpha^2} + \frac{1}{\beta^2} \sum_{i=2}^{n} y_i^2 \leq 1 \right\}.$$

We need to pick the values of τ, α and β such that $\mathcal{E}_1 \subseteq C$. Observe that by symmetry, it is enough to enforce that $\mathcal{E}_1 \subseteq C$ in the first two dimensions. Thus, we can consider C and $\mathcal{E}$ to be in two dimensions.

Thus, the center of $\mathcal{E}_1$ is going to be on the x-axis, at the point $(\tau, 0)$. The set $\mathcal{E}$ is just an ellipse with axises parallel to x and y. See Figure 10.1. In particular, we require that $(-1, 0)$ would be on the boundary of $\mathcal{E}_1$. This implies that $(-1-\tau)^2 = \alpha^2$ and that $0 \le \tau \le (\rho + (-1))/2$. Namely, $\alpha = 1 + \tau$. In particular, the equation of $\mathcal{E}_1$ is

$$F(x,y) = \frac{(x-\tau)^2}{(1+\tau)^2} + \frac{y^2}{\beta^2} - 1 = 0.$$

We next compute the value of β^2.

The tangent line ℓ from $a = (\rho, 0)$ touching the unit circle, touches the unit circle at a point b. See Figure 10.1. We have $\triangle aob \triangleq \triangle obc$. As such $\|cb\| / \|bo\| = \|bo\| / \|oa\|$. Since $\|bo\| = 1$, we have $\|cb\| = 1/\rho$. Since b is on the unit circle, we have $b = (1/\rho, \sqrt{1 - 1/\rho^2})$. As such, the equation of this line is

$$\langle b, (x,y) \rangle = 1 \quad \Rightarrow \quad \frac{x}{\rho} + \sqrt{1 - 1/\rho^2}\, y = 1 \quad \Rightarrow \quad \ell \equiv y = -\frac{1}{\sqrt{\rho^2 - 1}} x - \frac{\rho}{\sqrt{\rho^2 - 1}}.$$

The slope of the tangent ℓ' to $\mathcal{E}_1$ at (u, v) is

$$\frac{dy}{dx} = -\frac{F_x(u,v)}{F_y(u,v)} = -\left(\frac{2(u-\tau)}{(1+\tau)^2}\right) \Big/ \left(\frac{2v}{\beta^2}\right) = \frac{\beta^2 (u-\tau)}{v(1+\tau)^2}$$

by derivatives of implicit functions. The line ℓ' is just

$$\frac{(u-\tau)}{\alpha^2}(x-\tau) + \frac{v}{\beta^2} y = 1,$$

since it has the required slope and it passes through (u, v). Namely $\ell' \equiv y = -\dfrac{\beta^2(u-\tau)}{v\alpha^2}(x-\tau) + \dfrac{\beta^2}{v}$. The line ℓ' passes through $(\rho, 0)$. As such,

$$\frac{(\rho-\tau)(u-\tau)}{\alpha^2} = 1 \;\Rightarrow\; \frac{(u-\tau)}{\alpha} = \frac{\alpha}{\rho - \tau} \;\Rightarrow\; \frac{(u-\tau)^2}{\alpha^2} = \frac{\alpha^2}{(\rho-\tau)^2}. \tag{10.1}$$

Since ℓ and ℓ' are the same line, we have that

$$\frac{\beta^2(u-\tau)}{v\alpha^2} = \frac{1}{\sqrt{\rho^2 - 1}}.$$

However, $\dfrac{\beta^2(u-\tau)}{v\alpha^2} = \dfrac{\beta^2}{v\alpha} \cdot \dfrac{(u-\tau)}{\alpha} = \dfrac{\beta^2}{v\alpha} \cdot \dfrac{\alpha}{\rho - \tau} = \dfrac{\beta^2}{v(\rho-\tau)}$. Thus,

$$\frac{\beta^2}{v} = \frac{\rho - \tau}{\sqrt{\rho^2 - 1}}.$$

Squaring and inverting both sides, we have $\dfrac{v^2}{\beta^4} = \dfrac{\rho^2 - 1}{(\rho - \tau)^2}$ and thus $\dfrac{v^2}{\beta^2} = \beta^2 \dfrac{\rho^2 - 1}{(\rho-\tau)^2}$. The point $(u, v) \in \partial\mathcal{E}_1$, and as such $\dfrac{(u-\tau)^2}{\alpha^2} + \dfrac{v^2}{\beta^2} = 1$. Using Eq. (10.1) and the above, we get

$$\frac{\alpha^2}{(\rho-\tau)^2} + \beta^2 \frac{\rho^2 - 1}{(\rho - \tau)^2} = 1,$$

and thus $\beta^2 = \dfrac{(\rho-\tau)^2 - \alpha^2}{\rho^2 - 1} = \dfrac{(\rho-\tau)^2 - (\tau+1)^2}{\rho^2 - 1} = \dfrac{(\rho - 2\tau - 1)(\rho + 1)}{\rho^2 - 1} = \dfrac{\rho - 2\tau - 1}{\rho - 1} = 1 - \dfrac{2\tau}{\rho - 1}$.

Namely, for $n \geq 2$, and $0 \leq \tau < (\rho - 1)/2$, we have that the ellipsoid $\mathcal{E}_1$ defined by the parameters $\alpha = 1 + \tau$ and $\beta^2 = \frac{(\rho-\tau)^2-(\tau+1)^2}{\rho^2-1}$ is contained inside C. We have $\mathrm{Vol}(\mathcal{E}_1) = \beta^{n-1}\alpha\,\mathrm{Vol}(\mathbf{b})$. As such,

$$\mu = \ln\frac{\mathrm{Vol}(\mathcal{E}_1)}{\mathrm{Vol}(\mathbf{b})} = (n-1)\ln\beta + \ln\alpha = \frac{n-1}{2}\ln\beta^2 + \ln\alpha.$$

For $\tau > 0$ sufficiently small, we have $\ln\alpha = \ln(1+\tau) = \tau + O(\tau^2)$, because of the Taylor expansion $\ln(1+x) = x - x^2/2 + x^3/3 - \cdots$, for $-1 < x \leq 1$. Similarly, $\ln\beta^2 = \ln\left(1 - \frac{2\tau}{\rho-1}\right) = -\frac{2\tau}{\rho-1} + O(\tau^2)$. Thus,

$$\mu = \frac{n-1}{2}\left(-\frac{2\tau}{\rho-1} + O(\tau^2)\right) + \tau + O(\tau^2) > 0,$$

for τ sufficiently small and if $\rho > n$. Thus, $\mathrm{Vol}(\mathcal{E}_1)/\,\mathrm{Vol}(\mathbf{b}) = \exp(\mu) > 1$ implying that $\mathrm{Vol}(\mathcal{E}_1) > \mathrm{Vol}(\mathbf{b})$. A contradiction. ■

A convex body K centered at the origin is *symmetric* if $p \in K$, implies that $-p \in K$. Interestingly, the constant in Theorem 10.2.2 can be improved to $\sqrt{n}$ in this case. We omit the proof, since it is similar to the proof of Theorem 10.2.2.

Theorem 10.2.3 *Let $K \subset \mathbb{R}^n$ be a symmetric convex body, and let $\mathcal{E} \subseteq K$ be its maximum volume ellipsoid. Suppose that $\mathcal{E}$ is centered at the origin, then $K \subseteq \sqrt{n}\,\mathcal{E}$.*

10.3 Bibliographical notes

We closely follow the exposition of Barvinok [Bar02]. One can approximate the John ellipsoid using the interior point techniques [GLS88]. However, this is not very efficient in low dimensions. In the next lecture, we will show how to do this by other (simpler) techniques, which are faster (and simpler) for point sets in low dimensions.

The maximum volume ellipsoid is sometimes referred to as John's ellipsoid. In particular, Theorem 10.2.2 is known as John's theorem, and was originally proved by Fritz John [Joh48].

There are numerous interesting results in convexity theory about how convex shapes looks like. One of the surprising results is Dvoretsky's theorem, which states that any symmetric convex body K around the origin (in "high enough" dimension) can be cut by a k-dimensional subspace, such that the intersection of K with this k-dimensional space, contains an ellipsoid $\mathcal{E}$ and is contained inside an ellipsoid $(1+\varepsilon)\mathcal{E}$ (k here is an arbitrary parameter). Since one can define a metric in a Banach space by providing a symmetric convex body which defines the unit ball, this implies that any high enough dimensional Banach space contains (up to translation that maps the ellipsoid to a ball) a subspace which is almost Euclidean.

A survey of those results is provided by Ball [Bal97].

Chapter 11

Approximating the Minimum Volume Bounding Box of a Point Set

Isn't it an artificial, sterilized, didactically pruned world, a mere sham world in which you cravenly vegetate, a world without vices, without passions without hunger, without sap and salt, a world without family, without mothers, without children, almost without women? The instinctual life is tamed by meditation. For generations you have left to others dangerous, daring, and responsible things like economics, law, and politics. Cowardly and well-protected, fed by others, and having few burdensome duties, you lead your drones' lives, and so that they won't be too boring you busy yourselves with all these erudite specialties, count syllables and letters, make music, and play the Glass Bead Game, while outside in the filth of the world poor harried people live real lives and do real work.
– The Glass Bead Game, Hermann Hesse

11.1 Some Geometry

Let $\mathcal{H} = [0,1]^{\mathsf{d}}$ denote the unit hypercube in $\mathbb{R}^{\mathsf{d}}$. The *width* of a convex body P in $\mathbb{R}^{\mathsf{d}}$ is the minimum distance between two parallel planes that encloses P. Alternatively, the width is the minimum length of the projection of P into a line in $\mathbb{R}^{\mathsf{d}}$. The *diameter* of P is the maximum distance between two points of P. Alternatively, this is the maximum length projection of P into a line.

Lemma 11.1.1 *The width of $\mathcal{H}$ is 1 and its diameter is $\sqrt{d}$.*

Proof: The upper bound on the width is obvious. As for the lower bound, observe that P encloses a ball of radius $1/2$, and as such its projection in any direction is at least 1.

The diameter is just the distance between the two points $(0,\ldots,0)$ and $(1,\ldots,1)$. ■

Lemma 11.1.2 *Let P be a convex body, and let h be a hyperplane cutting P. Let $\mu = \mathrm{Vol}(h \cap P)$, and let $\vec{v}$ be unit vector orthogonal to h. Let ρ be the length of the projection of P into the direction of $\vec{v}$. Then* $\mathrm{Vol}(P) \geq \mu \cdot v/\mathsf{d}$.

Proof: Let $P^+ = P \cap h^+$ and a be the point of maximum distance of P^+ from h, where h^+ denotes the positive half space induced by h.

Let $C = h \cap P$. By the convexity of P, the pyramid $R = \mathcal{CH}(C \cup \{a\})$ is contained inside P, where $\mathcal{CH}(S)$ denotes the convex-hull of S. We explicitly compute the volume of R for the sake of completeness. To this end, let s be the segment connecting a to its projection on h, and let α denote the length of s. Parameterizing s by the interval $[0,\rho^-]$, let $C(t)$ denote the intersection of the hyperplane passing through $s(t)$ and R, where $s(0) = a$, and $s(t) \in h$. Clearly, $\mathrm{Vol}(C(t)) = (t/\alpha)^{d-1}\,\mathrm{Vol}(C) = (t/\alpha)^{d-1}\mu$. (We abuse notations a bit and refer to the $(d-1)$-dimensional volume and d-dimensional volume by the same notation.) Thus,

$$\mathrm{Vol}(R) = \int_{t=0}^{\alpha} \mathrm{Vol}(C(t))\,dt = \int_{t=0}^{\alpha} (t/\alpha)^{d-1}\,dt = \frac{\mu}{\alpha^{d-1}} \int_{t=0}^{\alpha} t^{d-1}\,dt = \frac{\mu}{\alpha^{d-1}} \cdot \frac{\alpha^d}{d} = \frac{\alpha\mu}{d}.$$

Thus, $\mathrm{Vol}(P^+) \geq \alpha\mu/\mathsf{d}$. Similar argumentation can be applied to $P^- = P \cap h^-$. Thus, we have $\mathrm{Vol}(P) \geq \rho\mu/\mathsf{d}$. ■

Lemma 11.1.3 *Let h be any hyperplane. We have* $\mathrm{Vol}(h \cap [0,1]^{\mathsf{d}}) \leq \mathsf{d}$.

Proof: Since the width of $\mathcal{H} = [0,1]^{\mathsf{d}}$ is 1 and thus the projection of $\mathcal{H}$ on the direction perpendicular to h is of length ≥ 1. As such, by Lemma 11.1.2 if $\mathrm{Vol}(h \cap \mathcal{H}) > \mathsf{d}$ then $\mathrm{Vol}(\mathcal{H}) > 1$. A contradiction. ■

Lemma 11.1.4 *Let $P \subseteq [0,1]^{\mathsf{d}}$ be a convex body. Let $\mu = \mathrm{Vol}(P)$. Then $\omega(P) \geq \mu/\mathsf{d}$ and P contains a ball of radius $\mu/(2\mathsf{d}^2)$.*

Proof: By Lemma 11.1.3, any hyperplane cut P in a set of volume at most d. Thus, $\mu = \mathrm{Vol}(P) \leq \omega(P)\mathsf{d}$. Namely, $\omega(P) \geq \mu/\mathsf{d}$.

Next, let $\mathcal{E}$ be the largest volume ellipsoid that is contained inside P. By John's theorem, we have that $P \subseteq d\mathcal{E}$. Let α be the length of the shortest axis of $\mathcal{E}$. Clearly, $\omega(P) \leq 2\mathsf{d}\alpha$, since $\omega(d\mathcal{E}) = 2\mathsf{d}\alpha$. Thus $2\mathsf{d}\alpha \geq \mu/\mathsf{d}$. This implies that $\alpha \geq \mu/(2\mathsf{d}^2)$.

Thus, $\mathcal{E}$ is an ellipsoid with its shortest axis is of length $\alpha \geq \mu/(2\mathsf{d}^2)$. In particular, $\mathcal{E}$ contains a ball of radius α, which is in turn contained inside P. ■

In Lemma 11.1.4 we used the fact that $r(P) \geq \omega(P)/(2\mathsf{d})$ (which we proved using John's theorem), where $r(P)$ is the radius of the largest ball enclosed inside P. Not surprisingly, considerably better bounds are known. In particular, it is known that $\omega(P)/(2\sqrt{\mathsf{d}}) \leq r(p)$ for add dimension, and $\omega(P)/(2(\mathsf{d}+1)/\sqrt{\mathsf{d}+2}) \leq r(P)$. Thus, $r(p) \geq \omega(P)/(2\sqrt{\mathsf{d}+1})$ [GK92]. Plugging this fact into the proof of Lemma 11.1.4, will give us slightly better result.

11.2 Approximating the Diameter

Lemma 11.2.1 *Given a point set S in $\mathbb{R}^{\mathsf{d}}$, one can compute in $O(n\mathsf{d})$ time a pair of points $s,t \in S$, such that $|st| \leq \mathrm{diam}(S) \leq \min(2, \sqrt{\mathsf{d}})|st|$.*

Proof: Let B be the minimum axis-parallel box containing S, and let s and t be the points in S that define the longest edge of B, whose length is denoted by l. By the diameter definition, $\|st\| \leq \mathrm{diam}(S)$, and clearly, $\mathrm{diam}(S) \leq \sqrt{d}\,l \leq \sqrt{d}\|st\|$. The points s and t are easily found in $O(n\mathsf{d})$ time.

Alternatively, pick a point $s' \in S$, and compute its furthest point $t' \in S$. Next, let a,b be the two points realizing the diameter. We have $\mathrm{diam}(S) = \|ab\| \leq \|as'\| + \|s'b\| \leq 2\|s't;\|$. Thus, $\|s't'\|$ is 2-approximation to the diameter of P. ■

11.3 Approximating the minimum volume bounding box

11.3.1 Constant Factor Approximation

Lemma 11.3.1 *Given a set* $\mathbf{P}$ *of n points in* $\mathbb{R}^{\mathsf{d}}$*, one can compute in* $O(\mathsf{d}^2 n)$ *time a bounding box* $B(\mathbf{P})$ *with* $\mathrm{Vol}(B_{\mathrm{opt}}(\mathbf{P})) \leq \mathrm{Vol}(B(\mathbf{P})) \leq 2^{\mathsf{d}}\mathsf{d}!\,\mathrm{Vol}(B_{\mathrm{opt}}(\mathbf{P}))$*, where B_{opt} is the minimum volume bounding box of* $\mathbf{P}$.

Furthermore, there exists a vector $v \in \mathbb{R}^{\mathsf{d}}$, such that $\overline{c} \cdot B + v \subseteq \mathcal{CH}(\mathbf{P})$. constant $\overline{c} = 1/\left(2^{\mathsf{d}}\mathsf{d}!\mathsf{d}^{5/2}\right)$.

Proof: By using the algorithm of Lemma 11.2.1 we compute in $O(n)$ time two points $s, t \in \mathbf{P}$ which form a 2-approximation of the diameter of $\mathbf{P}$. For the simplicity of exposition, we assume that st is on the x_d-axis (i.e., the line $\ell \equiv \cup_x(0, \ldots, 0, x)$), and there is one point of S that lies on the hyperplane $h \equiv x_d = 0$, an that $x_d \geq 0$ for all points of $\mathbf{P}$.

Let $\mathbf{P}'$ be the orthogonal projection of $\mathbf{P}$ into h, and let I be the shortest interval on ℓ which contain the projection of $\mathbf{P}$ into ℓ_s.

By recursion, we can compute a bounding box B' of $\mathbf{P}'$ in h. Let the bounding box be $B = B' \times I$. Note, that in the bottom of the recursion, the point-set is one dimensional, and the minimum interval containing the points can be computed in linear time.

Clearly, $\mathbf{P} \subseteq B$, and thus we only need to bound the quality of approximation. We next show that $\mathrm{Vol}(B) \geq \mathrm{Vol}(P)/c_d$, where $C = \mathcal{CH}(\mathbf{P})$, and $c_d = 2^d \cdot d!$. We prove this by induction on the dimension. For $d = 1$ the claim trivially holds. Otherwise, by induction, that $\mathrm{Vol}(B') \geq \mathrm{Vol}(C')/c_{d-1}$, where $C' = \mathcal{CH}(\mathbf{P}')$.

For a point $p \in C'$, let ℓ_p be the line parallel to x_d-axis passing through p. Let $L(p)$ be the minimum value of x_d for the points of ℓ_p lying inside C, and similarly, let $U(p)$ be the maximum value of x_d for the points of ℓ_p lying inside C. That is $\ell_p \cap C = [L[p), U(p)]$. Clearly, since C is convex, the function $L(\cdot)$ is concave, and $U(\cdot)$ is convex. As such, $\gamma(p) = U(p) - L(p)$ is a convex function, being the difference between a convex and a concave function. In particular, $\gamma(\cdot)$ induces the following convex body

$$U = \bigcup_{x \in C'} \Big[(x, 0), (x, \gamma(x))\Big].$$

Clearly, $\mathrm{Vol}(U) = \mathrm{Vol}(C)$. Furthermore, $\gamma((0, \ldots, 0)) \geq \|st\|$ and U is shaped like a "pyramid" its base is on the hyperplane $x_d = 0$ is the set C', and the segment $[(0, \ldots, 0), (0, \ldots, 0, \|st\|)]$ is contained inside it. Thus,

$$\mathrm{Vol}(C) = \mathrm{Vol}(U) \geq |st| \, \mathrm{Vol}(C')/d,$$

by Lemma 11.1.2. Let $r = |I|$ be the length of the projection of S into the line ℓ, we have that $r \leq 2\|st\|$. Thus,

$$\mathrm{Vol}(B) = \mathrm{Vol}(B') \, |I| \leq \mathrm{Vol}(B')\|st\| \, 2 \leq \mathrm{Vol}(C') \cdot c_{d-1}\|st\| \, 2.$$

On the other hand,

$$\mathrm{Vol}(B_{\mathrm{opt}}(\mathbf{P})) \geq \mathrm{Vol}(C) \geq \frac{\mathrm{Vol}(C')\|st\|}{d} \geq \frac{(\mathrm{Vol}(B')/c_{d-1}) \cdot (2\|st\|)}{2d} \geq \frac{\mathrm{Vol}(B') \, |I|}{2c_{d-1}d} = \frac{\mathrm{Vol}(B)}{2^d d!}.$$

Let T be an affine transformation that maps B to the unit hypercube $\mathcal{H} = [0, 1]^d$. Observe that $\mathrm{Vol}(T(C)) \geq 1/c_d$. By Lemma 11.1.4, there is ball $\mathbf{b}$ of radius $r \geq 1/(c_d \cdot 2d^2)$ contained inside $T(C)$. The ball $\mathbf{b}$ contains a hypercube of sidelength $2r/\sqrt{d} \geq 2/(2d^2\sqrt{d}c_d)$. Thus, for $\overline{c} = 1/\big(2^d d! d^{5/2}\big)$, there exists a vector $v' \in \mathbb{R}^d$, such that $\overline{c}\mathcal{H} + v' \subseteq T(C)$. Thus, applying T^{-1} to both sides, we have that there exists a vector $v \in \mathbb{R}^d$, such that $\overline{c} \cdot B + v = \overline{c} \cdot T^{-1}(\mathcal{H}) + v \subseteq C$. ■

11.4 Exact Algorithms

11.4.1 An exact algorithm 2d

Let $\mathbf{P}$ be a set of points in the plane. We compute the convex hull $C = \mathcal{CH}(\mathbf{P})$. Next, we rotate to lines parallel to each other, which touches the boundary of C. This can be easily done in linear time. We can also rotate two parallel lines which are perpendicular to the first set. Those four lines together induces a rectangle. It is easy to observe that during this rotation, we will encounter the minimum area rectangle. The function those lines define, changes every time the lines changes the vertices they rotate around. This happens $O(n)$ time. When the vertices the lines rotate around are fixed, the area function is a constant size function, and as such its minimum/maximum can be computed in linear time. Thus, the minimum volume bounding box, can be computed in $O(n \log n)$ time.

An important property of the minimum area rectangle, is that one of the edges of the convex hull lie on one of the bounding rectangle edges. We will refer to this edge as being *flush*. Thus, there are only n possibilities we have to check.

11.4.2 An exact algorithm 3d

Let $\mathbf{P}$ be a set of points in $\mathbb{R}^3$, our purpose is to compute the minimum volume bounding box B_{opt} of $\mathbf{P}$. It is easy to verify that B_{opt} must touch $\mathbf{P}$ on every one of its faces. In fact, consider an edge e for the bounding box B_{opt}. Clearly, if we project the points in the direction of e into a perpendicular plane h, it must hold that the projection of B_{opt} into this plane is a minimum area rectangle. As such, it has one flush edges, which corresponds to a flush edge of the convex hull of $\mathbf{P}$ that must lie on a face of B_{opt}. In fact, there must be two adjacent faces of B_{opt} that have flush edges of $\mathcal{CH}(\mathbf{P})$ on them.

Otherwise, consider a face f of B_{opt} that has an edge flush on it. All the four adjacent faces of B_{opt} do not have flush edges on them. But thats not possible, since we can project the points in the direction of the normal of f, and argue that in the projection there must be a flush edge. This flush edge, corresponds to an edge of $\mathcal{CH}(\mathbf{P})$ that lies on one of the faces of B_{opt} that is adjacent to f.

Lemma 11.4.1 *If B_{opt} is the minimum volume bounding box of $\mathbf{P}$, then it has two adjacent faces which are flush.*

This provides us with a natural algorithm to compute the minimum volume bounding box. Indeed, let us check all possible pair of edges $e, e' \in \mathcal{CH}(\mathbf{P})$. For each such pair, compute the minimum volume bounding box that has e and e' as flush.

Consider the normal $\vec{n}$ of the face of a bounding box that contains e. The normal $\vec{n}$ lie on a great circle on the sphere of directions, which are all the directions that are orthogonal to e. Let us parameterize $\vec{n}$ by a point on this normal. Next, consider the normal $\vec{n'}$ to the face that is flush to e'. Clearly, $\vec{n'}$ is orthogonal both to e' and $\vec{n}$. As such, we can compute this normal in constant time. Similarly, we can compute the third direction of the bounding box using vector product in const time. Thus, if e and e' are fixed, there is one dimensional family of bounding boxes of $\mathbf{P}$ that have e and e' flush on them, and comply with all the requirements to be a minimum volume bounding box.

It is now easy to verify that we can compute the representation of this family of bounding boxes, by tracking what vertices of the convex-hull the bounding boxes touches (i.e., this is similar to the rotating calipers algorithm, but one has to be more careful about the details). This can be done in linear time, and as such, one con compute the minimum volume bounding box in this family in linear time. Doing this for all pair of edges, results in $O(n^3)$ time algorithm, where $n = |\mathbf{P}|$.

Theorem 11.4.2 *Let $\mathbf{P}$ be a set of n points in $\mathbb{R}^3$. One can compute the minimum volume bounding box of $\mathbf{P}$ in $O(n^3)$ time.*

11.5 Approximating the Minimum Volume Bounding Box in Three Dimensions

Let $\mathbf{P}$ be a set of n points in $\mathbb{R}^3$, and let B_{opt} denote the minimum volume bounding box of $\mathbf{P}$. We remind the reader, that for two sets A and B in $\mathbb{R}^3$. The *Minkowski sum* of A and B is the set $A \oplus B = \left\{ a + b \;\middle|\; a \in A, b \in B \right\}$.

Let $B = B(\mathbf{P})$ be the bounding box of $\mathbf{P}$ computed by Lemma 11.3.1, and let B_ε be a translated copy of $\frac{\varepsilon}{\overline{c}}B$ centered at the origin, where $\overline{c}$ is an appropriate constant to be determined shortly. In addition, define $\mathcal{Q} = \mathcal{CH}(\mathbf{P}) \oplus B_\varepsilon$ and $\mathcal{G} = \mathrm{G}(\frac{1}{2}B_\varepsilon)$ denote the grid covering space, where every grid cell is a translated copy of $B_\varepsilon/2$. We approximate $\mathbf{P}$ on $\mathcal{G}$. For each point $p \in \mathbf{P}$ let $\mathcal{G}(p)$ be the set of eight vertices of the cell of $\mathcal{G}$ that contains p, and let $S_\mathcal{G} = \cup_{p\in S}\mathcal{G}(p)$. Define $\mathcal{P} = \mathcal{CH}(S_\mathcal{G})$. Clearly, $\mathcal{CH}(\mathbf{P}) \subseteq P \subseteq \mathcal{Q}$. Moreover, one can compute $\mathcal{P}$ in $O(n + (1/\varepsilon^2)\log(1/\varepsilon))$ time. On the other hand, $\mathcal{P} \subseteq B \oplus B_\varepsilon$. The latter term is a box which contains at most $k = 2\overline{c}/\varepsilon + 1$ grid points along each of the directions set by B, so k is also an upper bound for the number of grid points contained by $\mathcal{P}$ in each direction. As such, the convex hull of $\mathcal{CH}(P)$ is $O(k^2)$, as every grid line can contribute at most two vertices to the convex hull. Let $\mathbf{P}''$ the set of vertices of $\mathcal{P}$. We next apply the exact algorithm of Theorem 11.4.2 to $\mathbf{P}''$. Let $\widehat{B}$ denote the resulting bounding box.

It remains to show that $\widehat{B}$ is a $(1+\varepsilon)$-approximation of $B_{\mathrm{opt}}(\mathbf{P})$. Let $B^\varepsilon_{\mathrm{opt}}$ be a translation of $\frac{\varepsilon}{4}B_{\mathrm{opt}}(\mathbf{P})$ that contains B_ε. (The existence of $B^\varepsilon_{\mathrm{opt}}$ is guaranteed by Lemma 11.3.1, if we take $\overline{c} = 160$.) Thus, $\mathbf{P}'' \subseteq \mathcal{CH}(\mathbf{P}) \oplus B_\varepsilon \subseteq \mathcal{CH}(\mathbf{P}) \oplus B^\varepsilon_{\mathrm{opt}} \subseteq B_{\mathrm{opt}}(\mathbf{P}) \oplus B^\varepsilon_{\mathrm{opt}}$. Since $B_{\mathrm{opt}}(\mathbf{P}) \oplus B^\varepsilon_{\mathrm{opt}}$ is a box, it is a bounding box of P and therefore also of $\mathcal{CH}(\mathbf{P})$. Its volume is

$$\mathrm{Vol}(B_{\mathrm{opt}}(\mathbf{P}) \oplus B^\varepsilon_{\mathrm{opt}}) = \left(1 + \frac{\varepsilon}{4}\right)^3 \mathrm{Vol}(B_{\mathrm{opt}}(\mathbf{P})) < (1+\varepsilon)\,\mathrm{Vol}(B_{\mathrm{opt}}(\mathbf{P})),$$

as desired. (The last inequality is the only place where we use the assumption $\varepsilon \le 1$.)

To recap, the algorithm consists of the four following steps:

1. Compute the box $B(\mathbf{P})$ (see Lemma 11.3.1) in $O(n)$ time.

2. Compute the point set $S_\mathcal{G}$ in $O(n)$ time.

3. Compute $P = \mathcal{CH}(S_\mathcal{G})$ in $O(n + (1/\varepsilon^2)\log(1/\varepsilon))$ time. This is done by computing the convex hull of all the extreme points of $S_\mathcal{G}$ along vertical lines of $\mathcal{G}$. We have $O(1/\varepsilon^2)$ such points, thus computing their convex hull takes $O((1/\varepsilon^2)\log(1/\varepsilon))$ time. Let $\mathbf{P}''$ be the set of vertices of P.

4. Compute $B_{\mathrm{opt}}(\mathbf{P}'')$ by the algorithm of Theorem 11.4.2. This step requires $O((1/\varepsilon^2)^3) = O(1/\varepsilon^6)$ time.

Theorem 11.5.1 *Let* $\mathbf{P}$ *be a set of n points in* $\mathbb{R}^3$*, and let* $0 < \varepsilon \le 1$ *be a parameter. One can compute in* $O(n + 1/\varepsilon^6)$ *time a bounding box* $B(\mathbf{P})$ *with* $\mathrm{Vol}(B(\mathbf{P})) \le (1+\varepsilon)\,\mathrm{Vol}(B_{\mathrm{opt}}(\mathbf{P}))$.

Note that the box $B(S)$ computed by the above algorithm is most likely not minimal along its directions. The minimum bounding box of $\mathbf{P}$ homothet of $B(S)$ can be computed in additional $O(n)$ time.

11.6 Bibliographical notes

Our exposition follows roughly the work of Barequet and Har-Peled [BH01]. However, the basic idea, of finding the diameter, projecting along it and recursively finding a good bounding box on the projected input, is much older, and can be traced back to the work of Macbeath [Mac50].

For approximating the diameter, one can find in linear time a $(1/\sqrt{3})$-approximation of the diameter in *any* dimension; see [EK89].

The rotating calipers algorithm (Section 11.4.1) is due to Toussaint [Tou83]. The elegant extension of this algorithm to the computation of the exact minimum volume bounding box algorithm is due to O'Rourke [O'R85].

Lemma 11.3.1 is (essentially) from [BH01].

The current constants in Lemma 11.3.1 are unreasonable, but there is no reason to believe they are tight.

Conjecture 11.6.1 *The constants in Lemma 11.3.1 can be improved to be polynomial in the dimension.*

Coresets. One alternative approach to the algorithm of Theorem 11.5.1 is to construct G using $B_\varepsilon/2$ as before, and picking from each non-empty cell of G, one point of **P** as a representative point. This results in a set $\mathcal{S}$ of $O(1/\varepsilon^2)$ points. Compute the minimum volume bounding box $\mathcal{S}$ using the exact algorithm. Let B denote the resulting bounding box. It is easy to verify that $(1+\varepsilon)B$ contains **P**, and that it is a $(1+\varepsilon)$-approximation to the optimal bounding box of **P**. The running time of the new algorithm is identical. The interesting property is that we are running the exact algorithm on on a subset of the input.

This is a powerful technique for approximation algorithms. You first extract a small subset from the input, and run an exact algorithm on this input, making sure that the result provides the required approximation. The subset $\mathcal{S}$ is referred to as *coreset* of B as it preserves a geometric property of P (in our case, the minimum volume bounding box). We will see more about this notion in the following lectures.

Chapter 12

VC Dimension, ε-nets and ε-approximation

"I've never touched the hard stuff, only smoked grass a few times with the boys to be polite, and that's all, though ten is the age when the big guys come around teaching you all sorts to things. But happiness doesn't mean much to me, I still think life is better. Happiness is a mean son of a bitch and needs to be put in his place. Him and me aren't on the same team, and I'm cutting him dead. I've never gone in for politics, because somebody always stand to gain by it, but happiness is an even crummier racket, and their ought to be laws to put it out of business."
– – Momo, Emile Ajar

12.1 VC Dimension

Definition 12.1.1 A *range space* S is a pair (X, R), where X is a (finite or infinite) set and R is a (finite or infinite) family of subsets of X. The elements of X are *points* and the elements of R are *ranges*. For $A \subseteq X$, $P_R(A) = \left\{ r \cap A \;\middle|\; r \in R \right\}$ is the *projection* of R on A.

If $P_R(A)$ contains all subsets of A (i.e., if A is finite, we have $|P_R(A)| = 2^{|A|}$) then A is *shattered* by R.

The *Vapnik-Chervonenkis* dimension (or VC-dimension) of S, denoted by $\mathrm{VC}(S)$, is the maximum cardinality of a shattered subset of X. It there are arbitrarily large shattered subsets then $\mathrm{VC}(S) = \infty$.

12.1.1 Examples

Example 12.1.2 Let $X = \mathbb{R}^2$, and let R be the set of disks in the plane. Clearly, for three points in the plane $1, 2, 3$, one can find 8 disks that realize all possible 2^3 different subsets. See Figure 12.1.

But can disks shatter a set with four points? Consider such a set P of four points, and there are two possible options. Either the convex-hull of P has three points on its boundary, and in this case, the subset having those vertices in the subset but not including the middle point is impossible, by convexity. Alternatively, if all four points are vertices of the convex hull, and they are p_1, p_2, p_3, p_4 along the boundary of the convex hull, either the set $\{p_1, p_3\}$ or the set $\{p_2, p_4\}$ is not realizable. Indeed, if both options are realizable, then consider the two disks D_1, D_2 the realizes those assignments. Clearly, D_1 and D_2 must intersect in four points, but this is not possible, since two disks have at most two intersection points. See Figure 12.1 (b).

Example 12.1.3 Consider the range space $S = (\mathbb{R}^2, R)$, where R is the set of all (closed) convex sets in the plane. We claim that the $\mathrm{VC}(S) = \infty$. Indeed, consider a set U of n points $p_1, \ldots, p_n$ all lying on the boundary of the unit circle in the plane. Let V be any subset of U, and consider the convex-hull $\mathcal{CH}(V)$. Clearly, $\mathcal{CH}(V) \in R$, and furthermore, $\mathcal{CH}(V) \cap U = V$. Namely, any subset of U is realizable by S. Thus, S can shatter sets of arbitrary size, and its VC dimension is unbounded.

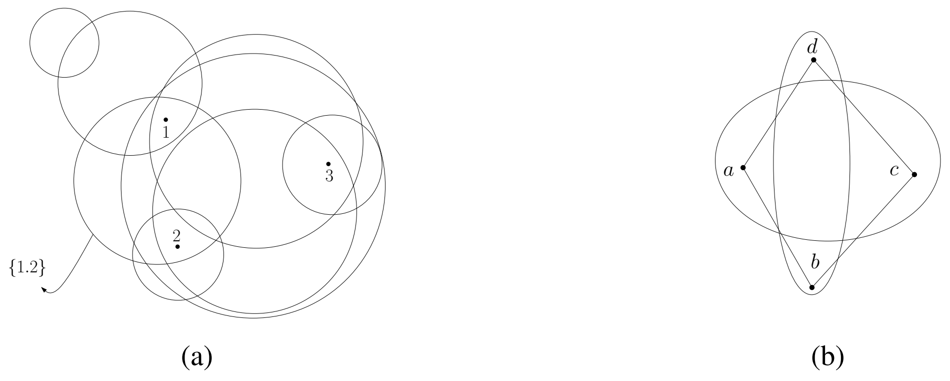


Figure 12.1: Disks in the plane can shatter three points.

Example 12.1.4 Let $S = (X, R)$, where $X = \mathbb{R}^d$ and R is the set of all (closed) halfspaces in $\mathbb{R}^d$. To see what is the VC dimension of S, we nee the following result of Radon:

> **Theorem 12.1.5 (Randon's Lemma)** *Let A be a set of $d + 2$ points in $\mathbb{R}^d$. Then, there exists two disjoint subsets C, D of A, such that $\mathcal{CH}(C) \cap \mathcal{CH}(D) \neq \emptyset$.*
>
> *Proof:* The points $p_1, \ldots, p_{d+2}$ of A are linearly dependent. As such, there exists $\beta_1, \ldots, \beta_{d+2}$, non all of them zero, such that $\sum_i \beta_i p_i = 0$ and $\sum_i \beta_i = 0$ (to see that, remember that the affine subspace spanned by $p_1, \ldots, p_{d+2}$ is induced by all points that can be represented as $p_1 + \sum_{i=2}^{d+2} \alpha_i (p_i - p_1)$ where $\sum_i \alpha_i = 0$). Assume, for the sake of simplicity of exposition, that the $\beta_1, \ldots \beta_k \geq 0$ and $\beta_{k+1}, \ldots, \beta_{d+2} < 0$. Furthermore, let $\mu = \sum_{i=1}^{k} \beta_i$. We have that
>
> $$\sum_{i=0}^{k} \beta_i p_i = - \sum_{i=k+1}^{d+2} \beta_i p_i.$$
>
> In particular, $v = \sum_{i=0}^{k} (\beta_i/\mu) p_i$ is a point in the $\mathcal{CH}(\{p_1, \ldots, p_k\})$ and $\sum_{i=k+1}^{d+2} -(\beta_i/\mu) p_i \in \mathcal{CH}(\{p_{k+1}, \ldots, p_{d+2}\})$. We conclude that v is in the intersection of the two convex hulls, as required. ■

In particular, this implies that if a set Q of $d + 2$ points is being shattered by S, we can partition this set Q into two disjoint sets A and B such that $\mathcal{CH}(A) \cap \mathcal{CH}(B) \neq \emptyset$. It should now be clear that any halfspace h^+ containing all the points of A, must also contain a point of the $\mathcal{CH}(B)$. But this implies that a point of B must be in h^+. Namely, the subset A can not be realized by a halfspace, which implies that Q can not be shattered. Thus $\mathrm{VC}(S) < d + 2$. It is also easy to verify that the regular simplex with $d + 1$ vertices is being shattered by S. Thus, $\mathrm{VC}(S) = d + 1$.

Lemma 12.1.6 *Let $S = (X, R)$ and $S' = (X, R')$ be two range spaces of dimension d and d', respectively, where $d, d' > 1$. Let $\widehat{R} = \left\{ r \cup r' \;\middle|\; r \in R, r' \in R' \right\}$. Then, for the range space $\widehat{S} = (X, \widehat{R})$, we have that* $\mathrm{VC}\left(\widehat{S}\right) = O((d + d') \log(d + d'))$

Proof: Let A be a set of n points in X that are being shattered by $\widehat{S}$. There are $g(n, d)$ and $G(n, d')$ different assignments for the elements of A by ranges of R and R', respectively. Every subset C of A realized by $\widehat{r} \in \widehat{R}$, is a union of two subsets $A \cap r$ and $A \cap r'$ where $r \in R$ and $r' \in R'$. Thus, the number of different subsets of A realized by $\widehat{S}$ is bounded by $g(n, d) g(n, d')$. Thus, $2^n \leq n^d n^{d'}$, for $d, d' > 1$. We conclude $n \leq (d + d') \lg n$, which implies that $n \leq O((d + d') \log(d + d'))$. ■

12.2 On ε-nets and ε-sampling

Let

$$g(d,n) = \sum_{i=0}^{d} \binom{n}{i}.$$

Note that for all $n, d \geq 1$, $g(d,n) = g(d,n-1) + g(d-1,n-1)$

Lemma 12.2.1 (Sauer's Lemma) *If (X,R) is a range space of* VC*-dimension d with $|X| = n$ points then $|R| \leq g(d,n)$.*

Proof: The claim trivially holds for $d = 0$ or $n = 0$.

Let x be any element of X, and consider the sets

$$R_x = \left\{ r \setminus \{x\} \;\middle|\; x \in r, r \in R, r \setminus \{x\} \in R \right\}$$

and

$$R \setminus x = \left\{ r \setminus \{x\} \;\middle|\; r \in R \right\}.$$

Observe that $|R| = |R_x| + |R \setminus x|$ (Indeed, if r does not contain x than it is counted in R_x, if does contain x but $r \setminus x \notin R$, then it is also counted in R_x. The only remaining case is when both $r \setminus \{x\}$ and $r \cup \{x\}$ are in R, but then it is being counted once in R_x and once in $R \setminus x$.)

Observe that R_x has VC dimension $d-1$, as the largest set that can be shattered is of size $d-1$. Indeed, any set $A \subset X$ shattered by R_x, implies that $A \cup \{x\}$ is shattered in R.

Thus,

$$|R| = |R_x| + |R \setminus x| = g(n-1, d-1) + g(n-1, d) = g(d,n),$$

by induction. ∎

By applying Lemma 12.2.1, to a finite subset of X, we get:

Corollary 12.2.2 *If (X,R) is a range space of* VC*-dimension d then for every finite subset A of X, we have $|P_R(A)| \leq g(d,|A|)$.*

Definition 12.2.3 Let (X,R) be a range space, and let A be a finite subset of X. For $0 \leq \varepsilon \leq 1$, a subset $B \subseteq A$, is an *ε-sample* for A if for any range $r \in R$, we have

$$\left| \frac{|A \cap r|}{|A|} - \frac{|B \cap r|}{|B|} \right| \leq \varepsilon.$$

Similarly, $N \subseteq A$ is an *ε-net* for A, if for any range $r \in R$, if $|r \cap A| \geq \varepsilon |A|$ implies that r contains at least one point of N (i.e., $r \cap N \neq \emptyset$).

Theorem 12.2.4 *There is a positive constant c such that if (X,R) is any range space of* VC*-dimension at most d, $A \subseteq X$ is a finite subset and $\varepsilon, \delta > 0$, then a random subset B of cardinality s of A where s is at least the minimum between $|A|$ and*

$$\frac{c}{\varepsilon^2}\left(d \log \frac{d}{\varepsilon} + \log \frac{1}{\delta} \right)$$

is an ε-sample for A with probability at least $1 - \delta$.

Theorem 12.2.5 *Let (X, R) be a range space of* VC*-dimension d, let A be a finite subset of X and suppose* $0 < \varepsilon, \delta < 1$. *Let N be a set obtained by m random independent draws from A, where*

$$m \geq \max\left(\frac{4}{\varepsilon} \log \frac{2}{\delta}, \frac{8d}{\varepsilon} \log \frac{8d}{\varepsilon}\right). \tag{12.1}$$

Then N is an ε-net for A with probability at least $1 - \delta$.

12.3 Proof of the ε-net Theorem

Let (X, R) be a range space of VC-dimension d, and let A be a subset of X of cardinality n. Suppose that m satisfiers Eq. (12.1). Let $N = (x_1, \ldots, x_m)$ be the sample obtained by m independent samples from A (the elements of N are not necessarily distinct, and thats why we treat N as a ordered set). Let E_1 be the probability that N fails to be an ε-net. Namely,

$$E_1 = \left\{ \exists r \in R \;\middle|\; |r \cap A| \geq \varepsilon n, r \cap N = \emptyset \right\}.$$

(Namely, there exists a "heavy" range r that does not contain any point of N.) To complete the proof, we must show that $\mathbf{Pr}[E_1] \leq \delta$. Let $T = (y_1, \ldots, y_m)$ be another random sample generated in a similar fashion to N. Let E_2 be the event that N fails, but T "works", formally

$$E_2 = \left\{ \exists r \in R \;\middle|\; |r \cap A| \geq \varepsilon n, r \cap N = \emptyset, |r \cap T| \geq \frac{\varepsilon m}{2} \right\}.$$

(We remind the reader that $|r \cap T|$ denotes the number of elements of T belong to r.)

Intuitively, since $E_T\Big[|r \cap T|\Big] \geq \varepsilon m$, then for the range r that N fails for, we have with "good" probability that $|r \cap T| \geq \frac{\varepsilon n}{2}$. Namely, E_1 and E_2 have more or less the same probability.

Claim 12.3.1 $\mathbf{Pr}[E_2] \leq \mathbf{Pr}[E_1] \leq 2\,\mathbf{Pr}[E_2]$.

Proof: Clearly, $E_2 \subseteq E_1$, and thus $\mathbf{Pr}[E_2] \leq \mathbf{Pr}[E_1]$. As for the other part, note that $\mathbf{Pr}\Big[E_2 \;\Big|\; E_1\Big] = \mathbf{Pr}[E_2 \cap E_1] \,/\, \mathbf{Pr}[E_1] = \mathbf{Pr}[E_2] \,/\, \mathbf{Pr}[E_1]$. It is thus enough to show that $\mathbf{Pr}\Big[E_2 \;\Big|\; E_1\Big] \geq 1/2$.

Assume that E_1 occur. There is $r \in R$, such that $|r \cap A| > \varepsilon n$ and $r \cap N = \emptyset$. The required probability is at least the probability that for this specific r, we have $|r \cap T| \geq \frac{\varepsilon n}{2}$. However, $|r \cap T|$ is a binomial variable with expectation εm, and variance $\varepsilon(1 - \varepsilon)m \leq \varepsilon m$. Thus, by Chebychev inequality (Theorem 19.1.2),

$$\mathbf{Pr}\left[|r \cap T| < \frac{\varepsilon m}{2}\right] \leq \mathbf{Pr}\left[\Big||r \cap T| - \varepsilon m\Big| > \frac{\varepsilon m}{2}\right] \mathbf{Pr}\left[\Big||r \cap T| - \varepsilon m\Big| > \frac{\sqrt{\varepsilon m}}{2} \sqrt{\varepsilon m}\right] \leq \frac{4}{\varepsilon m} \leq \frac{1}{2},$$

by Eq. (12.1). Thus, $\mathbf{Pr}[E_2] \,/\, \mathbf{Pr}[E_1] = \mathbf{Pr}\Big[|r \cap T| \geq \frac{\varepsilon n}{2}\Big] = 1 - \mathbf{Pr}\Big[|r \cap T| < \frac{\varepsilon m}{2}\Big] \geq \frac{1}{2}$. ■

Thus, it is enough to bound the probability of E_2. Let

$$E_2' = \left\{ \exists r \in R \;\middle|\; r \cap N = \emptyset, |r \cap T| \geq \frac{\varepsilon m}{2} \right\},$$

Clearly, $E_2 \subseteq E_2'$. Thus, bounding the probability of E_2' is enough to prove the theorem. Note however, that a shocking thing happened! We no longer have A as participating in our event. Namely, we turned bounding an event that depends on a global quantity, into bounding a quantity that depends only on local quantity/experiment. This is the crucial idea in this proof.

Claim 12.3.2 $\mathbf{Pr}[E_2] \leq \mathbf{Pr}\Big[E_2'\Big] \leq g(d, 2m)2^{-em/2}$.

Proof: We imagine that we sample the elements of $N \cup T$ together, by picking $Z = (z_1, \ldots, z_{2m})$ independently from A. Next, we randomly decide the m elements of Z that go into N, and remaining elements go into T. Clearly,

$$\mathbf{Pr}\Big[E_2'\Big] = \sum_Z \mathbf{Pr}\Big[E_2' \,\Big|\, Z\Big]\mathbf{Pr}[Z].$$

Thus, from this point on, we fix the set Z, and we bound $\mathbf{Pr}\Big[E_2' \,\Big|\, Z\Big]$.

It is now enough to consider the ranges in the projection space $P_Z(R)$. By Lemma 12.2.1, we have $|P_Z(r)| \leq g(d, 2m)$.

Let us fix any $r \in P_Z(R)$, and consider the event

$$E_r = \left\{ r \cap N = \emptyset \text{ and } |r \cap T| > \frac{\varepsilon m}{2} \right\}.$$

For $k = |r \cap (N \cup T)|$, we have

$$\begin{aligned}
\mathbf{Pr}[E_r] &\leq \mathbf{Pr}\left[r \cap N = \emptyset \,\middle|\, |r \cap (N \cup T)| > \frac{\varepsilon m}{2} \right] = \frac{\binom{2m-k}{m}}{\binom{2m}{m}} \\
&= \frac{(2m-k)(2m-k-1)\cdots(m-k+1)}{2m(2m-1)\cdots(m+1)} \\
&= \frac{m(m-1)\cdots(m-k+1)}{2m(2m-1)\cdots(2m-k+1)} \leq 2^{-k} \leq 2^{-\varepsilon m/2}.
\end{aligned}$$

Thus,

$$\mathbf{Pr}\Big[E_2' \,\Big|\, Z\Big] \leq \sum_{r \in P_Z(R)} \mathbf{Pr}[E_r] \leq |P_Z(R)|\, 2^{-\varepsilon m/2} = g(d, 2m)2^{-\varepsilon m/2},$$

implying that $\mathbf{Pr}\Big[E_2'\Big] \leq g(d, 2m)2^{-\varepsilon m/2}$. ■

Proof of Theorem 12.2.5. By Lemma 12.3.1 and Lemma 12.3.2, we have $\mathbf{Pr}[E_1] \leq 2g(d, 2m)2^{-\varepsilon m/2}$. It is thus remains to verify that if m satisfies Eq. (12.1), then $2g(d, 2m)2^{-\varepsilon m/2} \leq \delta$. One can verify that this inequality is implied by Eq. (12.1).

Indeed, we know that $2m \geq 8d$ and as such $g(d, 2m) = \sum_{i=0}^{d} \binom{2m}{i} \leq \sum_{i=0}^{d} \frac{(2m)^i}{i!} \leq (2m)^d$, for $d > 1$. Thus, it is sufficient to show that the inequality $2(2m)^d 2^{-\varepsilon m/2} \leq \delta$ holds. By taking lg of both sides and rearranging, we have that this is equivalent to

$$\frac{\varepsilon m}{2} \geq d \lg(2m) + \lg \frac{2}{\delta}.$$

By our choice of m (see Eq. (12.1)), we have that $\varepsilon m/4 \geq \lg(2/\delta)$. Thus, we need to show that

$$\frac{\varepsilon m}{4} \geq d \lg(2m).$$

We verify this inequality for $m = \frac{8d}{\varepsilon} \lg \frac{8d}{\varepsilon}$, indeed

$$2d \lg \frac{8d}{\varepsilon} \geq d \lg\left(\frac{16d}{\varepsilon} \lg \frac{8d}{\varepsilon} \right).$$

This is equivalent to $\left(\frac{8d}{\varepsilon}\right)^2 \geq \frac{16d}{\varepsilon} \lg \frac{8d}{\varepsilon}$. Which is equivalent to $\frac{4d}{\varepsilon} \geq \lg \frac{8d}{\varepsilon}$, which is certainly true for $0 \leq \varepsilon 1$ and $d > 1$. Note that it is easy to verify that the inequality holds for $m \geq \frac{8d}{\varepsilon} \lg \frac{8d}{\varepsilon}$, by deriving both sides of the inequality.

This completes the proof of the theorem. ■

12.4 Exercises

Exercise 12.4.1 (Flip and Flop) [20 Points]

(a) **[5 Points]** Let $b_1, \ldots, b_{2m}$ be m binary bits. Let Ψ be the set of all permutations of $1, \ldots, 2m$, such that for any $\sigma \in \Psi$, we have $\sigma(i) = i$ or $\sigma(i) = m + i$, for $1 \leq i \leq m$, and similarly, $\sigma(m+i) = i$ or $\sigma(m+i) = m+i$. Namely, $\sigma \in \Psi$ either leave the pair $i, i+m$ in their positions, or it exchange them, for $1 \leq i \leq m$. As such $|\Psi| = 2^m$.

Prove that for a random $\sigma \in \Psi$, we have

$$\mathbf{Pr}\left[\left|\frac{\sum_{i=1}^m b_{\sigma(i)}}{m} - \frac{\sum_{i=1}^m b_{\sigma(i+m)}}{m}\right| \geq \varepsilon\right] \leq 2e^{-\varepsilon^2 m/2}.$$

(b) **[5 Points]** Let Ψ' be the set of all permutations of $1, \ldots, 2m$. Prove that for a random $\sigma \in \Psi'$, we have

$$\mathbf{Pr}\left[\left|\frac{\sum_{i=1}^m b_{\sigma(i)}}{m} - \frac{\sum_{i=1}^m b_{\sigma(i+m)}}{m}\right| \geq \varepsilon\right] \leq 2e^{-C\varepsilon^2 m/2},$$

where C is an appropriate constant. [**Hint:** Use (a), but be careful.]

(c) **[10 Points]** Prove Theorem 12.2.4 using (b).

12.5 Bibliographical notes

The exposition here is based on [AS00]. The usual exposition of the ε-net/ε-sample tend to be long and tedious in the learning literature. The proof of the ε-net theorem is due Haussler and Welzl [HW87]. The proof of the ε-sample theorem is due to Vapnik and Chervonenkis [VC71]. However, the importance of Vapnik and Chervonenkis result was not realized at the time, and only in the late eighties the strong connection to learning was established.

An alternative proof of both theorems exists via the usage of discrepancy. Using discrepancy, one can compute ε-samples and ε-nets deterministically. In fact, in some geometric cases, discrepancy yields better results than the ε-net and ε-sample theorem. See [Mat99, Cha01] for more details.

Exercise 12.4.1 is from Anthony and Bartlett [AB99].

Chapter 13

Approximating the Directional Width of a Shape

"From the days of John the Baptist until now, the kingdom of heaven suffereth violence, and the violent bear it away."
– – Matthew 11:12

13.1 Coreset for Directional Width

Let $\mathbf{P}$ be a set of points in $\mathbb{R}^{\mathsf{d}}$. For a vector $v \in \mathbb{R}^{\mathsf{d}}$, such that $v \neq 0$, let

$$\overline{\omega}(v, \mathbf{P}) = \max_{p \in \mathbf{P}} \langle v, p \rangle - \min_{p \in \mathbf{P}} \langle v, p \rangle,$$

denote the *directional width* of $\mathbf{P}$ in the direction of v.

A set $Q \subseteq P$ is a *ε-coreset for directional width*, if

$$\forall v \in \mathbb{S}^{(\mathsf{d}-1)} \quad \overline{\omega}(v, Q) \geq (1 - \varepsilon)\overline{\omega}(v, P).$$

Namely, the coreset Q provides a concise approximation to the directional width of $\mathbf{P}$. The usefulness of such a coreset might become clearer in the light of the following claim.

Claim 13.1.1 *Let $\mathbf{P}$ be a set of points in $\mathbb{R}^{\mathsf{d}}$, $0 < \varepsilon \leq 1$ a parameter and let Q be a δ-coreset of $\mathbf{P}$ for directional width, for $\delta = \varepsilon/(8\mathsf{d})$. Let $\mathcal{B}_{\mathrm{opt}}(Q)$ denote the minimum volume bounding box of Q. Let B' be the rescaling of $\mathcal{B}_{\mathrm{opt}}(Q)$ around its center by a factor of $(1 + 3\delta)$.*

Then, $\mathbf{P} \subset B'$, and in particular, $\mathrm{Vol}(B') \leq (1 + \varepsilon)\mathcal{B}_{\mathrm{opt}}(\mathbf{P})$, where $\mathcal{B}_{\mathrm{opt}}(\mathbf{P})$ denotes the minimum volume bounding box of $\mathbf{P}$.

Proof: Let v be a direction parallel to one of the edges of $\mathcal{B}_{\mathrm{opt}}(Q)$, and let ℓ be a line through the origin with the direction of v. Let I and I' be the projection of $\mathcal{B}_{\mathrm{opt}}(Q)$ and B', respectively, into ℓ. Let I_P be the interval formed by the projection of $\mathcal{CH}(\mathbf{P})$ into ℓ. We have that $I \subseteq I_P$ and $|I| \geq (1 - \delta)|I_P|$. The interval I' is the result of expanding I around its center point c by a factor of $1 + 3\delta$. In particular, the distance between c and the furthest endpoint of I_p is $\leq (1 - (1 - \delta)/2)\left|I_p\right| = (1 + \delta)\left|I_p\right|/2$. Thus, we need to verify that after the expansion of I it contains this endpoint. Namely,

$$(1 + 3\delta)\frac{1 - \delta}{2}|I_P| \geq \frac{1 + 2\delta - 3\delta^2}{2}|I_P| \geq \frac{1 + \delta}{2}|I_P|,$$

for $\delta \leq 1/3$. Thus, $I_P \subseteq I'$ and $\mathbf{P} \subseteq B'$.

Observe that $\mathrm{Vol}(B') \le (1 + 3\delta)^{\mathsf{d}} \mathrm{Vol}(\mathcal{B}_{\mathrm{opt}}(Q)) \le \exp(3\delta\mathsf{d}) \mathrm{Vol}(\mathcal{B}_{\mathrm{opt}}(Q)) \le (1 + \varepsilon)\mathcal{B}_{\mathrm{opt}}(Q) \le (1 + \varepsilon)\mathcal{B}_{\mathrm{opt}}(\mathbf{P})$. ■

It is easy to verify, that a coreset for directional width, also preserves (approximately) the diameter and width of the point set. Namely, it captures "well" the geometry of $\mathbf{P}$. Claim 13.1.1 hints on the connection between coreset for directional width and the minimum volume bounding box. In particular, if we have a good bounding box, we can compute a small coreset for directional width.

Lemma 13.1.2 *Let $\mathbf{P}$ be a set of n points in $\mathbb{R}^{\mathsf{d}}$, and let B be a bounding box of $\mathbf{P}$, such that $v + c_d B \subseteq \mathcal{CH}(\mathbf{P})$, where v is a vector in $\mathbb{R}^{\mathsf{d}}$, $c_d = (4\mathsf{d} + 1)\mathsf{d}$ and $c_d B$ denote the rescaling of B by a factor of c_d around its center point.*

Then, one can compute a ε-coreset $\mathcal{S}$ for directional width of $\mathbf{P}$. The size of the coreset is $O(1/\varepsilon^{\mathsf{d}-1})$, and construction time is $O\Big(n + \min(n, 1/\varepsilon^{\mathsf{d}-1})\Big)$.

Proof: We partition B into a grid, by breaking each edge of B into $M = \lceil 4/(\varepsilon c_d) \rceil$ equal length intervals (namely, we tile B with M^{d} copies of B/M). A cell in this grid is uniquely defined by a d-tuple $(i_1, \ldots, i_{\mathsf{d}})$. In particular, for a point $p \in \mathbf{P}$, lets $I(p)$ denote the ID of this point. Clearly, $I(p)$ can be computed in constant time.

Given a $(\mathsf{d} - 1)$-tuple $I = (i_1, \ldots, i_{\mathsf{d}-1})$ its *pillar* is the set of grid cells that have $(i_1, \ldots, i_{\mathsf{d}-1})$ as the first $\mathsf{d} - 1$ coordinates of their ID. Scan the points of $\mathbf{P}$, and for each pillar record the highest and lowest point encountered. Here highest/lowest refer to their value in the dth direction.

We claim that the resulting set $\mathcal{S}$ is the required coreset. Indeed, consider a direction $v \in \mathbb{S}^{(\mathsf{d}-1)}$, and a point $p \in \mathbf{P}$. Let q, q' be the highest and lowest points in P which are inside the pillar of p, and are thus in $\mathcal{S}$. Let $B_q, B_{q'}$ be the two grid cells containing q and q'. Clearly, the projection of $\mathcal{CH}(B_q \cup B_{q'})$ into the direction of v contains the projection of p into the direction of v. Thus, for a vertex u of B_q it is sufficient to show that

$$\overline{\omega}(v, \{u, q\}) \le \overline{\omega}(v, B/M) \le (\varepsilon/2)\overline{\omega}(v, \mathbf{P}),$$

since this implies that $\overline{\omega}(v, \mathcal{S}) \ge \overline{\omega}(v, \mathbf{P}) - 2\overline{\omega}(v, B/M) \ge (1 - \varepsilon)\overline{\omega}(v, \mathbf{P})$. Indeed,

$$\overline{\omega}(v, B/M) \le \overline{\omega}(v, B)/M \le \overline{\omega}(v, \mathbf{P})/(c_d M) \le \frac{\overline{\omega}(v, \mathbf{P})}{4/\varepsilon} \le \frac{\varepsilon}{4}\overline{\omega}(v, \mathbf{P}).$$

As for the preprocessing time, it requires a somewhat careful implementation. We construct a hash-table (of size $O(n)$) and store for every pillar the top and bottom points encountered. When handling a point this hash table can be updated in constant time. Once the coreset was computed, the coreset can be extracted from the hash-table in linear time. ■

Theorem 13.1.3 *Let $\mathbf{P}$ be a set of n points in $\mathbb{R}^{\mathsf{d}}$, and let $0 < \varepsilon < 1$ be a parameter. One can compute a ε-coreset $\mathcal{S}$ for directional width of $\mathbf{P}$. The size of the coreset is $O(1/\varepsilon^{\mathsf{d}-1})$, and construction time is $O\Big(n + \min(n, 1/\varepsilon^{\mathsf{d}-1})\Big)$.*

Proof: Compute a good bounding box of P using Theorem **??**. Then apply Lemma 13.1.2 to $\mathbf{P}$. ■

13.2 Smaller coreset for directional width

We call $\mathbf{P} \subseteq \mathbb{R}^{\mathsf{d}}$ *α-fat*, for $\alpha \le 1$, if there exists a point $p \in \mathbb{R}^{\mathsf{d}}$ and a hypercube $\overline{\mathcal{H}}$ centered at the origin so that

$$p + \alpha\overline{\mathcal{H}} \subset \mathcal{CH}(P) \subset p + \overline{\mathcal{H}}.$$

13.2.1 Transforming a set into a fat set

Lemma 13.2.1 *Let* $\mathbf{P}$ *be a set of* n *points in* $\mathbb{R}^{\mathsf{d}}$ *such that the volume of* $\mathcal{CH}(\mathbf{P})$ *is non-zero, and let* $\mathcal{H} = [-1,1]^{\mathsf{d}}$*. One can compute in* $O(n)$ *time an affine transform* τ *so that* $\tau(\mathbf{P})$ *is an* α*-fat point set satisfying* $\alpha\mathcal{H} \subset \mathcal{CH}(\tau(\mathbf{P})) \subset \mathcal{H}$*, where* α *is a positive constant depending on* d*, and so that a subset* $\mathcal{S} \subseteq \mathbf{P}$ *is an* ε*-coreset of* $\mathbf{P}$ *for directional width if and only if* $\tau(Q)$ *is an* ε*-coreset of* $\tau(\mathbf{P})$ *for directional width.*

Proof: Using the algorithm of Theorem **??** compute, in $O(n)$ time, a bounding B of P, such that there exists a vector $\vec{w}$ such that $\vec{w} + \frac{1}{\mathsf{d}(4\mathsf{d}+1)}B \subseteq \mathcal{CH}(\mathbf{P}) \subseteq B$.

Let T_1 be the linear transformation that translates the center of T to the origin. Clearly, $\mathcal{S} \subseteq \mathbf{P}$ is a ε-coreset for direction width of $\mathbf{P}$, if and only if $\mathcal{S}_1 = T_1(\mathcal{S})$ is an ε-coreset for directional width of $\mathbf{P}_1 = T_1(\mathbf{P})$. Next, let T_2 be a rotation that rotates $B_1 = T_1(B)$ such that its sides are parallel to the axises. Again, $\mathcal{S}_2 = T_2(\mathcal{S}_1)$ is a ε-coreset for $\mathbf{P}_2 = T_2(\mathbf{P}_1)$ if and only if $\mathcal{S}_1$ is a coreset for $\mathbf{P}_1$. Finally, let T_3 be a scaling of the axises such that $B_2 = T_2(B_1)$ is mapped to the hypercube $\mathcal{H}$.

Note, that T_3 is just a diagonal matrix. As such, for any $p \in \mathbb{R}^{\mathsf{d}}$, and a vector $v \in \mathbb{S}^{(d-1)}$ we have

$$\langle v, T_3 p\rangle = v^T T_3 p = \left(T_3^T v\right)^T p == \left\langle T_3^T v, p\right\rangle = \langle T_3 v, p\rangle .$$

Let $\mathcal{S}_3 = T_3(\mathcal{S}_2)$ and let $\mathbf{P}_3 = T_3(\mathbf{P}_2)$. Clearly, for $v \in \mathbb{R}^{\mathsf{d}}, v \neq 0$ we have

$$\begin{aligned}\overline{\omega}(v, \mathbf{P}_3) &= \max_{p\in\mathbf{P}_3}\langle v, p\rangle - \min_{p\in\mathbf{P}_3}\langle v, p\rangle = \max_{p\in\mathbf{P}_2}\langle v, T_3 p\rangle - \min_{p\in\mathbf{P}_2}\langle v, T_3 p\rangle = \max_{p\in\mathbf{P}_2}\langle T_3 v, p\rangle - \min_{p\in\mathbf{P}_2}\langle T_3 v, p\rangle \\ &= \overline{\omega}(T_3 v, \mathbf{P}_2).\end{aligned}$$

Similarly, $\overline{\omega}(v, \mathcal{S}_3) = \overline{\omega}(T_3 v, \mathcal{S}_2)$.

By definition, $\mathcal{S}_2$ is a ε-coreset for $\mathbf{P}_2$ iff for any non zero $v \in \mathbb{R}^{\mathsf{d}}$, we have $\overline{\omega}(v, \mathbf{P}_2) \geq (1-\varepsilon)\overline{\omega}(v, \mathcal{S}_2)$. Since T_3 non singular, this implies that for any non-zero v, we have $\overline{\omega}(T_3 v, \mathcal{S}_2) \geq (1-\varepsilon)\overline{\omega}(T_3 v, \mathbf{P}_2)$, which holds iff $\overline{\omega}(v, \mathcal{S}_3) = \overline{\omega}(v, T_3(\mathcal{S}_2)) \geq (1-\varepsilon)\overline{\omega}(v, T_3(\mathbf{P}_2)) = (1-\varepsilon)\overline{\omega}(v, \mathbf{P}_3)$. Thus $\mathcal{S}_3$ is a ε-coreset for $\mathbf{P}_3$. Clearly, the other direction holds by a similar argumentation.

Set $T = T_3 T_2 T_1$, and observe that, by the above argumentation, $\mathcal{S}$ is a ε-coreset for $\mathbf{P}$ if and only if $T(\mathcal{S})$ is a ε-coreset for $T(\mathbf{P})$. However, note that $T(B) = \mathcal{H}$, and $T(\vec{w} + \frac{1}{\mathsf{d}(4\mathsf{d}+1)}B) \subseteq \mathcal{CH}(T(\mathbf{P}))$. Namely, there exists a vector $\vec{w'}$ such that $\vec{w'} + \frac{1}{\mathsf{d}(4\mathsf{d}+1)}\mathcal{H} \subseteq \mathcal{CH}(T(\mathbf{P})) \subseteq \mathcal{H}$. Namely, the point set $T(\mathbf{P})$ is $\alpha = \frac{1}{\mathsf{d}(4\mathsf{d}+1)}$-fat. ■

13.2.2 Computing a smaller coreset

Observation 13.2.2 *If* A *is a* δ*-coreset for directional width of* B*, and* B *is a* ε*-coreset for directional width of* C*, then* A *is a* $(\delta+\varepsilon)$*-coreset of* C*.*

Proof: For any vector v, we have $\overline{\omega}(v, A) \geq (1-\delta)\overline{\omega}(v, B) \geq (1-\delta)(1-\varepsilon)\overline{\omega}(v, C) \geq (1-\delta-\varepsilon)\overline{\omega}(v, C)$. ■

Thus, given a point-set $\mathbf{P}$, we can first extract from it a $\varepsilon/2$-coreset of size $O(1/\varepsilon^{\mathsf{d}-1})$, using Lemma 13.1.2. Let $\mathbf{P}'$ denote the resulting set. We will compute a $\varepsilon/2$-coreset for $\mathbf{P}'$, which would be by the above observation a ε-coreset for directional width of $\mathbf{P}$.

We need the following technical lemma.

Lemma 13.2.3 *Let* $\mathbf{b}$ *be a ball of radius* r *centered at* $(L, 0, \ldots, 0) \in \mathbb{R}^{\mathsf{d}}$*, where* $L \geq 2r$*. Let* p *be an arbitrary point in* $\mathbf{b}$*, and let* $\mathbf{b}'$ *be the largest ball centered at* p *and touching the origin. Then, we have that for* $\mu(p) = \min_{(x_1, x_2, \ldots, x_{\mathsf{d}})\in\mathbf{b}'} x_1$ *we have* $\mu(p) \geq -r^2/L$*.*

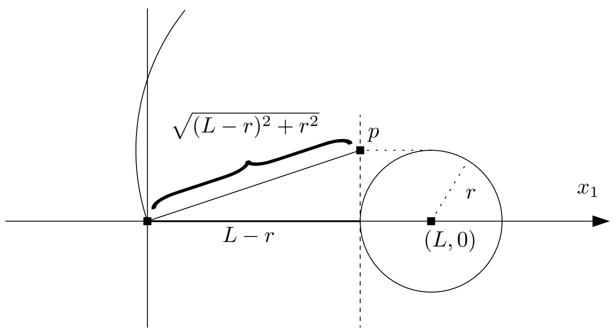


Figure 13.1: Illustration of the proof of Lemma 13.2.3.

Proof: Clearly, if we move p in parallel to the x_1-axis by decreasing the value of x_1, we are decreasing the value of $\mu(p)$. Thus, in the worst case $x_1(p) = L - r$. Similarly, the farther away p is from the x_1-axis the smaller $\mu(p)$ is. Thus, by symmetry, the worst case is when $p = (L - r, r, 0, \ldots, 0)$. See Figure 13.1. The distance between p and the origin is $\sqrt{(L-r)^2 + r^2}$, and

$$\mu(p) = (L-r) - \sqrt{(L-r)^2 + r^2} = \frac{(L-r)^2 - (L-r)^2 - r^2}{(L-r) + \sqrt{(L-r)^2 + r^2}} \geq -\frac{r^2}{2(L-r)} \geq -\frac{r^2}{L},$$

since $L \geq 2r$. ■

Lemma 13.2.4 *Let* $\mathbf{P}'$ *be a set of m points. Then one can compute a* $\varepsilon/2$*-coreset for* $\mathbf{P}'$ *of size* $O(1/\varepsilon^{(d-1)/2})$*, in time* $O(m/\varepsilon^{(d-1)/2})$.

Proof: Note, that byLemma 13.2.1, we can assume that $\mathbf{P}'$ is α-fat for some constant α, and $v + \alpha[-1,1]^{\mathsf{d}} \subseteq \mathcal{CH}(\mathbf{P}') \subseteq [-1,1]^{\mathsf{d}}$, where $v \in \mathbb{R}^d$. In particular, for any direction $u \in \mathbb{S}^{(\mathsf{d}-1)}$, we have $\overline{\omega}(u, \mathbf{P}') \geq 2\alpha$.

Let $\mathbb{S}$ be the sphere of radius $\sqrt{d} + 1$ centered at the origin. Set $\delta = \sqrt{\varepsilon\alpha/4} \leq 1/4$. One can construct a set $\mathcal{I}$ of $O(1/\delta^{\mathsf{d}-1}) = O(1/\varepsilon^{(\mathsf{d}-1)/2})$ points on the sphere $\mathbb{S}$ so that for any point x on $\mathbb{S}$, there exists a point $y \in \mathcal{I}$ such that $\|x - y\| \leq \delta$. We process $\mathbf{P}'$ into a data structure that can answer ε-approximate nearest-neighbor queries. For a query point q, let $\phi(q)$ be the point of $\mathbf{P}'$ returned by this data structure. For each point $y \in \mathcal{I}$, we compute $\phi(y)$ using this data structure. We return the set $\mathbb{S} = \{\phi(y) \mid y \in \mathcal{I}\}$; see Figure 13.2 (ii).

We now show that $\mathbb{S}$ is is an $(\varepsilon/2)$-coreset of $\mathbf{P}'$. For simplicity, we prove the claim under the assumption that $\phi(y)$ is the *exact* nearest-neighbor of y in $\mathbf{P}'$. Fix a direction $u \in \mathbb{S}^{(\mathsf{d}-1)}$. Let $\sigma \in \mathbf{P}'$ be the point that maximizes $\langle u, p\rangle$ over all $p \in \mathbf{P}'$. Suppose the ray emanating from σ in direction u hits $\mathbb{S}$ at a point x. We know that there exists a point $y \in \mathcal{I}$ such that $\|x - y\| \leq \delta$. If $\phi(y) = \sigma$, then $\sigma \in \mathbb{S}$ and

$$\max_{p \in \mathbf{P}'} \langle u, p\rangle - \max_{q \in \mathbb{S}} \langle u, q\rangle = 0.$$

Now suppose $\phi(y) \neq \sigma$. Rotate and translate space, such that σ is at the origin, and u is the positive x_1 axis. Setting $L = \|ox\|$ and $r = \delta$, we have that $\langle u, y\rangle \geq -r^2/L \geq -\delta^2/1 = -\delta^2 = -\varepsilon\alpha/4$, by Lemma 13.2.3. We conclude that $\overline{\omega}(u, \mathbb{S}) \geq \overline{\omega}(u, \mathbf{P}') - 2(\varepsilon\alpha/4) = \overline{\omega}(u, \mathbf{P}') - \varepsilon\alpha/2$. On the other hand, since $\overline{\omega}(u, \mathbf{P}') \geq 2\alpha$, it follows tat $\overline{\omega}(u, \mathbb{S}) \geq (1 - \varepsilon/2)\overline{\omega}(u, \mathbf{P}')$.

As for the running time, we just perform the scan in the most naive way to find $\phi(y)$ for each $y \in \mathcal{I}$. Thus, the running time is as stated. ■

Theorem 13.2.5 *Let* $\mathbf{P}$ *be a set of n points in* $\mathbb{R}^{\mathsf{d}}$*. One can compute a* ε*-coreset for directional width of* $\mathbf{P}$ *in* $O(n + 1/\varepsilon^{3(d-1)/2})$ *time. The coreset size is* $O(\varepsilon^{(d-1)/2})$.

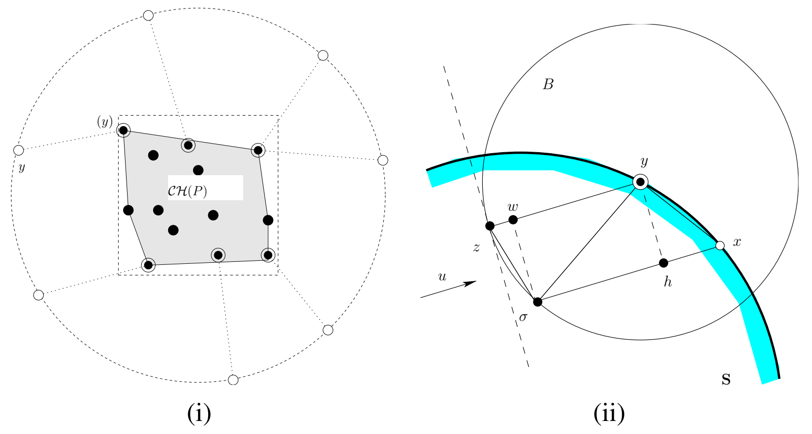


Figure 13.2: (i) An improved algorithm. (ii) Correctness of the improved algorithm.

Proof: We use the algorithm of Lemma 13.2.1 and Lemma 13.1.2 on the resulting set. This computes a $\varepsilon/2$-coreset $\mathbf{P}'$ of $\mathbf{P}$ of size $O(1/\varepsilon^{\mathsf{d}-1})$. Next, we apply Lemma 13.2.4 and compute a $\varepsilon/2$-coreset $\mathcal{S}$ of $\mathbf{P}'$. This is a ε-coreset of $\mathbf{P}$. ■

13.3 Exercises

Exercise 13.3.1 [5 Points]

Prove that in the worst case, a ε-coreset for directional width has to be of size $\Omega(\varepsilon^{-(d-1)/2})$.

13.4 Bibliographical notes

Section 13.1 and Section 13.2.1 is from [AHV04]. The result of Section 13.2.2 was observed independently by Chan [Cha04] and Yu *et al.* [YAPV04]. It is a simplification of an algorithm of Agarwal *et al.* [AHV04] which in turn is an adaptation of a method of Dudley [Dud74].

The running time of Theorem 13.2.5 can be improved to $O(n+1/\varepsilon^{\mathsf{d}-1})$ by using special nearest neighbor algorithm on a grid. Trying to use some of the other ANN data-strictures will not work since it would not improve the running time over the naive algorithm. The interested reader, can see the paper by Chan [Cha04].

Chapter 14

Approximating the Extent of Lines, Hyperplanes and Moving Points

Once I sat on the steps by a gate of David's Tower, I placed my two heavy baskets at my side. A group of tourists was standing around their guide and I became their target marker. "You see that man with the baskets? Just right of his head there's an arch from the Roman period. Just right of his head."

"But he's moving, he's moving!"

I said to myself: redemption will come only if their guide tells them, "You see that arch from the Roman period? It's not important: but next to it, left and down a bit, there sits a man who's bought fruit and vegetables for his family."

– –Yehuda Amichai, Tourists

14.1 Preliminaries

Definition 14.1.1 Given a set of hyperplanes $\mathcal{H}$ in $\mathbb{R}^{\mathsf{d}}$, the minimization diagram of $\mathcal{H}$, known as the *lower envelope* of $\mathcal{H}$, is the function $\mathcal{L}_{\mathcal{H}} : \mathbb{R}^{\mathsf{d}-1} \to \mathbb{R}$, where we have $\mathcal{L}(\mathbf{x}) = \min_{h \in \mathcal{H}} h(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^{\mathsf{d}-1}$.

Similarly, the *upper envelope* of $\mathcal{H}$ is the function $\mathcal{U}(\mathbf{x}) = \max_{h \in \mathcal{H}} h(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^{\mathsf{d}-1}$.

The *extent* of $\mathcal{H}$ and $\mathbf{x} \in \mathbb{R}^{\mathsf{d}-1}$ is the vertical distance between the upper and lower envelope at $\mathbf{x}$; namely, $\mathcal{E}_{\mathcal{H}}(\mathbf{x}) = \mathcal{U}(\mathbf{x}) - \mathcal{L}(\mathbf{x})$.

14.2 Motivation - Maintaining the Bounding Box of Moving Points

Let $P = \{p_1, \ldots, p_n\}$ be a set of n points moving in $\mathbb{R}^{\mathsf{d}}$. For a given time t, let $p_i(t) = (x_i^1(t), \ldots, x_i^{\mathsf{d}}(t))$ denote the position of p_i at time t. We will use $P(t)$ denote the set P at time t. We say that the motion of P has *degree* k if every $x_i^j(t)$ is a polynomial of degree at most k. We call a motion of degree 1 *linear*. Namely, $p_i(t) = Ↄ_i + {}_i t$, where $Ↄ_i, {}_i \in \mathbb{R}^{\mathsf{d}}$. The values $Ↄ_i, {}_i$ are fixed.

Our purpose is to develop efficient approaches for maintaining various descriptors of the extent of P, including the smallest enclosing orthogonal rectangle of P. This measure indicates how spread out the point set P is. As the points move continuously, the extent measure of interest changes continuously as well, though its combinatorial realization changes only at certain discrete times. For example, the smallest orthogonal rectangle containing P can be represented by a sequence of $2d$ points, each lying on one of the facets of the rectangle. As the points move, the rectangle also changes continuously. At certain discrete times, the points lying on the boundary of the rectangle change, and we have to update the sequence of points defining the rectangle. Similarly, Our approach is to focus on these discrete changes (or *events*) and track through time the combinatorial description of the extent measure of interest.

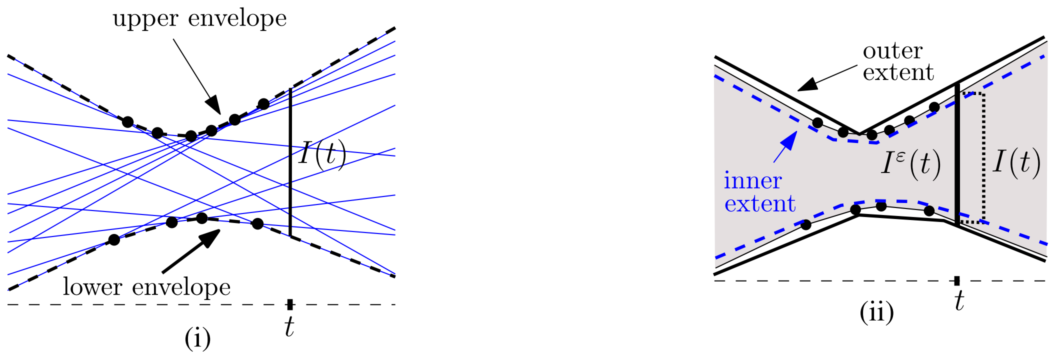


Figure 14.1: (i) The extent of the moving points, is no more than the vertical segment connecting the lower envelope to the upper envelope. The black dots mark where the movement description of $I(t)$ changes. (ii) The approximate extent.

Since we are computing the axis parallel bounding box of the moving points, we can solve the problem in each dimension separately. Thus, consider the points as points moving linearly in one dimension. The *extent* $B(t)$ of $P(t)$ is the smallest interval containing $P(t)$.

It will be convenient to work in a parametric xt-plane in which a moving point $p(t) \in \mathbb{R}$ at time t is mapped to the point $(t, p(t))$. For $1 \leq i \leq n$, we map the point $p_i \in P$ to the line $\ell_i = \bigcup_t (t, p_i(t))$, for $i = 1, \ldots, n$. Let $L = \{\ell_1, \ldots, \ell_n\}$ be the resulting set of lines, and let $\mathcal{A}(L)$ be their arrangement. Clearly, the extent $B(t_0)$ of $P(t_0)$ is the vertical interval $I(t_0)$ in the arrangement $\mathcal{A}(L)$ connecting the upper and lower envelopes of L at $t = t_0$. See Figure 14.1(i). The combinatorial structure of $I(t)$ changes at the vertices of the two envelopes of L, and all the different combinatorial structures of $I(t)$ can be computed in $O(n \log n)$ time by computing the upper and lower envelopes of L.

We want to maintain a vertical interval $I^+_\varepsilon(t)$ so that $I(t) \subseteq I^+_\varepsilon(t)$ and $\left|I^+_\varepsilon(t)\right| \leq (1 + \varepsilon)\left|I(t)\right|$ for all t, so that the endpoints of $I^+_\varepsilon(t)$ follow piecewise-linear trajectories, and so that the number of combinatorial changes in $I^\varepsilon(t)$ is small. Alternatively, we want to maintain a vertical interval $I^-_\varepsilon(t) \subseteq I(t)$ such that $\left|I^-_\varepsilon(t)\right| \geq (1 - \varepsilon)\left|I(t)\right|$. Clearly, having one approximation would imply the other by appropriate rescaling.

Geometrically, this has the following interpretation: We want to simplify the upper and lower envelopes of $\mathcal{A}(L)$ by convex and concave polygonal chains, respectively, so that the simplified upper (resp. lower) envelope lies above (resp. below) the original upper (resp. lower) envelope and so that for any t, the vertical segment connecting the simplified envelopes is contained in $(1 + \varepsilon)I(t)$. See Figure 14.1 (ii).

14.3 Duality and Extent

A hyperplane $h : x_d = b_1 x_1 + \cdots + b_{d-1} x_{d-1} - b_d$ in $\mathbb{R}^d$ can be interpreted as a function from $\mathbb{R}^{d-1}$ to $\mathbb{R}$. Given a point $(y_1, \ldots, y_d)$ let $h(y) = b_1 y_1 + \cdots + b_{d-1} y_{d-1} - b_d$. In particular, a point y lies *above* the hyperplane h if $y_d > h(y)$. Similarly, y is *below* the hyperplane h if $y_d < h(y)$. Finally, a point is on the hyperplane if $h(y) = y_d$.

The *dual* of a point $b = (b_1, \ldots, b_d) \in \mathbb{R}^d$ is a hyperplane $b^* : x_d = b_1 x_1 + \cdots b_{d-1} x_{d-1} - b_d$, and the *dual* of a hyperplane $h : x_d = a_1 x_1 + a_2 x_2 + \cdots + a_{d-1} x_{d-1} + a_d$ is the point $h^* = (a_1, \ldots, a_{d-1}, -a_d)$. There are several alternative definitions of duality, but they are essentially similar. Summarizing:

$$\begin{aligned} b = (b_1, \ldots, b_d) \quad &\Rightarrow \quad b^* : x_d = b_1 x_1 + \cdots b_{d-1} x_{d-1} - b_d \\ h : x_d = a_1 x_1 + a_2 x_2 + \cdots + a_{d-1} x_{d-1} + a_d \quad &\Rightarrow \quad h^* = (a_1, \ldots, a_{d-1}, -a_d). \end{aligned}$$

The proof of the following lemma is straightforward, and is delegated to Exercise 17.3.1.

Lemma 14.3.1 *For a point $b = (b_1, \ldots, b_d)$, we have:*

(i) $b^{**} = b$.

(ii) *The point b lies above (resp. below, on) the hyperplane h, if and only if the point h^* lies above (resp. below, on) the hyperplane b^*.*

(iii) *The vertical distance between b and h is the same as that between b^* and h^*.*

(iv) *The vertical distance $\delta(h, g)$ between two parallel hyperplanes h and g is the same as the length of the vertical segment h^*g^*.*

(v) *For $\mathbf{x} \in \mathbb{R}^{\mathsf{d}-1}$, the hyperplane h dual to the point $\mathcal{L}_{\mathcal{H}}(\mathsf{p})$ (resp. $\mathcal{U}_{\mathcal{H}}(\mathsf{p})$) is normal to the vector $(\mathsf{p}, -1) \in \mathbb{R}^{\mathsf{d}}$ and supports $\mathcal{CH}(\mathcal{H}^*)$. Furthermore, $\mathcal{H}^*$ lies below (resp. above) the hyperplane h. Furthermore, $\mathcal{E}_{\mathcal{H}}(\mathbf{x})$ is the vertical distance between these two parallel supporting planes.*

Definition 14.3.2 For a set of hyperplanes $\mathcal{H}$, a subset $\mathcal{S} \subset \mathcal{H}$ is a *ε-coreset* of $\mathcal{H}$ for the extent measure, if for any $\mathbf{x} \in \mathbb{R}^{\mathsf{d}-1}$ we have $\mathcal{E}_{\mathcal{S}} \geq (1 - \varepsilon)\mathcal{E}_{\mathcal{H}}$.

Similarly, for a point-set $\mathbf{P} \subseteq \mathbb{R}^{\mathsf{d}}$, a set $\mathcal{S} \subseteq \mathbf{P}$ is a *ε-coreset for vertical extent* of $\mathbf{P}$, if, for any direction $v \in \mathbb{S}^{(\mathsf{d}-1)}$, we have that $\mu_v(\mathcal{S}) \geq (1-\varepsilon)\mu_v(\mathbf{P})$, where $\mu_v(\mathbf{P})$ is the vertical distance between the two supporting hyperplanes of $\mathbf{P}$ which are perpendicular to v.

Thus, to compute a coreset for a set of hyperplanes, it is by duality and Lemma 17.2.1 enough to find a coreset for the vertical extent of a point-set.

Lemma 14.3.3 *The set $\mathcal{S}$ is a ε-coreset of the point set $\mathbf{P} \subseteq \mathbb{R}^{\mathsf{d}}$ for vertical extent if and only if $\mathcal{S}$ is a ε-coreset for directional width.*

Proof: Consider any direction $v \in \mathbb{S}^{(\mathsf{d}-1)}$, and let α be its (smaller) angle with with the x_{d} axis. Clearly, $\overline{\omega}(v, \mathcal{S}) = \mu_v(\mathcal{S})\cos\alpha$ and $\overline{\omega}(v, \mathbf{P}) = \mu_v(PntSet)\cos\alpha$. Thus, if $\overline{\omega}(v, \mathcal{S}) \geq (1 - \varepsilon)\overline{\omega}(v, \mathbf{P})$ then $\mu_v(\mathcal{S}) \geq (1 - \varepsilon)\mu_v(\mathbf{P})$, and vice versa. ■

Theorem 14.3.4 *Let $\mathcal{H}$ be a set of n hyperplanes in $\mathbb{R}^{\mathsf{d}}$. One can compute a ε-coreset of $\mathcal{H}$ of, size $O(1/\varepsilon^{\mathsf{d}-1})$, in $O(n + \min(n, 1/\varepsilon^{\mathsf{d}-1}))$ time. Alternatively, one can compute a ε-coreset of size $O(1/\varepsilon^{(\mathsf{d}-1)/2})$, in time $O(n + 1/\varepsilon^{3(\mathsf{d}-1)/2})$.*

Proof: By Lemma 14.3.3, the coreset computation is equivalent to computing coreset for directional width. However, this can be done in the stated bounds, by Theorem 13.1.3 and Theorem 13.2.5. ■

Going back to our motivation, we have the following result:

Lemma 14.3.5 *Let $P(t)$ be a set of n points with linear motion in $\mathbb{R}^{\mathsf{d}}$. We can compute an axis parallel moving bounding box $b(t)$ for $P(t)$ that changes $O(\mathsf{d}/\sqrt{\varepsilon})$ times (in other times, the bounding box moves with linear motion). The time to compute this bounding box is $O(\mathsf{d}(n + 1/\varepsilon^{3/2}))$.*

Furthermore, we have that $\mathrm{Box}(P(t)) \subseteq b(t) \subseteq (1 + \varepsilon)\mathrm{Box}(P(t))$*, where* $\mathrm{Box}(t)$ *is the minimum axis parallel bounding box of P.*

Proof: We compute the solution for each dimension separately. In each dimension, we compute a coreset of the resulting set of lines in two dimensions, and compute the upper and lower envelope of the coreset. Finally, we expand the upper and lower envelopes appropriately so that the include the original upper and lower envelopes. The bounds on the running time follows from Theorem 14.3.4. ■

14.4 Coresets

At this point, our discussion exposes a very powerful technique for approximate geometric algorithms: (i) extract small subset that represents that data well (i.e., coreset), and (ii) run some other algorithm on the coreset. To this end, we need a more unified definition of coresets.

Definition 14.4.1 (Coresets) Given a set $\mathbf{P}$ of points (or geometric objects) in $\mathbb{R}^{\mathsf{d}}$, and an objective function $f : 2^{\mathbb{R}^{\mathsf{d}}} \to \mathbb{R}$ (say, $f(\mathbf{P})$ is the width of $\mathbf{P}$), a ε-*coreset* is a subset $\mathcal{S}$ of the points of $\mathbf{P}$ such that

$$f(\mathcal{S}) \geq (1 - \varepsilon) f(\mathbf{P}).$$

We will state this fact, by saying that $\mathcal{S}$ is a ε-coreset of P for $f(\cdot)$.

If the function $f(\cdot)$ is parameterized, namely $f(Q, v)$, then $\mathcal{S} \subseteq \mathbf{P}$ is a coreset if

$$\forall v \quad f(\mathcal{S}, v) \geq (1 - \varepsilon) f(\mathbf{P}, v).$$

As a concrete example, for v a unit vector, consider the function $\overline{\omega}(v, \mathbf{P})$ which is the directional width of $\mathbf{P}$; namely, it is the length of the projection of $\mathcal{CH}(\mathbf{P})$ into the direction of v.

Coresets are of interest when they can be computed quickly, and have small size, hopefully of size independent of n, the size of the input set P. Interestingly, our current techniques are almost sufficient to show the existence of coresets for a large family of problems.

14.5 Extent of Polynomials

Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a family of d-variate polynomials and let $u_1, \ldots, u_{\mathsf{d}}$ be the variables over which the functions of $\mathcal{F}$ are defined. Each f_i corresponds to a surface in $\mathbb{R}^{\mathsf{d}+1}$: For example, any d-variate linear function can be considered as a hyperplane in $\mathbb{R}^{\mathsf{d}+1}$ (and vice versa). The upper/lower envelopes and extent of $\mathcal{F}$ can now be defined similar to the hyperplane case.

Each monomial over $u_1, \ldots, u_{\mathsf{d}}$ appearing in $\mathcal{F}$ can be mapped to a distinct variable x_i. Let $x_1, \ldots, x_s$ be the resulting variables. As such $\mathcal{F}$ can be linearized into a set $\mathcal{H} = \{h_1, \ldots, h_n\}$ of linear functions over $\mathbb{R}^s$. In particular, $\mathcal{H}$ is a set of n hyperplanes in $\mathbb{R}^{s+1}$. Note that the surface induced by f_i in $\mathbb{R}^{\mathsf{d}+1}$ corresponds only to a subset of the surface of h_i in $\mathbb{R}^{s+1}$. This technique is called *linearization*.

For example, consider a family of polynomials $\mathcal{F} = \{f_1, \ldots, f_n\}$, where $f_i(x, y) = a_i(x^2+y^2)+b_i x+c_i y+d_i$, and $a_i, b_i, c_i, d_i \in \mathbb{R}$, for $i = 1, \ldots, n$. This family of polynomials defined over $\mathbb{R}^2$, can be linearized to a family of linear functions defined over $\mathbb{R}^3$, by $h_i(x, y, z) = a_i z + b_i x + c_i y + d_i$, and setting $\mathcal{H} = \{h_1, \ldots, h_n\}$. Clearly, $\mathcal{H}$ is a set of hyperplanes in $\mathbb{R}^4$, and $f_i(x, y) = h_i(x, y, x^2 + y^2)$. Thus, for any point $(x, y) \in \mathbb{R}^2$, instead of evaluating $\mathcal{F}$ on (x, y), we can evaluate $\mathcal{H}$ on $\eta(x, y) = (x, y, x^2 + y^2)$, where $\eta(x, y)$ is the *linearization image* of (x, y). The advantage of this linearization is that $\mathcal{H}$, being a family of linear functions, is now easier to handle than $\mathcal{F}$.

Observe, that $X = \eta(\mathbb{R}^2)$ is a *subset* of $\mathbb{R}^3$ (this is the “standard” paraboloid), and we are interested in the value of $\mathcal{H}$ only on points belonging to X. In particular, the set X is not necessarily convex. The set X resulting from the linearization is a semi-algebraic set of constant complexity, and as such basic manipulation operations of X can be performed in constant time.

Note that for each $1 \leq i \leq n$, $f_i(p) = h_i(\eta(p))$ for $p \in \mathbb{R}^d$. As such, if $\mathcal{H}' \subseteq \mathcal{H}$ is a ε-coreset of $\mathcal{H}$ for the extent, then clearly the corresponding subset in $\mathcal{F}$ is a ε-coreset of $\mathcal{F}$ for the extent measure. The following theorem is a restatement of Theorem 14.3.4 in this settings.

Theorem 14.5.1 *Given a family of* d*-variate polynomials* $\mathcal{F} = \{f_1, \ldots, f_n\}$*, and parameter* ε*, one can compute, in* $O(n + 1/\varepsilon^s)$ *time, a subset* $\mathcal{F}' \subseteq \mathcal{F}$ *of* $O(1/\varepsilon^s)$ *polynomials, such that* $\mathcal{F}'$ *is a* ε*-coreset of* $\mathcal{F}$ *for the extent measure. Here* s *is the number of different monomials present in the polynomials of* $\mathcal{F}$*.*

Alternatively, one can compute a ε*-coreset, of size* $O(1/\varepsilon^{s/2})$*, in tiem* $O(n + 1/\varepsilon^{3s/2})$*.*

14.6 Roots of Polynomials

We now consider the problem of approximating the extent a family of square-roots of polynomials. Note, that this is considerably harder than handling polynomials because square-roots of polynomials can not be directly linearized. It turns out, however, that it is enough to $O(\varepsilon^2)$-approximate the extent of the functions inside the roots, and take the root of the resulting approximation.

Theorem 14.6.1 *Let $\mathcal{F} = \left\{(f_1)^{1/2}, \ldots, (f_n)^{1/2}\right\}$ be a family of k-variate functions (over $\mathsf{p} = (x_1, \ldots, x_k) \in \mathbb{R}^k$), where each f_i is a polynomial that is non-negative for every $\mathsf{p} \in \mathbb{R}^k$. Given any $\varepsilon > 0$, we can compute, in $O(n + 1/\varepsilon^{2k'})$ time, a ε-coreset $\mathcal{G} \subseteq \mathcal{F}$ of size $O(1/\varepsilon^{2k'})$, for the measure of the extent. Here k' is the number of different monomials present in the polynomials in $f_1, \ldots, f_n$.*

Alternatively, one can compute a set $\mathcal{G}' \subseteq \mathcal{F}$, in $O(n + 1/\varepsilon^{3k'})$ time, that ε-approximates $\mathcal{F}$, so that $|\mathcal{G}'| = O(1/\varepsilon^{k'})$.

Proof: Let $\mathcal{F}^2$ denote the family $\{f_1, \ldots, f_n\}$. Using the algorithm of Theorem 14.5.1, we compute a δ'-coreset $\mathcal{G}^2 \subseteq \mathcal{F}^2$ of $\mathcal{F}^2$, where $\delta' = \varepsilon^2/64$. Let $\mathcal{G} \subseteq \mathcal{F}$ denote the family $\left\{(f_i)^{1/2} | f_i \in \mathcal{G}^2\right\}$.

Consider any point $\mathbf{x} \in \mathbb{R}^k$. We have that $\mathcal{E}_{\mathcal{G}^2}(\mathbf{x}) \geq (1-\delta')\mathcal{E}_{\mathcal{F}^2}(\mathbf{x})$, and let $a = \mathcal{L}_{\mathcal{F}^2}(\mathbf{x})$, $A = \mathcal{L}_{\mathcal{G}^2}(\mathbf{x})$, $B = \mathcal{U}_{\mathcal{G}^2}(\mathbf{x})$, and $b = \mathcal{U}_{\mathcal{F}^2}(\mathbf{x})$. Clearly, we have $0 \leq a \leq A \leq B \leq b$ and $B - A \geq (1-\delta')(b-a)$. Since $(1+2\delta')(1-\delta') \geq 0$, we have that $(1+2\delta')(B-A) \geq b-a$.

By Lemma 14.6.2 below, we have that Then, $\sqrt{A} - \sqrt{a} \leq (\varepsilon/2)U$, and $\sqrt{b} - \sqrt{B} \leq (\varepsilon/2)U$, where $U = \sqrt{B} - \sqrt{A}$. Namely, $\sqrt{B} - \sqrt{A} \geq (1-\varepsilon)(\sqrt{b} - \sqrt{a})$. Namely, $\mathcal{G}$ is a ε-coreset for the extent of $\mathcal{F}$.

The bounds on the size of $\mathcal{G}$ and the running time are easily verified. ■

Lemma 14.6.2 *Let $0 \leq a \leq A \leq B \leq b$, and $0 < \varepsilon \leq 1$ be given parameters, so that $b - a \leq (1+\delta)(B-A)$, where $\delta = \varepsilon^2/16$. Then, $\sqrt{A} - \sqrt{a} \leq (\varepsilon/2)U$, and $\sqrt{b} - \sqrt{B} \leq (\varepsilon/2)U$, where $U = \sqrt{B} - \sqrt{A}$.*

Proof: Clearly,

$$\sqrt{A} + \sqrt{B} \quad \leq \quad \sqrt{a} + \sqrt{A-a} + \sqrt{b} \leq \sqrt{a} + \sqrt{\delta b} + \sqrt{b} \leq (1+\sqrt{\delta})(\sqrt{a} + \sqrt{b}).$$

Namely, $\frac{\sqrt{A}+\sqrt{B}}{1+\sqrt{\delta}} \leq \sqrt{a} + \sqrt{b}$. On the other hand,

$$\begin{aligned}\sqrt{b} - \sqrt{a} &= \frac{b-a}{\sqrt{b}+\sqrt{a}} \leq \frac{(1+\delta)(B-A)}{\sqrt{b}+\sqrt{a}} \leq (1+\delta)(1+\sqrt{\delta})\frac{B-A}{\sqrt{B}+\sqrt{A}} \\ &= `(1+\varepsilon^2/16)(1+\varepsilon/4)(\sqrt{B} - \sqrt{A}) \leq (1+\varepsilon/2)(\sqrt{B} - \sqrt{A}).\end{aligned}$$

■

14.7 Applications

14.7.1 Minimum Width Annulus

Let $\mathbf{P} = \{p_1, \ldots, p_n\}$ be a set of n points in the plane. Let $f_i(q)$ denote the distance of the ith point from the point q. It is easy to verify that $f_i(q) = \sqrt{\left(x_q - x_{p_i}\right)^2 + \left(y_q - y_{p_i}\right)^2}$. Let $\mathcal{F} = \{f_1, \ldots, f_n\}$. It is easy to verify that for a center point $\mathbf{x} \in \mathbb{R}^2$, the width of the minimum width annulus containing P which is centered at $\mathbf{x}$ has width $\mathcal{E}_{\mathcal{F}}(\mathbf{x})$. Thus, we would like to compute a ε-coreset for $\mathcal{F}$.

Consider the set of functions $\mathcal{F}^2$. Clearly, $f_i^2(x,y) = \left(x - x_{p_i}\right)^2 + \left(y - y_{p_i}\right)^2 = x^2 - 2x_{p_i}x + x_{p_i}^2 + y^2 - 2y_{p_i}y + y_{p_i}^2$. Clearly, all the functions of $\mathcal{F}^2$ have this (additive) common factor of $x^2 + y^2$. Since we only care about the vertical extent, we have $\mathcal{H} = \left\{-2x_{p_i}x + x_{p_i}^2 - 2y_{p_i}y + y_{p_i}^2 \;\middle|\; i = 1, \ldots, n\right\}$ has the same extent as $\mathcal{F}^2$; formally, for any $\mathbf{x} \in \mathbb{R}^2$, we have $\mathcal{E}_{\mathcal{F}^2}(\mathbf{x}) = \mathcal{E}_{\mathcal{H}}(\mathbf{x})$. Now, $\mathcal{H}$ is just a family of hyperplanes in $\mathbb{R}^3$, and it has a $\varepsilon^2/64$-coreset $\mathcal{S}_{\mathcal{H}}$ for the extent of size $1/\varepsilon$ which can be computed in $O(n + 1/\varepsilon^3)$ time. This corresponds to a $\varepsilon^2/64$-coreset $\mathcal{S}_{\mathcal{F}^2}$ of $\mathcal{F}^2$. By Theorem 14.6.1, this corresponds to a ε-coreset $\mathcal{S}_{\mathcal{F}}$ of $\mathcal{F}$. Finally, this corresponds to coreset $\mathcal{S} \subseteq \mathbf{P}$ of size $O(1/\varepsilon)$, such that the minimum width annulus of $\mathcal{S}$, if we expand it by $(1 + 2\varepsilon)$, it contains all the points of $\mathbf{P}$. Thus, we can just find the minimum width annulus of $\mathcal{S}$. This can be done in $O(1/\varepsilon^2)$ time using an exact algorithm. Putting everything together, we get:

Theorem 14.7.1 *Let* $\mathbf{P}$ *be a set of n points in the plane, and let* $0 \le \varepsilon \le 1$ *be a parameter. One can compute a* $(1 + \varepsilon)$*-approximate minimum width annulus to* $\mathbf{P}$ *in* $O(n + 1/\varepsilon^3)$ *time.*

14.8 Exercises

14.9 Bibliographical notes

Linearization was widely used in fields such as machine learning [CS00] and computational geometry [AM94].

There is a general technique for finding the best possible linearization (i.e., a mapping η with the target dimension as small as possible), see [AM94] for details.

Chapter 15

15 - Approximating the Extent of Lines, Hyperplanes and Moving Points II

Drug misuse is not a disease, it is a decision, like the decision to step out in front of a moving car. You would call that not a disease but an error in judgment. When a bunch of people begin to do it, it is a social error, a life-style. In this particular life-style the motto is "be happy now because tomorrow you are dying," but the dying begins almost at once, and the happiness is a memory. ... If there was any "sin," it was that these people wanted to keep on having a good time forever, and were punished for that, but, as I say, I feel that, if so, the punishment was far too great, and I prefer to think of it only in a Greek or morally neutral way, as mere science, as deterministic impartial cause-and-effect.

– A Scanner Darkly, Philip K. Dick

15.1 More Coresets

15.1.1 Maintaining certain measures of moving points

We show that our techniques can be extended to handle other measure of moving points (width, diameter, etc). Let $\mathbf{P} = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^{\mathsf{d}}$, each moving independently. Let $p_i(t) = (p_{i1}(t), \ldots, p_{id}(t))$ denote the position of point p_i at time t. Set $\mathbf{P}(t) = \{p_i(t) \mid 1 \le i \le n\}$. If each p_{ij} is a polynomial of degree at most r, we say that the motion of $\mathbf{P}$ has *degree r*. We call the motion of $\mathbf{P}$ *linear* if $r = 1$ and *algebraic* if r is bounded by a constant.

Given a parameter $\varepsilon > 0$, we call a subset $Q \subseteq P$ an ε-coreset of $\mathbf{P}$ for directional width if for any direction $u \in \mathbb{S}^{(\mathsf{d}-1)}$, we have

$$(1-\varepsilon)\overline{\omega}(u, P(t)) \le \overline{\omega}(u, Q(t)) \qquad \text{for all } t \in \mathbb{R}.$$

15.1.1.1 Computing an ε-coreset for directional width.

First let us assume that the motion of $\mathbf{P}$ is linear, i.e., $p_i(t) = a_i + b_i t$, for $1 \le i \le n$, where $a_i, b_i \in \mathbb{R}^{\mathsf{d}}$. For a direction $u = (u_1, \ldots, u_{\mathsf{d}}) \in \mathbb{S}^{(\mathsf{d}-1)}$, we define a $(\mathsf{d}+1)$-variate polynomial

$$f_i(u,t) \quad = \quad \langle p_i(t), u\rangle = \langle a_i + b_i t, u\rangle = \sum_{j=1}^{\mathsf{d}} a_{ij} u_j + \sum_{j=1}^{\mathsf{d}} b_{ij} \cdot (t u_j).$$

Set $\mathcal{F} = \{f_1, \ldots, f_n\}$. Then

$$\overline{\omega}(u, P(t)) = \max_i \langle p_i(t), u\rangle - \min_i \langle p_i(t), u\rangle = \max_i f_i(u,t) - \min_i f_i(u,t) = \mathcal{E}_{\mathcal{F}}(u,t).$$

Since $\mathcal{F}$ is a family of $(\mathsf{d}+1)$-variate polynomials, which admits a linearization of dimension $2\mathsf{d}$ (there are $2\mathsf{d}$ monomials), using Theorem 14.5.1, we conclude the following.

Theorem 15.1.1 *Given a set* **P** *of n points in $\mathbb{R}^{\mathsf{d}}$, each moving linearly, and a parameter $\varepsilon > 0$, we can compute an ε-coreset of* **P** *for directional width of size $O(1/\varepsilon^{2\mathsf{d}})$, in $O(n + 1/\varepsilon^{2\mathsf{d}})$ time, or an ε-coreset of size $O(1/\varepsilon^{\mathsf{d}})$ in $O(n + 1/\varepsilon^{3(\mathsf{d})})$ time.*

If the degree of motion of **P** is $r > 1$, we can write the d-variate polynomial $f_i(u,t)$ as:

$$f_i(u,t) \quad = \quad \langle p_i(t), u\rangle = \left\langle \sum_{j=0}^{r} a_{ij}t^j, u \right\rangle = \sum_{j=0}^{r} \left\langle a_{ij}t^j, u \right\rangle$$

where $a_{ij} \in \mathbb{R}^{\mathsf{d}}$. A straightforward extension of the above argument shows that f_i's admit a linearization of dimension $(r+1)\mathsf{d}$. Using Theorem 14.5.1, we obtain the following.

Theorem 15.1.2 *Given a set* **P** *of n moving points in $\mathbb{R}^{\mathsf{d}}$ whose motion has degree $r > 1$ and a parameter $\varepsilon > 0$, we can compute an ε-coreset for directional width of* **P** *of size $O(1/\varepsilon^{(r+1)\mathsf{d}})$ in $O(n + 1/\varepsilon^{(r+1)\mathsf{d}})$ time, or of size $O(1/\varepsilon^{(r+1)\mathsf{d}}/2)$ in $O(n + 1/\varepsilon^{3(r+1)\mathsf{d}/2})$ time.*

15.1.2 Minimum-width cylindrical shell

Let $\mathbf{P} = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$. Let $w^* = w^*(P)$ denote the width of the thinnest cylindrical shell, the region lying between two co-axial cylinders, containing **P**. Let $d(\ell, p)$ denote the distance between a point $p \in \mathbb{R}^d$ and a line $\ell \subset \mathbb{R}^d$. If we fix a line ℓ, then the width of the thinnest cylindrical shell with axis ℓ and containing **P** is $w(\ell, P) = \max_{p\in P} d(\ell, p) - \min_{p\in P} d(\ell, p)$. A line $\ell \in \mathbb{R}^d$ not parallel to the hyperplane $x_d = 0$ can be represented by a $(2d-2)$-tuple $(x_1, \ldots, x_{2d-2}) \in \mathbb{R}^{2d-2}$:

$$\ell = \{p + tq \mid t \in \mathbb{R}\},$$

where $p = (x_1, \ldots, x_{d-1}, 0)$ is the intersection point of ℓ with the hyperplane $x_d = 0$ and $q = (x_d, \ldots, x_{2d-2}, 1)$ is the orientation of ℓ (i.e., q is the intersection point of the hyperplane $x_d = 1$ with the line parallel to ℓ and passing through the origin). (The lines parallel to the hyperplane $x_d = 0$ can be handled separately by a simpler algorithm, so let us assume this case does not happend.) The distance between ℓ and a point p is the same as the distance of the line $\ell' = \{(p - \mathsf{p}) + tq \mid t \in \mathbb{R}\}$ from the origin; see Figure 15.1. The point y on ℓ' closest to the origin satisfies $y = (p - \mathsf{p}) + tq$ for some t, and at the same time $\langle y, q\rangle = \langle (p-\mathsf{p}) + tq, q\rangle = 0$, which implies that $t = -\langle (p-\mathsf{p}), q\rangle / \|q\|^2$. Thus,

$$d(\ell, \mathsf{p}) = \|y\| = \|(p-\mathsf{p}) + tq\| = \left\| (p-\mathsf{p}) - \frac{[\langle p-\mathsf{p}, q\rangle]q}{\|q\|^2} \right\|,$$

Define $f_i(\ell) = f_i(p,q) = d(\ell, p_i)$, and set $\mathcal{F} = \{f_i \mid p_i \in P\}$. Then $w^* = \min_{x\in\mathbb{R}^{2d-2}} \mathcal{E}_{\mathcal{F}}(x)$. (We assume for simplicity that the axis of the optimal shell is not parallel to the hyperplane $x_d = 0$.) Let $f'_i(p,q) = \|q\|^2 \cdot f_i(p,q) = \left\| \|q\|^2 (p - p_i) - \langle p - p_i, q\rangle q \right\|$, and set $\mathcal{F}' = \left\{ f'_1, \ldots, f'_n \right\}$.

Define $g_i(p,q) = \left(f'_i(p,q)\right)^2$, and let $\mathcal{G} = \{g_1 \ldots, g_n\}$. Then g_i is a $(2d-2)$-variate polynomial and has $O(d^2)$ monomials. Therefore $\mathcal{G}$ admits a linearization of dimension $O(d^2)$. By Theorem 14.5.1, we compute a $O(\varepsilon^2)$-coreset of $\mathcal{G}$ of size $O\left(1/\varepsilon^{d^2}\right)$ in $O\left(n + 1/\varepsilon^{O(d^2)}\right)$. This in turn correspons to a ε-coreset of $\mathcal{F}'$, by Theorem 14.6.1. It is now easy to verfiy that this corresponds to a ε-coreset (for the extent) for $\mathcal{F}$. Finally, this corresponds to a subset $\mathcal{S} \subseteq \mathbf{P}$, such that $\mathcal{S}$ is a coreset of **P** for $w(\ell, \mathbf{P})$. Formally, a subset $\mathcal{S} \subseteq \mathbf{P}$ is a *ε-coreset for cyilindrical shell width* , if

$$w(\ell, \mathcal{S}) \geq (1-\varepsilon) w(\ell, \mathbf{P}), \quad \text{for all } v \in \mathbb{S}^{(\mathsf{d}-1)}.$$

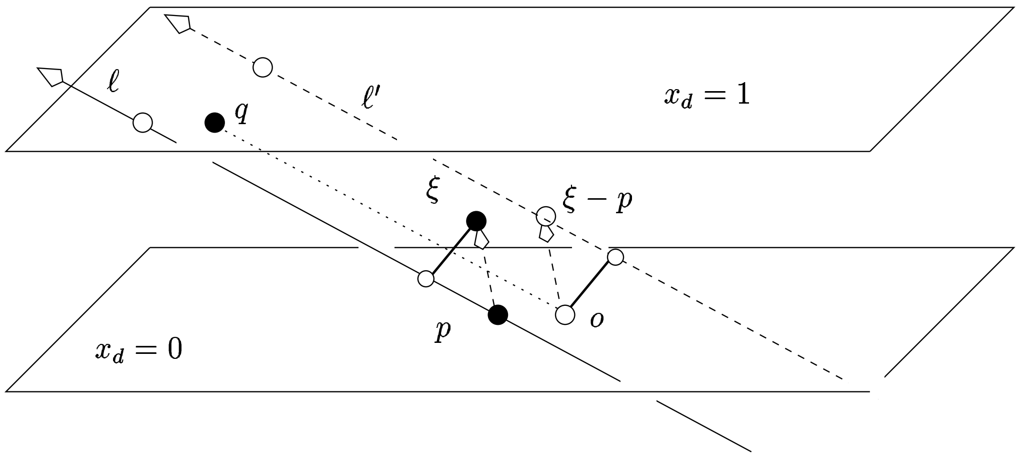


Figure 15.1: Parametrization of a line ℓ in $\mathbb{R}^3$ and its distance from a point p; the small hollow circle on ℓ is the point closest to p.

Thus, we can compute in $O(n + 1/\varepsilon^{O(d^2)})$ time a set $Q \subseteq P$ of $1/\varepsilon^{O(d^2)}$ points so that for any line ℓ, $w(\ell, P) \geq w(\ell, Q) \geq (1 - \varepsilon)w(\ell, P)$ as well as a cylindrical shell of width at most $(1 + \varepsilon)w^*(P)$ that contains $\mathbf{P}$. Hence, we conclude the following.

Theorem 15.1.3 *Given a set* $\mathbf{P}$ *of n points in* $\mathbb{R}^d$ *and a parameter* $\varepsilon > 0$, *we can compute in* $O(n + 1/\varepsilon^{O(d^2)})$ *time a subset* $\mathcal{S} \subseteq P$ *of size* $O(1/\varepsilon^{O(d^2)})$ *so that for any line* ℓ *in* $\mathbb{R}^d$, *we have* $w(\ell, \mathcal{S}) \geq (1 - \varepsilon)w(\ell, P)$.

Note, that Theorem 15.1.3 does not compute the optimal cyilinder, it just computes a small coreset for this problem. Clearly, we can now run any brute force algorithm on this coreset. This would result in running time $O(n + 1/\varepsilon^{O(d^4)})$, which would output a cylinder which if expanded by factor $1 + \varepsilon$, will cover all the points of P. In fact, the running time can be further improved.

15.2 Exercises

15.3 Bibliographical notes

Section 15.1.1 is from Agarwal *et al.* [AHV04], and the results can be (very slightly) improved by treating hte direction as $(\mathsf{d} - 1)$-dimensional entity, see [AHV04] for details.

Theorem 15.1.3 is also from [AHV04]. The improved running time to compute the approximate cylinder mentioned in the text, follows by a more involved algorithm, which together with the construction of the coreset, also compute a compact representation of the extent of the coreset. The technical details are not trivial, and we skip them. In particular, the resulting running time for computing the approximate cylinderical shell is $O(n + 1/\varepsilon^{O(d^2)})$. See [AHV04] for more details.

Computing a compact representation of the extent of hyperplanes. FILL IN

Chapter 16

16 - Approximation Using Shell Sets

"And so ended Svejk's Budejovice anabasis. It is certain that if Svejk had been granted liberty of movement he would have got to Budejovice on his own. However much the authorities may boast that it was they who brought Svejk to his place of duty, this is nothing but a mistake. With Svejk energy and irresistible desire to fight, the authorities action was like throwing a spanner into the works."
– – The good soldier Svejk, Jaroslav Hasek

16.1 Covering problems, expansion and shell sets

Consider a set $\mathbf{P}$ of n points in $\mathbb{R}^{\mathsf{d}}$, that we are interested in covering by the best shape in a family of shapes $\mathcal{F}$. For example, $\mathcal{F}$ might be the set of all balls in $\mathbb{R}^{\mathsf{d}}$, and we are looking for the minimum enclosing ball of $\mathbf{P}$. A ε-coreset $\mathcal{S} \subseteq \mathbf{P}$ would guarantee that *any* ball that covers $\mathcal{S}$ will cover the whole point set if we expand it by $(1 + \varepsilon)$.

However, sometimes, computing the coreset is computationally expensive, the coreset does not exist at all, or its size is prohibitively large. It is still natural to look for a small subset $\mathfrak{S}$ of the points, such that finding the optimal solution for $\mathfrak{S}$ generates (after appropriate expansion) an approximate solution to the original problem.

Definition 16.1.1 (Shell sets) Given a set $\mathbf{P}$ of points (or geometric objects) in $\mathbb{R}^{\mathsf{d}}$, and $\mathcal{F}$ be a family of shapes in $\mathbb{R}^{\mathsf{d}}$. Let $f : \mathcal{F} \to \mathbb{R}$ be a target optimization function, and assume that there is a natural expansion operation defined over $\mathcal{F}$. Namely, given a set $\mathbf{r} \in \mathcal{F}$, one can compute a set $(1 + \varepsilon)\mathbf{r}$ which is the expansion of $\mathbf{r}$ by a factor of $1 + \varepsilon$. In particular, we would require that $f((1 + \varepsilon)\mathbf{r}) \leq (1 + \varepsilon) f(\mathbf{r})$.

Let $f_{\mathrm{opt}}(\mathbf{P}) = \min_{\mathbf{r} \in \mathcal{F}, \mathbf{P} \subseteq \mathbf{r}} f(\mathbf{r})$ be the shape in $\mathcal{F}$ that bests fits $\mathbf{P}$.

Furthermore, assume that $f_{\mathrm{opt}}(\cdot)$ is a monotone function, that is for $A \subseteq B \subseteq \mathbf{P}$ we have $f_{\mathrm{opt}}(A) \leq f_{\mathrm{opt}}(B)$.

A subset $\mathfrak{S} \subseteq \mathbf{P}$ is a *ε-shell set* for $\mathbf{P}$, if `SlowAlg` on a set B that contains $\mathfrak{S}$, if the range $\mathbf{r}$ returned by `SlowAlg`$(\mathfrak{S})$ covers $\mathfrak{S}$, $(1 + \varepsilon)\mathbf{r}$ covers $\mathbf{P}$, and $f(\mathbf{r}) \leq (1 + \varepsilon) f_{\mathrm{opt}}(\mathfrak{S})$. Namely, the range $(1 + \varepsilon)\mathbf{r}$ is an $(1 + \varepsilon)$-approximation to the optimal range of $\mathcal{F}$ covering $\mathbf{P}$.

A shell set $\mathfrak{S}$ is a *monotone ε-shell set* if for any subset B containing $\mathfrak{S}$, if we apply `SlowAlg`(B) and get the range $\mathbf{r}$, then $\mathbf{P}$ is contained inside $(1 + \varepsilon)\mathbf{r}$ and $\mathbf{r}$ covers B.

Note, that ε-shell sets are considerably more restricted and weaker than coresets. Of course, a ε-coreset is automatically a (monotone) ε-shell set. Note also, that if a problem has a monotone shell set, then to approximate it efficiently, all we need to do is to find some set, hopefully small, that contains the shell set.

```
ComputeShellSet(P)
We initialize all the points of P to have weight 1, and we repeatedly do the following:
    • Pick a random sample $R$ from P of size $r = O((\alpha/\delta)\log(\alpha/\delta))$, where $\delta = 1/(4k_{\mathrm{opt}})$. With
      constant probability $R$ is a $\delta$-net for P by Theorem 12.2.5.
    • Compute, using SlowAlg($R$) the range r in $\mathcal{F}$, such that $(1+\varepsilon)$r covers $R$ and realizes
      (maybe approximately) $f_{\mathrm{opt}}(R)$.
    • Compute the set $S$ of all the points of P outside $(1+\varepsilon)$r. If the total weight of those points
      exceeds $\delta w(P)$ then the random sample is bad, and return to the first step.
    • If the set $S$ is empty then return $R$ as the required shell set, and r as the approximation.
    • Otherwise, double the weight of the points of $S$.
When done, return r and the set $R$.
```

Figure 16.1: The algorithm for approximating optimal cover and computing a small shell set.

16.2 The Setting

Let $\mathbf{P}$ be a set of n points in $\mathbb{R}^d$, and let $\mathcal{F}$ be a family of shapes in $\mathbb{R}^d$. Furthermore, let us assume that the range space $X = (\mathbb{R}^d, \mathcal{F})$ has low VC dimension α. Finally, assume that we want to compute the best shape in $\mathcal{F}$ that covers $\mathbf{P}$ under a target function $f(\cdot)$. Namely, we would like to compute $f_{\mathrm{opt}}(P) = \min_{\mathbf{r}\in\mathcal{F}, P\subseteq B} f(\mathbf{r})$.

Assume that $f_{\mathrm{opt}}(\cdot)$ is a *monotone target function*. Namely, if $S \subseteq T \subseteq \mathbf{P}$ then $f_{\mathrm{opt}}(S) \leq f_{\mathrm{opt}}(T)$. This monotonicity property holds (almost) always for problems with small coresets.

Next, assume that we only have a slow algorithm `SlowAlg`that can solve (maybe approximately) the given optimization problem. Namely, given a subset $S \subseteq \mathbf{P}$, it computes a range $\mathbf{r} \in \mathcal{F}$ such that $f(\mathbf{r}) \leq (1+\varepsilon) f_{\mathrm{opt}}(S)$, and the running time of `SlowAlg` is $T_{\mathtt{SlowAlg}}(|S|)$.

Finally, assume that we know that a small *monotone* shell set of size k_{opt} exists for $\mathbf{P}$, but unfortunately we have no way of computing it explicitly (because, for example, we only have a constructive proof of the existence of such a shell set).

A natural question is how to compute this small shell set quickly, or alternatively compute an approximate shell set which is not much bigger. Clearly, once we have such a small shell set, we can approximate the optimal cover for $\mathbf{P}$ in $\mathcal{F}$.

Example. We start with a toy example, a more interesting example is given below. Let $\mathcal{F}$ be the set of all balls in $\mathbb{R}^d$, and let $f(\mathbf{r})$ be the radius of the ball $\mathbf{r} \in \mathcal{F}$. It is known that there is a ε-shell set for the minimum radius ball of size $O(1/\varepsilon)$ (we will prove this fact later in the course). The expansion here is the natural enlargement of a ball radius.

16.3 The Algorithm for Computing the Shell Set

Assume, that a kind oracle, told us that the there exist a monotone ε-shell set for $\mathbf{P}$ of size k_{opt}, and that $\mathcal{F}$ is of VC dimension α. The algorithm to approximate the optimal cover of $\mathbf{P}$ and extract a small shell set is depicted in Figure 16.1. Note, that if we do not have a kind oracle at our possession, we can just perform a binary search for the right value of k_{opt}.

There are several non-trivial technicalities in implementing this algorithm The first one is that Theorem 12.2.5 is for unweighted sets, but by replicating a point p of w_p times (conceptually), where w_p is the weight of p, it follows that it still holds in this weighted settings.

Random sampling from a weighted set. Another technicality is that the weights might be quite large. To overcome this, we will store the weight of an element by storing an index i, such that the weight of the element is 2^i, We still need to do m independent draws from this weighted set. The easiest way to do that, is to compute the element e in of $\mathbf{P}$ in maximum weight, and observing that all elements of weight $\leq w_e/n^{10}$ have weight which is so tiny, so that it can be ignored, where w_p is the weight of e. Thus, normalize all the weights of by dividing them by $2^{\lfloor \lg w_e/n^{10} \rfloor}$, and remove all elements with weights smaller than 1. For a point p, let $\widehat{\omega}(p)$ denote its normalized weight. Clearly, all the normalized weights are integers in the range $1, \ldots, 2n^{10}$. Thus, we now have to pick points for a set with (small) integer weights. Place the elements in an array, and compute the prefix sum array of their weights. That is $a_k = \sum_{i=1}^{k} \widehat{\omega}(p_i)$, for $i = 1, \ldots, n$. Next, pick a random number γ uniformly in the range $[0, a_n]$, and using a binary search, find the j, such that $a_{j-1} \leq \gamma < a_j$. This picks the points p_j to be in the random sample. This requires $O(n)$ preprocessing, but a single random sample can now be done in $O(\log n)$ time. We need to perform r independent samples. Thus, this takes $O(n + r \log n)$.

16.3.1 Correctness

Lemma 16.3.1 *The algorithm described above computes a ε-shell set for $\mathbf{P}$ of size $O(r) = O(k_{\mathrm{opt}}\alpha \log (k_{\mathrm{opt}}\alpha))$. The algorithm performs $O(4k_{\mathrm{opt}} \ln n)$ iterations.*

Proof: We only need to prove that the algorithm terminates in the claimed number of iterations. Observe, that with constant probability (say ≥ 0.9), the sample R_i, in the ith iteration, is an δ-net for $\mathbf{P}_{i-1}$ (the weighted version of $\mathbf{P}$ in the end of the $(i-1)$th iteration), in relation to the ranges of $\mathcal{F}$. Observe, that this also implies that R_i is a δ-net for the complement family $\overline{\mathcal{F}} = \left\{ \mathbb{R}^{\mathsf{d}} \setminus \mathbf{r} \;\middle|\; \mathbf{r} \in \mathcal{F} \right\}$, with constant probability (since $(\mathbb{R}^{\mathsf{d}}, \mathcal{F})$ and $(\mathbb{R}^{\mathsf{d}}, \overline{\mathcal{F}})$ have the same VC dimension).

If R_i is such a δ-net, then we know that range $\mathbf{r}$ we compute completely covers the set R_i, and as such, for any range $\mathbf{r}' \in \overline{\mathcal{F}}$ that avoids R_i we have $w(\mathbf{r}') \leq \delta w(\mathbf{P}_i)$. In particular, this implies that $\omega(S_i) \leq \delta w(\mathbf{P}_{i-1})$. If not, than R_i is not a δ-net, and we resample. The probability for that is ≤ 0.1. As such, we expect to repeat this $O(1)$ times in each iteration, till we have $w(S_i) \leq \delta w(\mathbf{P}_{i-1})$.

Thus, in each iteration, the algorithm doubles the weight of at most a δ-fraction of the total point set. Thus $w(\mathbf{P}_i) \leq (1+\delta) w(\mathbf{P}_{i-1}) = n(1+\delta)^i$.

On the other hand, consider the smallest shell $\mathfrak{S}$ of $\mathbf{P}$, which is of size k_{opt}. If all the elements of $\mathfrak{S}$ are in R_i, then the algorithm would have terminated, since $\mathfrak{S}$ is a monotone shell set1. Thus, if we continue to the next iteration, it must be that $|\mathfrak{S} \cap S_i| \geq 1$. In particular, we are doubling the weight of at least one element of the shell set. We conclude that the weight of $\mathbf{P}_i$ in the ith iteration, is at least

$$k_{\mathrm{opt}} 2^{i/k_{\mathrm{opt}}},$$

since in every iteration at least one element of $\mathfrak{S}$ gets its weight redoubled. Thus, we have

$$\exp\left(\frac{i}{2k_{\mathrm{opt}}}\right) \leq 2^{i/k_{\mathrm{opt}}} \leq k_{\mathrm{opt}} 2^{i/k_{\mathrm{opt}}} \leq (1+\delta)^i n \leq n \cdot \exp(\delta i) = n \cdot \exp\left(\frac{i}{4k_{\mathrm{opt}}}\right).$$

Namely, $\exp\left(\frac{i}{4k_{\mathrm{opt}}}\right) \leq n$. Implying that $i \leq 4k_{\mathrm{opt}} \ln n$. Namely, after $4k_{\mathrm{opt}} \ln n$ iterations the algorithm terminates, and thus returns the required shell set and approximation. ■

Theorem 16.3.2 *Under the settings of Section 16.2, one can compute a monotone ε-shell set for $\mathbf{P}$ of size $O(k_{\mathrm{opt}}\alpha \log(k_{\mathrm{opt}}\alpha))$. The running time of the resulting algorithm is $O\Big((n + T(k_{\mathrm{opt}}\alpha \log(k_{\mathrm{opt}}\alpha)))k_{\mathrm{opt}} \ln n\Big)$, with high probability, for $k_{\mathrm{opt}} \leq n/\log^3 n$. Furthermore, one can compute an ε-approximation to $f_{\mathrm{opt}}(P)$ in the same time bounds.*

Proof: The algorithm is described above. The bounds on the running time follows from the bounds on the number of iterations from Lemma 16.3.1. The only problem we need to address, is that the resampling would repeatedly fail, and the algorithm would spend exuberant amount of time on resampling. However, the probability of failure in sampling is ≤ 0.1. Furthermore, we need at most $4k_{\mathrm{opt}} \log n$ good samples before the algorithm succeeds. It is now straightforward to show using Chernoff inequality, that with high probability, we will perform at most $8k_{\mathrm{opt}} \log n$ samplings before achieving the required number of good samples. ■

16.3.2 Set Covering in Geometric Settings

Interestingly, the algorithm we discussed, can be used to get an improved approximation algorithm for the set covering problem in geometric settings. We remind the reader that set covering is the following problem.

Problem: Set Covering

Instance: $(S, \mathcal{F})$
S - a set of n elements
$\mathcal{F}$ - a family of subsets of S, s.t. $\bigcup_{X \in \mathcal{F}} X = S$.
Question: What is the set $\mathcal{X} \subseteq \mathcal{F}$ such that $\mathcal{X}$ contains as few sets as possible, and $\mathcal{X}$ covers S?

The natural algorithm for this problem is the greedy algorihtm that repeatedly pick the set in the family $\mathcal{F}$ that covers the largest number of uncovered elements in S. It is not hard to show that this provides a $O(|S|)$ apporximation. In fact, it is known that set covering can be better approximated unless $P = NP$.

Assume, however, that we know that the VC dimension of the set system $(S, \mathcal{F})$ has VC dimension α. In fact, we need a stronger fact, that the dual family

$$\mathsf{S} = \left(\mathcal{F}, \left\{ U(s, \mathcal{F}) \;\middle|\; s \in S \right\} \right),$$

is of low VC dimension α, where $U(s, \mathcal{F}) = \left\{ X \;\middle|\; s \in X, X \in \mathcal{F} \right\}$.

It turns out that the algorithm of Figure 16.1 also works in this setting. Indeed, we set the weight of the sets to 1, we pick a random sample of sets. IF they cover the universe S, we are done. Otherwise, there must be a point p which is not covered. Arguing as above, we know that the random sample is a δ-net of (the weighted) S, and as such all the sets containing p have total weight $\leq \delta(\mathsf{S})$. As such, double the weight of all the sets covering p, and repeat. Arugeing as above, one can show that the algorithm terminates after $O(k_{\mathrm{opt}} \log m)$ iterations, where m is the number of sets, where k_{opt} is the number of sets in the optimal cover of S. Furhtermore, the size of the cover generated is $O(k_{\mathrm{opt}}\alpha \log(k_{\mathrm{opt}}\alpha))$.

Theorem 16.3.3 *Let $(S, \mathcal{F})$ be a range space, such that the dual range space S has VC dimension α. Then, one can compute a set covering for S using sets of $\mathcal{F}$ using $O(k_{\mathrm{opt}}\alpha \log(k_{\mathrm{opt}}\alpha))$ sets. This requires $O(k_{\mathrm{opt}} \log n)$ iterations, and takes polynomial time.*

Note, that we did not provide in Theorem 16.3.3 exact running time bounds. Usually in geometric settings, one can get improved running time using the underlying geometry. Interestingly, the property that the dual system has low VC dimension "buys" one a lot, as it implies that one can do $O(\log k_{\mathrm{opt}})$ approximation, instead of $O(\log n)$ in the general case.

16.4 Application - Covering Points by Cylinders

16.5 Clustering and Coresets

We would like to cover a set $\mathbf{P}$ of n points $\mathbb{R}^d$ by k balls, such that the radius of maximum radius ball is minimized. This is known as the *k-center clustering* problem (or just *k-center*). The *price function*, in this case, $\mathrm{rd}_k(P)$ is the radius of the maximum radius ball in the optimal solution.

Definition 16.5.1 Let $\mathbf{P}$ be a point set in $\mathbb{R}^d$, $1/2 > \varepsilon > 0$ a parameter.

For a cluster c, let $c(\delta)$ denote the cluster resulting form expanding c by δ. Thus, if c is a ball of radius r, then $c(\delta)$ is a ball of radius $r + \delta$. For a set C of clusters, let

$$C(\delta) = \left\{ c(\delta) \;\middle|\; c \in C \right\},$$

be the *additive expansion operator*; that is, $C(\delta)$ is a set of clusters resulting form expanding each cluster of C by δ.

Similarly,

$$(1+\varepsilon)C = \left\{ (1+\varepsilon)c \;\middle|\; c \in C \right\},$$

is the *multiplicative expansion operator*, where $(1+\varepsilon)c$ is the cluster resulting from expanding c by a factor of $(1+\varepsilon)$. Namely, if C is a set of balls, then $(1+\varepsilon)C$ is a set of balls, where a ball $c \in C$, corresponds to a ball radius $(1+\varepsilon)\,\mathrm{radius}(c)$ in $(1+\varepsilon)C$.

A set $\mathcal{S} \subseteq \mathbf{P}$ is an (additive) *ε-coreset* of $\mathbf{P}$, in relation to a price function radius, if for any clustering C of $\mathcal{S}$, we have that $\mathbf{P}$ is covered by $C(\varepsilon\,\mathrm{radius}(C))$, where $\mathrm{radius}(C) = \max_{c \in C} \mathrm{radius}(c)$. Namely, we expand every cluster in the clustering by an ε-fraction of the size of the *largest* cluster in the clustering. Thus, if C is a set of k balls, then $C(\varepsilon f(C))$ is just the set of balls resulting from expanding each ball by εr, where r is the radius of the largest ball.

A set $\mathcal{S} \subseteq \mathbf{P}$ is a *multiplicative ε-coreset* of $\mathbf{P}$, if for any clustering C of $\mathcal{S}$, we have that $\mathbf{P}$ is covered by $(1+\varepsilon)C$.

Lemma 16.5.2 *Let $\mathbf{P}$ be a set of n points in $\mathbb{R}^d$, and $\varepsilon > 0$ a parameter. There exists an additive ε-coreset for the k-center problem, and this coreset has $O(k/\varepsilon^d)$ points.*

Proof: Let C denote the optimal clustering of $\mathbf{P}$. Cover each ball of C by a grid of side length $\varepsilon r_{\mathrm{opt}}/d$, where r_{opt} is the radius of the optimal k-center clustering of $\mathbf{P}$. From each such grid cell, pick one points of $\mathbf{P}$. Clearly, the resulting point set $\mathcal{S}$ is of size $O(k/\varepsilon^d)$ and it is an additive coreset of $\mathbf{P}$. ■

The following is a minor extension of an argument used in [APV02].

Lemma 16.5.3 *Let $\mathbf{P}$ be a set of n points in $\mathbb{R}^d$, and $\varepsilon > 0$ a parameter. There exists a multiplicative ε-coreset for the k-center problem, and this coreset has $O\left(k!/\varepsilon^{dk}\right)$ points.*

Proof: For $k = 1$, the additive coreset of $\mathbf{P}$ is also a multiplicative coreset, and it is of size $O(1/\varepsilon^d)$.

As in the proof of Lemma 16.5.2, we cover the point set by a grid of radius $\varepsilon r_{\mathrm{opt}}/(5d)$, let SQ the set of cells (i.e., cubes) of this grid which contains points of $\mathbf{P}$. Clearly, $|\mathrm{SQ}| = O(k/\varepsilon^d)$.

Let $\mathcal{S}$ be the additive ε-coreset of $\mathbf{P}$. Let C be any k-center clustering of $\mathcal{S}$, and let Δ be any cell of SQ.

If Δ intersects all the k balls of C, then one of them must be of radius at least $(1 - \varepsilon/2)\mathrm{rd}(P, k)$. Let c be this ball. Clearly, when we expand c by a factor of $(1+\varepsilon)$ it would completely cover Δ, and as such it would also cover all the points of $\Delta \cap \mathbf{P}$.

Thus, we can assume that Δ intersects at most $k-1$ balls of C. As such, we can inductively compute an ε-multiplicative coreset of $P \cap \Delta$, for $k-1$ balls. Let Q_Δ be this set, and let $Q = \mathcal{S} \cup \bigcup_{\Delta \in \mathrm{SQ}} Q_\Delta$.

Note that $|Q| = T(k,\varepsilon) = O(k/\varepsilon^d)T(k-1,\varepsilon) + O(k/\varepsilon^{\mathsf{d}}) = O\Big(k!/\varepsilon^{dk}\Big)$. The set Q is the required multiplicative coreset by the above argumentation. ■

16.6 Union of Cylinders

Let assume we want to cover $\mathbf{P}$ by k cylinders of minimum maximum radius (i.e., fit the points to k lines). Formally, consider $\mathcal{G}$ to be the set of all cylinders in $\mathbb{R}^{\mathsf{d}}$, and let $\mathcal{F} = \left\{ c_1 \cup c_2 \cup \ldots \cup c_k \;\middle|\; c_1, \ldots, c_k \in \mathcal{F} \right\}$ be the set, which its memebers are union of k cyilinders. For $C \in \mathcal{F}$, let $f(C) = \max_{c \in C} \mathrm{radius}(c)$. Let $f_{\mathrm{opt}}(P) = \min_{C \in \mathcal{F}, P \subseteq C} f(C)$.

One can compute the optimal cover of $\mathbf{P}$ by k cylinders in $O(n^{(2\mathsf{d}-1)k+1})$ time, see below for details. Furthermore, $(\mathbb{R}^{\mathsf{d}}, \mathcal{F})$ has VC dimesnion $\alpha = O(dk \log(dk))$. Finally, one can show that this set of cylinders has ε-coreset of small size ???. Thus, we would like to compute a small ε-coreset, and compute an approximation quickly.

16.6.0.1 Covering by Cylinders - A Slow Algorithm

It is easy to verify (but tedios) that the VC dimension of $(\mathbb{R}^{\mathsf{d}}, \mathcal{F}_k)$ is bounded by $\alpha = O(dk \log(dk))$. Furthermore, it has a small coreset (see Section **??**). Furthermore, given a set $\mathbf{P}$ of n points, and consider its minimum radius enclosing cylinder c. The cyilinder c has (at most) $2\mathsf{d}-2$ points of $\mathbf{P}$ on its boundary which if we compute their minimum enclosing cylinder, it is c. Note, that c might contain even more points on its boundary, we are only claiming that there is a defining subset of size $2\mathsf{d}-1$. This is one of those "easy to see" but very tedious to verify facts. Let us quicly outline an intuitive explanation (but not a proof!) of this fact. Consider the set of lines $\mathcal{L}^{\mathsf{d}}$ of lines in $\mathbb{R}^{\mathsf{d}}$. Every member of $\ell \in \mathcal{L}^{\mathsf{d}}$ can be parameterized by the closest point p on ℓ to the origin, and consider the hyperplane that passes through p and is orthogonal to op, where o is the origin. The line ℓ now can be parameterized by its orientation in h. This requires specifying a point on the $\mathsf{d}-2$ dimensional unit hypersphere $\mathbb{S}^{(\mathsf{d}-2)}$. Thus, we can specify ℓ using $2\mathsf{d}-2$ real numbers. Next, define for each point $p_i \in \mathbf{P}$, its distance $g_i(\ell)$ from $\ell \in \mathcal{L}^{\mathsf{d}}$. This is a messy but a nice algebraic function defined over $2\mathsf{d}-2$ variables. In particualr, $g_i(\ell)$ induces a surface in $2\mathsf{d}-1$ dimensions (i.e., $\cup_\ell(\ell, g_i(\ell))$). Consider the arrangement $\mathcal{A}$ of those surfaces. Clearly, the minimum volume cylinder lies on a feature of this arrangement, thus to specify the minimum radius cylinder, we just need to specify the feature (i.e., vertex, edge, etc) of the arrangement that contains this point. However, every feature in an arrangement of well behaved surfaces in $2\mathsf{d}-1$ dimensions, can be specified by $2\mathsf{d}-1$ surfaces. (This is intuitvely clear but requires a proof - an intersection of k surfaces, is going to be $d-k$ dimensional, where d is the dimension of the surfaces. If we add a surface to the intersection and it does not reduce the dimension of the intersection, we can reject it, and take the next surface passing through the feature we care about.). Every such surface corresponds to a original point.

Thus, if we want to specify a minimum radius cylinder induced by a subset of $\mathbf{P}$, all we need to specify are $2\mathsf{d}-1$ points. To specify k such cylinders, we need to specify $M = (2\mathsf{d}-1)k$ points. This immediately implies that we can find the optimal cover of $\mathbf{P}$ by k cylinders in $O(n^{(2\mathsf{d}-1)k+1})$ time, but just enumerating all such subsets of M points, and computing for each subset its optimal cover (note, that the O notdation hids a constant that depends on k and d).

Thus, we have a slow algorithm that can compute the optimal cover of $\mathbf{P}$ by k cylinders.

16.6.1 Existence of a Small Coreset

Since the coreset in this case is either multiplicative or additive, it is first important to define the expansion operation carefully. In particualr, if C is a set of k cylinders, the $(1+\varepsilon)$-expanded set of cylinders would be $C(\varepsilon \,\mathrm{radius}(C))$, where $\mathrm{radius}(C)$ is the radius of the largest cylinder in C.

Let $\mathbf{P}$ be the given set of n points in $\mathbb{R}^{\mathsf{d}}$. Let C_{opt} be the optimal cover of $\mathbf{P}$ by k cylinders. For each cylinder of C_{opt} place $O(1/\varepsilon^{\mathsf{d}-1})$ parallel lines inside it, so that for any point inside the union of the cylinders, there is a line in this family in distance $\leq (\varepsilon/10) r_{\mathrm{opt}}$ from it. Let $\mathfrak{L}$ denote this set of lines.

Let $\mathbf{P}'$ be the point set resulting from snapping each point of $\mathbf{P}$ to its closest point on $\mathfrak{L}$. We claim that $\mathbf{P}'$ is a $(\varepsilon/10)$-coreset for $\mathbf{P}$, as can be easily verified. Indeed, if a set C of k cylinders cover $\mathbf{P}'$, then the largest cylinder must be of radius $r \geq (1-\varepsilon/10)\mathrm{rd}(\mathbf{P},k)$, where $\mathrm{rd}(\mathbf{P},k)$ is the radius of the optimal coverage of $\mathbf{P}$ by k cylinders. Otherwise, $\mathrm{rd}(\mathbf{P},k) \leq r + (\varepsilon/10)\mathrm{rd}(\mathbf{P},k) < \mathrm{rd}(\mathbf{P},k)$.

The set $\mathbf{P}'$ lies on $O(1/\varepsilon^{\mathsf{d}-1})$-lines. Let $\ell \in \mathfrak{L}$ be such a line, and consider the point-set $\mathbf{P}'_\ell$. Assume for a second that there was a multiplicative ε-coreset $\mathcal{T}_\ell$ on this line. If $\mathcal{T}_\ell$ is covered by k cylinders, each cylinder intersect ℓ along an interval. Expanding each such cylinder by a factor of $(1+\varepsilon)$ is equivalent to expanding each such intersecting interval by a factor of $1+\varepsilon$. However, by Lemma 16.5.3, we know that such a multiplicative $(\varepsilon/10)$-coreset exists, of size $O(1/\varepsilon^k)$. Thus, let $\mathcal{T}_\ell$ be the multiplicative $(\varepsilon/10)$-coreset for $\mathbf{P}'_\ell$ for k intervals on the line. Let $\mathcal{T} = \cup_{\ell \in \mathfrak{L}} \mathcal{T}_\ell$. We claim that $\mathcal{T}$ is a (additive) $(\varepsilon/10)$-coreset for $\mathbf{P}'$. This is trivial, since being a multiplicative coreset for each line imoplies that the union is a multiplicative coreset, and a δ-multiplicative coreset is also a δ-additive coreset. Thus, $\mathcal{T}$ is a $((1+\varepsilon/10)^2 - 1)$-coreset for $\mathbf{P}$. The only problem is that the points of $\mathcal{T}$ are not points in $\mathbf{P}$. However, they corresond to points in $\mathbf{P}$ which are in distance at most $(\varepsilon/10)\mathrm{rd}(\mathbf{P},k)$ from them. Let $\mathcal{S}$ be the corresponding set of points of $\mathbf{P}$. It is now easy to verify that $\mathcal{S}$ is indeed a ε-coreset for $\mathbf{P}$, since $((1+\varepsilon/10)^2 - 1) + \varepsilon/10 \leq \varepsilon$. We summerize:

Theorem 16.6.1 *Let $\mathbf{P}$ be a set of n points in $\mathbb{R}^{\mathsf{d}}$. There exists a (additive) ε-coreset for $\mathbf{P}$ of size $O(k/\varepsilon^{\mathsf{d}-1+k})$ for covering the points by k-cylinders of minimum radius.*

16.7 Bibliographical notes

Section 16.3.2 is due to Clarkson [Cla93]. This technique was used to approximate terrains [**?**], and covering polytopes [Cla93].

The observation that this argument can be used to speedup approximation algorithms is due to Agarwal *et al.* [APV02]. The discussion of shell sets is implict in the work of Bădoiu *et al.* [BHI02].

Chapter 17

Duality

I don't know why it should be, I am sure; but the sight of another man asleep in bed when I am up, maddens me. It seems to me so shocking to see the precious hours of a man's life - the priceless moments that will never come back to him again - being wasted in mere brutish sleep.
– – Jerome K. Jerome, Three men in a boat

Duality is a transformation that maps lines and points into points and lines, respectively, while preserving some properties in the process. Despite its relative simplicity, it is a powerful tool that can dualize what seems like "hard" problems into easy dual problems.

17.1 Duality of lines and points

Consider a line $\ell \equiv y = ax + b$ in two dimensions. It is being parameterized by two constants a and b, which we can interpret, paired together, as a point in the parametric space of the lines. Naturally, this also gives us a way of interpreting a point as defining coefficients of a line. Thus, conceptually, points are lines and lines are points.

Formally, the *dual point* to the line $\ell \equiv y = ax + b$ is the point $\ell^\star = (a, -b)$. Similarly, for a point $p = (c, d)$ its *dual line* is $p^\star = cx - d$. Namely,

$$\begin{aligned} \mathsf{p} = (a, b) \qquad &\Rightarrow \qquad \mathsf{p}^\star : y = ax - b \\ \ell : y = cx + d \qquad &\Rightarrow \qquad \ell^\star = (c, -d). \end{aligned}$$

We will consider a line $\ell \equiv y = cx + d$ to be a linear function in one dimension, and let $\ell(x) = cx + d$.

A point $\mathsf{p} = (a, b)$ lies *above* a line $\ell \equiv y = cx + d$ if p lies vertically above ℓ. Formally, we have that $b > \ell(a) = ca + d$. We will denote this fact by $\mathsf{p} > \ell$. Similarly, the point p lies *below* ℓ if $b < \ell(a) = ca + d$, denoted by $\mathsf{p} < \ell$.

A line ℓ *supports* a convex set $S \subseteq \mathbb{R}^2$ if it intersects S but the interior of S lies completely on one side of ℓ.

Basic properties. For a point $\mathsf{p} = (a, b)$ and a line $\ell \equiv y = cx + d$, we have:

(P1) $\mathsf{p}^{\star\star} = (\mathsf{p}^\star)^\star = \mathsf{p}$.

> Indeed, $\mathsf{p}^\star \equiv y = ax - b$ and $(\mathsf{p}^\star)^\star = (a, -(-b)) = \mathsf{p}$.

(P2) The point p lies above (resp. below, on) the line ℓ, if and only if the point $\ell^\star$ lies above (resp. below, on) the line $\mathsf{p}^\star$. (Namely, a point and a line change their vertical ordering in the dual.)

Indeed, $\mathsf{p} > \ell(a)$ if and only if $b > ca + d$. Similarly, $(c, -d) = \ell^\star > \mathsf{p}^\star \equiv y = ax - b$ if and only if $-d > ac - b$, which is equivalent to the above condition.

(P3) The vertical distance between p and ℓ is the same as that between $\mathsf{p}^\star$ and $\ell^\star$.

Indeed, the vertical distance between p and ℓ is $|b - \ell(a)| = |b - (ca + d)|$. The vertical distance between $\ell^\star = (c, -d)$ and $\mathsf{p}^\star \equiv y = ax - b$ is $|(-d) - \mathsf{p}^\star(c)| = |-d - (ac - b)| = |b - (ca + d)|$.

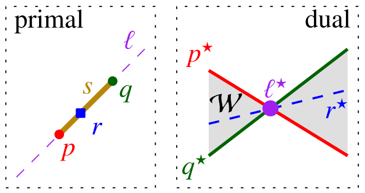
(P4) The vertical distance $\delta(\ell, \hbar)$ between two parallel lines ℓ and $\hbar \equiv y = ax + e$ is the same as the length of the vertical segment $\ell^\star\hbar^\star$.

The vertical distance between ℓ and $\hbar$ is $|b - e|$. Similarly, since $\ell^\star = (a, -b)$ and $\hbar^\star = (a, -e)$ we have that the vertical distance between them is $|(-b) - (-e)| = |b - e|$.

The missing lines. Consider the vertical line $\ell \equiv x = 0$. Clearly, ℓ does not have a dual point (specifically, its hypothetical dual point has an x coordinate with infinite value). In particular, our duality can not handle vertical lines. To visualize the problem, consider a sequence of non-vertical lines ℓ_i that converges to a vertical line ℓ. The sequence of dual points $\ell_i^\star$ is a sequence of points that diverges to infinity.

17.1.1 Examples

17.1.1.1 Segments and Wedges

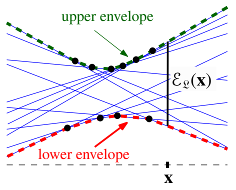


Consider a segment $s = \mathsf{pq}$ that lies on a line ℓ. Observe, that the dual of a point $\mathsf{r} \in \ell$ is a line $\mathsf{r}^\star$ that passes through the point $\ell^\star$. In fact, the two lines $\mathsf{p}^\star$ and $\mathsf{q}^\star$ define two double wedges. Let $\mathcal{W}$ be the double wedge that does not contain the vertical line that passes through $\ell^\star$.

Consider now the point r as it moves along s. When it is equal to p then its dual line $\mathsf{r}^\star$ is the line $\mathsf{p}^\star$. Now, as r moves along the segment s the dual line $\mathsf{r}^\star$ rotates around $\ell^\star$, till it arrives to $\mathsf{q}^\star$ (and then r reaches q).

What about the other wedge? It represents the two rays forming $\ell \setminus s$. The vertical line through $\ell^\star$ represents the singularity point in infinity where the two rays are "connected" together. Thus, as r travels along one of the rays (say starting at q) of $\ell \setminus s$, the dual line $\mathsf{r}^\star$ becomes steeper and steeper, till it becomes vertical. Now, the point r "jumps" from the "infinite endpoint" of the ray, to the "infinite endpoint" of the other ray. Simultaneously, the line $\mathsf{r}^\star$ is continuing to rotate from its current vertical position, sweeping over the whole wedge, till r travels back to p. (The reader that feels uncomfortable with notions line "infinite endpoint" can rest assured that the author feels the same way. As such, this should be taken as an intuitive description of whats going on and not a formally correct one.)

17.1.1.2 Convex hull and upper/lower envelopes



Consider a set of lines $\mathfrak{L}$ in the plane. The minimization diagram of $\mathfrak{L}$, known as the *lower envelope* of $\mathfrak{L}$, is the function $\mathcal{L}_{\mathfrak{L}} : \mathbb{R} \to \mathbb{R}$, where we have $\mathcal{L}(x) = \min_{\ell \in \mathfrak{L}} \ell(x)$, for $x \in \mathbb{R}$. Similarly, the *upper envelope* of $\mathfrak{L}$ is the function $\mathcal{U}(x) = \max_{\ell \in \mathfrak{L}} \ell(x)$, for $x \in \mathbb{R}$. The *extent* of $\mathfrak{L}$ at $x \in \mathbb{R}$ is the vertical distance between the upper and lower envelope at x; namely, $\mathcal{E}_{\mathfrak{L}}(x) = \mathcal{U}(x) - \mathcal{L}(x)$.

Computing the lower and/or upper envelopes can be useful. A line might represent a linear constraint, where the feasible solution must lie above this line. Thus, the feasible region is the region of points that lie above all the given lines. Namely, the region of the feasible solution is defined by the upper envelope of the lines. The upper envelope is just a polygonal chain made out of two infinite rays and a sequence of segments, where each segment/ray lies on one of the given lines. As such, the upper envelop can be described as the sequence of lines appearing on it, and the vertices where they change.

Developing an efficient algorithm for computing the upper envelope of a set of lines is a tedious but doable task. However, it becomes trivial if one uses duality.

Lemma 17.1.1 *Let $\mathfrak{L}$ be a set of lines in the plane. Let $\alpha \in \mathbb{R}$ be an any number, $\beta^- = \mathcal{L}_{\mathfrak{L}}(\alpha)$ and $\beta^+ = \mathcal{U}_{\mathfrak{L}}(\alpha)$. Let $\mathsf{p} = (\alpha, \beta^-)$ and $\mathsf{q} = (\alpha, \beta^+)$. Then:*

(i) the dual lines $\mathsf{p}^\star$ and $\mathsf{q}^\star$ are parallel, and they are both perpendicular to the direction $(\alpha, -1)$.

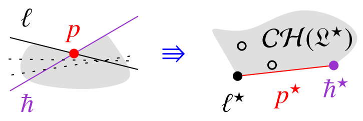
(ii) The lines $\mathsf{p}^\star$ and $\mathsf{q}^\star$ support $\mathcal{CH}(\mathfrak{L}^\star)$.

(iii) The extent $\mathcal{E}_{\mathfrak{L}}(\alpha)$ is the vertical distance between the lines $\mathsf{p}^\star$ and $\mathsf{q}^\star$.

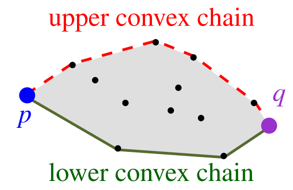
Proof: (i) We have $\mathsf{p}^\star \equiv y = \alpha x - \beta^-$ and $\mathsf{q}^\star \equiv y = \alpha x - \beta^+$. These two lines are parallel since they have the same slope. In particular, they are parallel to the direction $(1, \alpha)$. But this direction is perpendicular to the direction $(\alpha, -1)$.

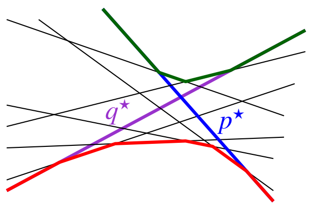
(ii) By property (P2), we have that all the points of $\mathfrak{L}^\star$ are below (or on) the line $\mathsf{p}^\star$. Furthermore, since p is on the lower envelope of $\mathfrak{L}$ it follows that $\mathsf{p}^\star$ must pass through one of the points $\mathfrak{L}^\star$. Namely, $\mathsf{p}^\star$ supports $\mathcal{CH}(\mathfrak{L}^\star)$ and it lies above it. Similar argument applies to $\mathsf{q}^\star$.

(iii) We have that $\mathcal{E}_{\mathfrak{L}}(\alpha) = \beta^+ - \beta^-$. The vertical distance between the two parallel lines $\mathsf{p}^\star$ and $\mathsf{q}^\star$ is $\mathsf{q}^\star(0) - \mathsf{p}^\star(0) = -\beta^+ - (-\beta^-) = \beta^+ - \beta^-$, as required. ■



Thus, consider a vertex p of the upper envelope of the set of lines $\mathfrak{L}$. The point p is the intersection point of two lines ℓ and $\hbar$ of $\mathfrak{L}$. Consider the dual set of points $\mathfrak{L}^\star$ and the dual line $\mathsf{p}^\star$. Since p lies above (or on) all the lines of $\mathfrak{L}$, by the above discussion, it must be that the line $\mathsf{p}^\star$ lies below (or on) all the points of $\mathfrak{L}^\star$. On the other hand, the line $\mathsf{p}^\star$ passes through the two points $\ell^\star$ and $\hbar^\star$. Namely, $\mathsf{p}^\star$ is a line that supports the convex hull of $\mathfrak{L}^\star$ and it passes through two of its vertices.





The convex hull of $\mathfrak{L}^\star$ is a convex polygon $\widehat{P}$, which can be broken into two convex chains by breaking it at the two extreme points in the x direction. We will refer to the upper polygonal chain of the convex hull as *upper convex chain* and to the other one as *lower convex chain*. In particular, two consecutive segments of the upper envelope corresponds to two consecutive vertices on the lower chain of the convex hull of $\mathfrak{L}^\star$. Thus, the convex-hull of $\mathfrak{L}^\star$ can be decomposed into two chains. The lower chain corresponds to the upper envelope of $\mathfrak{L}$, and the upper chain corresponds to the lower envelope of $\mathfrak{L}$. Of special interest are the two x extreme points p and q of the convex hull. They are the dual of the two lines with the highest/smallest slope in $\mathfrak{L}$ (we are assuming here that the slopes of lines in $\mathfrak{L}$ are distinct). These two lines appear on both the upper and lower envelope of the lines and they contain the four infinite rays of these envelopes.

Lemma 17.1.2 *Given a set $\mathfrak{L}$ of n lines in the plane, one can compute its lower and upper envelopes in $O(n \log n)$ time.*

Proof: One can compute the convex hull of n points in the plane in $O(n \log n)$ time. Thus, computing the convex hull of $\mathfrak{L}^\star$ and dualizing the upper and lower chains of $\mathcal{CH}(\mathfrak{L}^\star)$ results in the required envelopes. ■

17.2 Higher Dimensions

The above discussion can be easily extended to higher dimensions. We provide the basic properties without further proof, since they are easy extension of the two dimensional case. A hyperplane $h : x_\mathsf{d} = b_1 x_1 + \cdots + b_{\mathsf{d}-1} x_{\mathsf{d}-1} - b_\mathsf{d}$ in $\mathbb{R}^\mathsf{d}$ can be interpreted as a function from $\mathbb{R}^{\mathsf{d}-1}$ to $\mathbb{R}$. Given a point $\mathsf{p} = (p_1, \ldots, p_\mathsf{d})$ let $h(\mathsf{p}) = b_1 p_1 + \cdots + b_{\mathsf{d}-1} p_{\mathsf{d}-1} - b_\mathsf{d}$. In particular, a point p lies *above* the hyperplane h if $p_\mathsf{d} > h(\mathsf{p})$. Similarly, p is *below* the hyperplane h if $p_\mathsf{d} < h(\mathsf{p})$. Finally, a point is on the hyperplane if $h(\mathsf{p}) = p_\mathsf{d}$.

The *dual* of a point $\mathsf{p} = (p_1, \ldots, p_\mathsf{d}) \in \mathbb{R}^\mathsf{d}$ is a hyperplane $\mathsf{p}^\star \equiv x_\mathsf{d} = p_1 x_1 + \cdots p_{\mathsf{d}-1} x_{\mathsf{d}-1} - p_\mathsf{d}$, and the *dual* of a hyperplane $h \equiv x_\mathsf{d} = a_1 x_1 + a_2 x_2 + \cdots + a_{\mathsf{d}-1} x_{\mathsf{d}-1} + a_\mathsf{d}$ is the point $h^\star = (a_1, \ldots, a_{\mathsf{d}-1}, -a_\mathsf{d})$. There are several alternative definitions of duality, but they are essentially similar. Summarizing:

$$\begin{array}{rcl} \mathsf{p} = (p_1, \ldots, p_\mathsf{d}) & \Rightarrow & \mathsf{p}^\star \equiv x_\mathsf{d} = p_1 x_1 + \cdots p_{\mathsf{d}-1} x_{\mathsf{d}-1} - p_\mathsf{d} \\ h \equiv x_\mathsf{d} = a_1 x_1 + a_2 x_2 + \cdots + a_{\mathsf{d}-1} x_{\mathsf{d}-1} + a_\mathsf{d} & \Rightarrow & h^\star = (a_1, \ldots, a_{\mathsf{d}-1}, -a_\mathsf{d}). \end{array}$$

In the following we would slightly abuse notations, and for a point $\mathsf{p} \in \mathbb{R}^\mathsf{d}$ we will refer to $(p_1, \ldots, p_{\mathsf{d}-1}, \mathcal{L}_\mathcal{H}(\mathsf{p}))$ as the point $\mathcal{L}_\mathcal{H}(\mathsf{p})$. Similarly, $\mathcal{U}_\mathcal{H}(\mathsf{p})$ would denote the corresponding point on the upper envelope of $\mathcal{H}$.

The proof of the following lemma is an easy extension of the 2d case.

Lemma 17.2.1 *For a point* $\mathsf{p} = (b_1, \ldots, b_\mathsf{d})$*, we have:*

(i) $\mathsf{p}^{\star\star} = \mathsf{p}$.

(ii) The point p *lies above (resp. below, on) the hyperplane h, if and only if the point $h^\star$ lies above (resp. below, on) the hyperplane $h^\star$.*

(iii) The vertical distance between p *and h is the same as that between* $\mathsf{p}^\star$ *and $h^\star$.*

(iv) The vertical distance $\delta(h, g)$ between two parallel hyperplanes h and g is the same as the length of the vertical segment $h^\star g^\star$.

(v) Let $\mathcal{H}$ be the set of hyperplanes in $\mathbb{R}^d$. For any $x \in \mathbb{R}^{\mathsf{d}-1}$, the hyperplanes h and g dual to the points $\mathcal{L}_\mathcal{H}(\mathsf{p})$ *and* $\mathcal{U}_\mathcal{H}(\mathsf{p})$*, respectively, are parallel, normal to the vector* $(\mathsf{p}, -1) \in \mathbb{R}^\mathsf{d}$*, and supports the set $\mathcal{CH}(\mathcal{H}^\star)$. Furthermore, the points of $\mathcal{H}^\star$ lies below (resp., above) the hyperplane h (resp., above g). Also,* $\mathcal{E}_\mathcal{H}(\mathsf{p})$ *is the vertical distance between h and g.*

(vi) Computing the lower and upper envelope of $\mathcal{H}$ is equivalent to computing the convex hull of the dual set of points $\mathcal{H}^\star$.

17.3 Exercises

Exercise 17.3.1 Prove Lemma 17.2.1

Exercise 17.3.2 Show a counter example proving that no duality can preserve (exactly) orthogonal distances between points and lines.

17.4 Bibliographical notes

The duality discussed here should not be confused with linear programming duality [Van97]. Although the two topics seems to be connected somehow, the author is unaware of a natural and easy connection.

Duality of lines and points rises naturally in projective geometry, where it is a fundamental tool.

The "missing lines phenomena" is inherent to all dualities, since the space of a lines in the plane has the topology of an open Möbius strip which is not homeomorphic to the plane. There are a lot of other possible dualities, and the one presented here is the one most useful for our purposes.

A natural question is whether one can find a duality that preserves the orthogonal distances between lines and points. The surprising answer is no, as Exercise 17.3.2 testifies. In fact, it is not too hard to show using topological arguments that any duality must distort such distances arbitrarily bad [FH06].

Open Problem 17.4.1 *Given a set* **P** *of n points in the plane, and a set* $\mathfrak{L}$ *of n lines in the plane, consider the best possible duality (i.e., the one that minimizes the distortion of orthogonal distances) for* **P** *and* $\mathfrak{L}$*. What is the best distortion possible, as a function of n?*

Here, we define the distortion of the duality as

$$\max_{\mathsf{p}\in\mathbf{P},\ell\in\mathfrak{L}} \left(\frac{\mathbf{d}(\mathsf{p},\ell)}{\mathbf{d}(\mathsf{p}^\star,\ell^\star)}, \frac{\mathbf{d}(\mathsf{p}^\star,\ell^\star)}{\mathbf{d}(\mathsf{p},\ell)} \right).$$

A striking (negative) example of the power of duality is the work of Overmars and van Leeuwen [OvL81] on the dynamic maintenance of convex hull in 2d, and the maintenance of the lower/upper envelope of lines in the plane. Clearly, by duality, the two problems are identical. However, the authors (smart people indeed) did not observe it, and the paper is twice longer than it should be solving the two problems separately.

Duality is heavily used throughout computational geometry, and it is hard to imagine managing without it. Results and techniques that use duality include bounds on k-sets/k-levels [Dey98], partition trees [Mat92], and coresets for extent measure [AHV04] (this is a random short list of relevant results and it is by no means exhaustive).

Duality, Voronoi Diagrams and Delaunay Triangulations. Given a set **P** of points in $\mathbb{R}^{\mathsf{d}}$ its *Voronoi diagram*Voronoi Diagram is a partition of space into cells, where each cell is the region closest to one of the points of **P**. The Delaunay triangulation of a point-set is a planar graph (for $\mathsf{d} = 2$) where two points are connected by a straight segment if there is a ball that touches both points and its interior is empty. It is easy to verify that these two structures are dual to each other in the sense of graph duality. Maybe more interestingly, this duality has also an easy geometric interpretation.

Indeed, given **P**, its Voronoi diagram boils down to the computation of the lower envelope of cones. This set of cones can be linearized and then, the computation of the Voronoi diagram boils down to computing the lower envelope of hyperplanes, one hyperplane for each point of **P**. Similarly, the computation of the Delaunay triangulation of **P** can be reduced, after lifting the points to the hyperboloid, to the computation of the convex hull of the points. In fact, the projection down of the lower part of the convex hull is the required triangulation. Thus, the two structures are dual to each other also lifting/linearization and direct duality. The interested reader should check out [dBvKOS00].

Chapter 18

Finite Metric Spaces and Partitions

18.1 Finite Metric Spaces

Definition 18.1.1 A *metric space* is a pair $(\mathcal{X}, \mathbf{d})$ where $\mathcal{X}$ is a set and $\mathbf{d} : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ is a *metric*, satisfying the following axioms: (i) $\mathbf{d}(x, y) = 0$ iff $x = y$, (ii) $\mathbf{d}(x, y) = \mathbf{d}(y, x)$, and (iii) $\mathbf{d}(x, y) + \mathbf{d}(y, z) \geq \mathbf{d}(x, z)$ (triangle inequality).

For example, $\mathbb{R}^2$ with the regular Euclidean distance is a metric space.

It is usually of interest to consider the finite case, where $\mathcal{X}$ is an n-point set. Then, the function $\mathbf{d}$ can be specified by $\binom{n}{2}$ real numbers. Alternatively, one can think about $(\mathcal{X}, \mathbf{d})$ is a weighted complete graph, where we specify positive weights on the edges, and the resulting weights on the edges comply with the triangle inequality.

In fact, finite metric spaces rise naturally from (sparser) graphs. Indeed, let $G = (\mathcal{X}, E)$ be an undirected weighted graph defined over $\mathcal{X}$, and let $\mathbf{d}_G(x, y)$ be the length of the shortest path between x and y in G. It is easy to verify that $(\mathcal{X}, \mathbf{d}_G)$ is a finite metric space. As such if the graph G is sparse, it provides a compact representation to the finite space $(\mathcal{X}, \mathbf{d}_G)$.

Definition 18.1.2 Let $(\mathcal{X}, d)$ be an n-point metric space. We denote the *open ball* of radius r about $x \in \mathcal{X}$, by $\mathbf{b}(x, r) = \left\{ y \in \mathcal{X} \,\middle|\, \mathbf{d}(x, y) < r \right\}$.

Underling our discussion of metric spaces are algorithmic applications. The hardness of various computational problems depends heavily on the structure of the finite metric space. Thus, given a finite metric space, and a computational task, it is natural to try to map the given metric space into a new metric where the task at hand becomes easy.

Example 18.1.3 For example, computing the diameter is not trivial in two dimensions, but is easy in one dimension. Thus, if we could map points in two dimensions into points in one dimension, such that the diameter is preserved, then computing the diameter becomes easy. In fact, this approach yields an efficient approximation algorithm, see Exercise 18.7.3 below.

Of course, this mapping from one metric space to another, is going to introduce error. We would be interested in minimizing the error introduced by such a mapping.

Definition 18.1.4 Let $(\mathcal{X}, \mathbf{d}_{\mathcal{X}})$ and $(Y, \mathbf{d}_Y)$ be metric spaces. A mapping $f : \mathcal{X} \to Y$ is called an *embedding*, and is *C-Lipschitz* if $\mathbf{d}_Y(f(x), f(y)) \leq C \cdot \mathbf{d}_{\mathcal{X}}(x, y)$ for all $x, y \in \mathcal{X}$. The mapping f is called *K-bi-Lipschitz* if there exists a $C > 0$ such that

$$CK^{-1} \cdot \mathbf{d}_{\mathcal{X}}(x, y) \leq \mathbf{d}_Y(f(x), f(y)) \leq C \cdot \mathbf{d}_{\mathcal{X}}(x, y),$$

for all $x, y \in \mathcal{X}$.

The least K for which f is K-bi-Lipschitz is called the *distortion* of f, and is denoted $\mathrm{dist}(f)$. The least distortion with which $\mathcal{X}$ may be embedded in Y is denoted $c_Y(\mathcal{X})$.

There are several powerful results in this vain, that show the existence of embeddings with low distortion that would be presented:

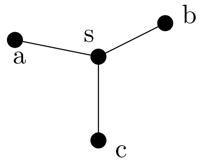
1. Probabilistic trees - every finite metric can be randomly embedded into a tree such that the "expected" distortion for a specific pair of points is $O(\log n)$.

2. Bourgain embedding - shows that any n-point metric space can be embedded into (finite dimensional) metric space with $O(\log n)$ distortion.

3. Johnson-Lindenstrauss lemma - shows that any n-point set in Euclidean space with the regular Euclidean distance can be embedded into $\mathbb{R}^k$ with distortion $(1 + \varepsilon)$, where $k = O(\varepsilon^{-2} \log n)$.

18.2 Examples

What is distortion? When considering a mapping $f : \mathcal{X} \to \mathbb{R}^d$ of a metric space $(\mathcal{X}, \mathbf{d})$ to $\mathbb{R}^d$, it would useful to observe that since $\mathbb{R}^d$ can be scaled, we can consider f to be an an expansion (i.e., no distances shrink). Furthermore, we can in fact assume that there is at least one pair of points $x, y \in \mathcal{X}$, such that $\mathbf{d}(x, y) = \|x - y\|$. As such, we have $\mathrm{dist}(f) = \max_{x,y} \frac{\|x-y\|}{\mathbf{d}(x,y)}$.



Why distortion is necessary? Consider the a graph $G = (V, E)$ with one vertex s connected to three other vertices a, b, c, where the weights on the edges are all one (i.e., G is the star graph with three leafs). We claim that G can not be embedded into Euclidean space with distortion $\leq \sqrt{2}$. Indeed, consider the associated metric space $(V, \mathbf{d}_G)$ and an (expansive) embedding $f : V \to \mathbb{R}^d$.

Consider the triangle formed by $\triangle = a'b'c'$, where $a' = f(a), b' = f(b)$ and $c' = f(c)$. Next, consider the following quantity $\max(\|a' - s'\|, \|b' - s'\|, \|c' - s'\|)$ which lower bounds the distortion of f. This quantity is minimized when $r = \|a' - s'\| = \|b' - s'\| = \|c' - s'\|$. Namely, s' is the center of the smallest enclosing circle of $\triangle$. However, r is minimize when all the edges of $\triangle$ are of equal length, and are in fact of length $\mathbf{d}_G(a, b) = 2$. It follows that $\mathrm{dist}(f) \geq r \geq 2/\sqrt{3}$.

It is known that $\Omega(\log n)$ distortion is necessary in the worst case. This is shown using expanders [Mat02].

18.2.1 Hierarchical Tree Metrics

The following metric is quite useful in practice, and nicely demonstrate why algorithmically finite metric spaces are useful.

Definition 18.2.1 *Hierarchically well-separated tree* (HST) is a metric space defined on the leaves of a rooted tree T. To each vertex $u \in T$ there is associated a label $\Delta_u \geq 0$ such that $\Delta_u = 0$ if and only if u is a leaf of T. The labels are such that if a vertex u is a child of a vertex v then $\Delta_u \leq \Delta_v$. The distance between two leaves $x, y \in T$ is defined as $\Delta_{\mathrm{lca}(x,y)}$, where $\mathrm{lca}(x, y)$ is the least common ancestor of x and y in T.

A HST T is a *k-HST* if for a vertex $v \in T$, we have that $\Delta_v \leq \Delta_{\overline{\mathrm{p}}(v)}/k$, where $\overline{\mathrm{p}}(v)$ is the parent of v in T.

Note that a HST is a very limited metric. For example, consider the cycle $G = C_n$ of n vertices, with weight one on the edges, and consider an expansive embedding f of G into a HST $\mathcal{H}$. It is easy to verify, that there must be two consecutive nodes of the cycle, which are mapped to two different subtrees of the root r of $\mathcal{H}$. Since $\mathcal{H}$ is expansive, it follows that $\Delta_r \geq n/2$. As such, $\mathrm{dist}(f) \geq n/2$. Namely, HSTs fail to faithfully represent even very simple metrics.

18.2.2 Clustering

One natural problem we might want to solve on a graph (i.e., finite metric space) $(\mathcal{X}, \mathbf{d})$ is to partition it into clusters. One such natural clustering is the *k-median clustering*, where we would like to choose a set $C \subseteq \mathcal{X}$ of k centers, such that $\nu_C(\mathcal{X}, \mathbf{d}) = \sum_{\mathsf{q} \in \mathcal{X}} \mathbf{d}(\mathsf{q}, C)$ is minimized, where $\mathbf{d}(\mathsf{q}, C) = \min_{c \in C} \mathbf{d}(\mathsf{q}, c)$ is the distance of q to its closest center in C.

It is known that finding the optimal k-median clustering in a (general weighted) graph is NP-complete. As such, the best we can hope for is an approximation algorithm. However, if the structure of the finite metric space $(\mathcal{X}, \mathbf{d})$ is simple, then the problem can be solved efficiently. For example, if the points of $\mathcal{X}$ are on the real line (and the distance between a and b is just $|a - b|$), then k-median can be solved using dynamic programming.

Another interesting case is when the metric space $(\mathcal{X}, \mathbf{d})$ is a HST. Is not too hard to prove the following lemma. See Exercise 18.7.1.

Lemma 18.2.2 *Let $(\mathcal{X}, \mathbf{d})$ be a HST defined over n points, and let $k > 0$ be an integer. One can compute the optimal k-median clustering of $\mathcal{X}$ in $O(k^2 n)$ time.*

Thus, if we can embed a general graph G into a HST $\mathcal{H}$, with low distortion, then we could approximate the k-median clustering on G by clustering the resulting HST, and "importing" the resulting partition to the original space. The quality of approximation, would be bounded by the distortion of the embedding of G into $\mathcal{H}$.

18.3 Random Partitions

Let $(\mathcal{X}, d)$ be a finite metric space. Given a partition $P = \{C_1, \ldots, C_m\}$ of $\mathcal{X}$, we refer to the sets C_i as *clusters*. We write $\mathcal{P}_{\mathcal{X}}$ for the set of all partitions of $\mathcal{X}$. For $x \in \mathcal{X}$ and a partition $P \in \mathcal{P}_{\mathcal{X}}$ we denote by $P(x)$ the unique cluster of P containing x. Finally, the set of all probability distributions on $\mathcal{P}_{\mathcal{X}}$ is denoted $\mathcal{D}_{\mathcal{X}}$.

18.3.1 Constructing the partition

Let $\Delta = 2^u$ be a prescribed parameter, which is the required diameter of the resulting clusters. Choose, uniformly at random, a permutation π of $\mathcal{X}$ and a random value $\alpha \in [1/4, 1/2]$. Let $R = \alpha\Delta$, and observe that it is uniformly distributed in the interval $[\Delta/4, \Delta/2]$.

The partition is now defined as follows: A point $x \in \mathcal{X}$ is assigned to the cluster C_y of y, where y is the first point in the permutation in distance $\leq R$ from x. Formally,

$$C_y = \left\{ x \in \mathcal{X} \;\middle|\; x \in \mathbf{b}(y, R) \text{ and } \pi(y) \leq \pi(z) \text{ for all } z \in \mathcal{X} \text{ with } x \in \mathbf{b}(z, R) \right\}.$$

Let $P = \{C_y\}_{y \in \mathcal{X}}$ denote the resulting partition.

Here is a somewhat more intuitive explanation: Once we fix the radius of the clusters R, we start scooping out balls of radius R centered at the points of the random permutation π. At the ith stage, we scoop out only the remaining mass at the ball centered at x_i of radius r, where x_i is the ith point in the random permutation.

18.3.2 Properties

Lemma 18.3.1 *Let $(\mathcal{X}, d)$ be a finite metric space, $\Delta = 2^u$ a prescribed parameter, and let P be the partition of $\mathcal{X}$ generated by the above random partition. Then the following holds:*

(i) For any $C \in P$, we have $\mathrm{diam}(C) \le \Delta$.

(ii) Let x be any point of $\mathcal{X}$, and t a parameter $\le \Delta/8$. Then,

$$\mathbf{Pr}[\mathbf{b}(x,t) \not\subseteq P(x)] \le \frac{8t}{\Delta} \ln \frac{b}{a},$$

where $a = |\mathbf{b}(x, \Delta/8)|$, $b = |\mathbf{b}(x,\Delta)|$.

Proof: Since $C_y \subseteq \mathbf{b}(y,R)$, we have that $\mathrm{diam}(C_y) \le \Delta$, and thus the first claim holds.

Let U be the set of points of $\mathbf{b}(x,\Delta)$, such that $w \in U$ iff $\mathbf{b}(w,R) \cap \mathbf{b}(x,t) \neq \emptyset$. Arrange the points of U in increasing distance from x, and let $w_1, \ldots, w_{b'}$ denote the resulting order, where $b' = |U|$. Let $I_k = [d(x,w_k) - t, d(x,w_k) + t]$ and write $\mathcal{E}_k$ for the event that w_k is the first point in π such that $\mathbf{b}(x,t) \cap C_{w_k} \neq \emptyset$, and yet $\mathbf{b}(x,t) \not\subseteq C_{w_k}$. Note that if $w_k \in \mathbf{b}(x,\Delta/8)$, then $\mathbf{Pr}[\mathcal{E}_k] = 0$ since $\mathbf{b}(x,t) \subseteq \mathbf{b}(x,\Delta/8) \subseteq \mathbf{b}(w_k,\Delta/4) \subseteq \mathbf{b}(w_k,R)$.

In particular, $w_1, \ldots, w_a \in \mathbf{b}(x,\Delta/8)$ and as such $\mathbf{Pr}[\mathcal{E}_1] = \cdots = \mathbf{Pr}[\mathcal{E}_a] = 0$. Also, note that if $\mathbf{d}(x,w_k) < R - t$ then $\mathbf{b}(w_k,R)$ contains $\mathbf{b}(x,t)$ and as such $\mathcal{E}_k$ can not happen. Similarly, if $\mathbf{d}(x,w_k) > R+t$ then $\mathbf{b}(w_k,R) \cap \mathbf{b}(x,t) = \emptyset$ and $\mathcal{E}_k$ can not happen. As such, if $\mathcal{E}_k$ happen then $R - t \le \mathbf{d}(x,w_k) \le R + t$. Namely, if $\mathcal{E}_k$ happen then $R \in I_k$. Namely, $\mathbf{Pr}[\mathcal{E}_k] = \mathbf{Pr}[\mathcal{E}_k \cap (R \in I_k)] = \mathbf{Pr}[R \in I_k] \cdot \mathbf{Pr}[\mathcal{E}_k \,|\, R \in I_k]$. Now, R is uniformly distributed in the interval $[\Delta/4, \Delta/2]$, and I_k is an interval of length $2t$. Thus, $\mathbf{Pr}[R \in I_k] \le 2t/(\Delta/4) = 8t/\Delta$.

Next, to bound $\mathbf{Pr}[\mathcal{E}_k \,|\, R \in I_k]$, we observe that $w_1, \ldots, w_{k-1}$ are closer to x than w_k and their distance to $\mathbf{b}(x,t)$ is smaller than R. Thus, if any of them appear before w_k in π then $\mathcal{E}_k$ does not happen. Thus, $\mathbf{Pr}[\mathcal{E}_k \,|\, R \in I_k]$ is bounded by the probability that w_k is the first to appear in π out of $w_1, \ldots, w_k$. But this probability is $1/k$, and thus $\mathbf{Pr}[\mathcal{E}_k \,|\, R \in I_k] \le 1/k$.

We are now ready for the kill. Indeed,

$$\begin{aligned}
\mathbf{Pr}[\mathbf{b}(x,t) \not\subseteq P(x)] &= \sum_{k=1}^{b'} \mathbf{Pr}[\mathcal{E}_k] = \sum_{k=a+1}^{b'} \mathbf{Pr}[\mathcal{E}_k] = \sum_{k=a+1}^{b'} \Pr[R \in I_k] \cdot \Pr[\mathcal{E}_k \,|\, R \in I_k] \\
&\le \sum_{k=a+1}^{b'} \frac{8t}{\Delta} \cdot \frac{1}{k} \le \frac{8t}{\Delta} \ln \frac{b'}{a} \le \frac{8t}{\Delta} \ln \frac{b}{a},
\end{aligned}$$

since $\sum_{k=a+1}^{b} \frac{1}{k} \le \int_a^b \frac{dx}{x} = \ln \frac{b}{a}$ and $b' \le b$. ■

18.4 Probabilistic embedding into trees

In this section, given n-point finite metric $(\mathcal{X}, \mathbf{d})$. we would like to embed it into a HST. As mentioned above, one can verify that for any embedding into HST, the distortion in the worst case is $\Omega(n)$. Thus, we define a randomized algorithm that embed $(\mathcal{X}, d)$ into a tree. Let T be the resulting tree, and consider two points $x, y \in \mathcal{X}$. Consider the *random variable* $\mathbf{d}_T(x,y)$. We constructed the tree T such that distances never shrink; i.e. $\mathbf{d}(x,y) \le \mathbf{d}_T(x,y)$. The *probabilistic distortion* of this embedding is $\max_{x,y} \mathbf{E}\left[\frac{\mathbf{d}_T(x,y)}{\mathbf{d}(x,y)}\right]$. Somewhat surprisingly, one can find such an embedding with logarithmic probabilistic distortion.

Theorem 18.4.1 *Given n-point metric $(\mathcal{X}, d)$ one can randomly embed it into a 2-HST with probabilistic distortion $\le 24 \ln n$.*

Proof: The construction is recursive. Let diam(P), and compute a random partition of $\mathcal{X}$ with cluster diameter diam(P)/2, using the construction of Section 18.3.1. We recursively construct a 2-HST for each cluster, and hang the resulting clusters on the root node v, which is marked by $\Delta_v = \mathrm{diam}(P)$. Clearly, the resulting tree is a 2-HST.

For a node $v \in T$, let $\mathcal{X}(v)$ be the set of points of $\mathcal{X}$ contained in the subtree of v.

For the analysis, assume diam(P) = 1, and consider two points $x, y \in \mathcal{X}$. We consider a node $v \in T$ to be in level i if $\mathrm{level}(v) = \lceil \lg \Delta_v \rceil = i$. The two points x and y correspond to two leaves in T, and let $\widehat{u}$ be the least common ancestor of x and y in t. We have $\mathbf{d}_T(x, y) \le 2^{\mathrm{level}(v)}$. Furthermore, note that along a path the levels are strictly monotonically increasing.

In fact, we are going to be conservative, and let w be the first ancestor of x, such that $\mathbf{b} = \mathbf{b}(x, \mathbf{d}(x, y))$ is not completely contained in $\mathcal{X}(u_1), \ldots, \mathcal{X}(u_m)$, where $u_1, \ldots, u_m$ are the children of w. Clearly, $\mathrm{level}(w) > \mathrm{level}(\widehat{u})$. Thus, $\mathbf{d}_T(x, y) \le 2^{\mathrm{level}(w)}$.

Consider the path σ from the root of T to x, and let $\mathcal{E}_i$ be the event that $\mathbf{b}$ is not fully contained in $\mathcal{X}(v_i)$, where v_i is the node of σ of level i (if such a node exists). Furthermore, let Y_i be the indicator variable which is 1 if $\mathcal{E}_i$ is the first to happened out of the sequence of events $\mathcal{E}_0, \mathcal{E}_{-1}, \ldots$. Clearly, $\mathbf{d}_T(x, y) \le \sum Y_i 2^i$.

Let $t = \mathbf{d}(x, y)$ and $j = \lfloor \lg \mathbf{d}(x, y) \rfloor$, and $n_i = \left|\mathbf{b}(x, 2^i)\right|$ for $i = 0, \ldots, -\infty$. We have

$$\mathbf{E}\big[\mathbf{d}_T(x, y)\big] \le \sum_{i=j}^{0} \mathbf{E}[Y_i]\, 2^i \le \sum_{i=j}^{0} 2^i\, \mathbf{Pr}\Big[\mathcal{E}_i \cap \overline{\mathcal{E}_{i-1}} \cap \overline{\mathcal{E}_{i-1}} \cdots \overline{\mathcal{E}_0}\Big] \le \sum_{i=j}^{0} 2^i \cdot \frac{8t}{2^i} \ln \frac{n_i}{n_{i-3}},$$

by Lemma 18.3.1. Thus,

$$\mathbf{E}\big[\mathbf{d}_T(x, y)\big] \le 8t \ln\left(\prod_{i=j}^{0} \frac{n_i}{n_{i-3}}\right) \le 8t \ln(n_0 \cdot n_1 \cdot n_2) \le 24 t \ln n.$$

It thus follows, that the expected distortion for x and y is $\le 24 \ln n$. ■

18.4.1 Application: approximation algorithm for k-median clustering

Let $(\mathcal{X}, \mathbf{d})$ be a n-point metric space, and let k be an integer number. We would like to compute the optimal k-median clustering. Number, find a subset $C_{\mathrm{opt}} \subseteq \mathcal{X}$, such that $\nu_{C_{\mathrm{opt}}}(\mathcal{X}, \mathbf{d})$ is minimized, see Section 18.2.2. To this end, we randomly embed $(\mathcal{X}, \mathbf{d})$ into a HST $\mathcal{H}$ using Theorem 18.4.1. Next, using Lemma 18.2.2, we compute the optimal k-median clustering of $\mathcal{H}$. Let C be the set of centers computed. We return C together with the partition of $\mathcal{X}$ it induces as the required clustering.

Theorem 18.4.2 *Let $(\mathcal{X}, \mathbf{d})$ be a n-point metric space. One can compute in polynomial time a k-median clustering of $\mathcal{X}$ which has expected price $O(\alpha \log n)$, where α is the price of the optimal k-median clustering of $(\mathcal{X}, \mathbf{d})$.*

Proof: The algorithm is described above, and the fact that its running time is polynomial can be easily be verified. To prove the bound on the quality of the clustering, for any point $p \in \mathcal{X}$, let $\overline{c}(p)$ denote the closest point in C_{opt} to p according to $\mathbf{d}$, where C_{opt} is the set of k-medians in the optimal clustering. Let C be the set of k-medians returned by the algorithm, and let $\mathcal{H}$ be the HST used by the algorithm. We have

$$\beta = \nu_C(\mathcal{X}, \mathbf{d}) \le \nu_C(\mathcal{X}, \mathbf{d}_{\mathcal{H}}) \le \nu_{C_{\mathrm{opt}}}(\mathcal{X}, \mathbf{d}_{\mathcal{H}}) \le \sum_{p \in \mathcal{X}} \mathbf{d}_{\mathcal{H}}(p, C_{\mathrm{opt}}) \le \sum_{p \in \mathcal{X}} \mathbf{d}_{\mathcal{H}}(p, \overline{c}(p)).$$

Thus, in expectation we have

$$\begin{aligned}
\mathbf{E}[\beta] &= \mathbf{E}\left[\sum_{p\in\mathcal{X}} \mathbf{d}_{\mathcal{H}}(p,\overline{c}(p))\right] = \sum_{p\in\mathcal{X}} \mathbf{E}[\mathbf{d}_{\mathcal{H}}(p,\overline{c}(p))] = \sum_{p\in\mathcal{X}} O(\mathbf{d}(p,\overline{c}(p))\log n)\\
&= O\left((\log n)\sum_{p\in\mathcal{X}} \mathbf{d}(p,\overline{c}(p))\right) = O\Big(\nu_{C_{\mathrm{opt}}}(\mathcal{X},\mathbf{d})\log n\Big),
\end{aligned}$$

by linearity of expectation and Theorem 18.4.1. ■

18.5 Embedding any metric space into Euclidean space

Lemma 18.5.1 *Let $(\mathcal{X},\mathbf{d})$ be a metric, and let $Y \subset \mathcal{X}$. Consider the mapping $f : \mathcal{X} \to \mathbb{R}$, where $f(x) = \mathbf{d}(x, Y) = \min_{y\in Y} \mathbf{d}(x, y)$. Then for any $x, y \in \mathcal{X}$, we have $|f(x) - f(y)| \leq \mathbf{d}(x, y)$. Namely f is nonexpansive.*

Proof: Indeed, let x' and y' be the closet points of Y, to x and y, respectively. Observe that $f(x) = \mathbf{d}(x, x') \leq \mathbf{d}(x, y') \leq \mathbf{d}(x, y) + \mathbf{d}(y, y') = \mathbf{d}(x, y) + f(y)$ by the triangle inequality. Thus, $f(x) - f(y) \leq \mathbf{d}(x, y)$. By symmetry, we have $f(y) - f(x) \leq \mathbf{d}(x, y)$. Thus, $|f(x) - f(y)| \leq \mathbf{d}(x, y)$. ■

18.5.1 The bounded spread case

Let $(\mathcal{X},\mathbf{d})$ be a n-point metric. The *spread* of $\mathcal{X}$, denoted by $\Phi(\mathcal{X}) = \frac{\mathrm{diam}(\mathcal{X})}{\min_{x,y\in\mathcal{X},x\neq y}\mathbf{d}(x,y)}$, is the ratio between the diameter of $\mathcal{X}$ and the distance between the closest pair of points.

Theorem 18.5.2 *Given a n-point metric $\mathcal{Y} = (\mathcal{X}, d)$, with spread Φ, one can embed it into Euclidean space $\mathbb{R}^k$ with distortion $O(\sqrt{\ln\Phi}\ln n)$, where $k = O(\ln\Phi \ln n)$.*

Proof: Assume that $\mathrm{diam}(\mathcal{Y}) = \Phi$ (i.e., the smallest distance in $\mathcal{Y}$ is 1), and let $r_i = 2^{i-2}$, for $i = 1, \ldots, \alpha$, where $\alpha = \lceil \lg \Phi \rceil$. Let $P_{i,j}$ be a random partition of P with diameter r_i, using Theorem 18.4.1, for $i = 1, \ldots, \alpha$ and $j = 1, \ldots, \beta$, where $\beta = \lceil c \log n \rceil$ and c is a large enough constant to be determined shortly.

For each cluster of $P_{i,j}$ randomly toss a coin, and let $V_{i,j}$ be the all the points of $\mathcal{X}$ that belong to clusters in $P_{i,j}$ that got 'T' in their coin toss. For a point $u \in x$, let $f_{i,j}(x) = \mathbf{d}(x, \mathcal{X} \setminus V_{i,j}) = \min_{v\in\mathcal{X}\setminus V_{i,j}} \mathbf{d}(x, v)$, for $i = 0, \ldots, m$ and $j = 1, \ldots, \beta$. Let $F : \mathcal{X} \to \mathbb{R}^{(m+1)\cdot\beta}$ be the embedding, such that $F(x) = \big(f_{0,1}(x), f_{0,2}(x), \ldots, f_{0,\beta}(x), f_{1,1}(x), f_{0,2}(x), \ldots, f_{1,\beta}(x), \ldots, f_{m,1}(x), f_{m,2}(x), \ldots, f_{m,\beta}(x)\big)$.

Next, consider two points $x, y \in \mathcal{X}$, with distance $\phi = \mathbf{d}(x, y)$. Let k be an integer such that $r_u \leq \phi/2 \leq r_{u+1}$. Clearly, in any partition of $P_{u,1}, \ldots, P_{u,\beta}$ the points x and y belong to different clusters. Furthermore, with probability half $x \in V_{u,j}$ and $y \notin V_{u,j}$ or $x \notin V_{u,j}$ and $y \in V_{u,j}$, for $1 \leq j \leq \beta$.

Let $\mathcal{E}_j$ denote the event that $\mathbf{b}(x, \rho) \subseteq V_{u,j}$ and $y \notin V_{u,j}$, for $j = 1, \ldots, \beta$, where $\rho = \phi/(64 \ln n)$. By Lemma 18.3.1, we have

$$\mathbf{Pr}\Big[\mathbf{b}(x,\rho) \not\subseteq P_{u,j}(x)\Big] \leq \frac{8\rho}{r_u}\ln n \leq \frac{\phi}{8r_u} \leq 1/2.$$

Thus,

$$\begin{aligned}
\mathbf{Pr}\Big[\mathcal{E}_j\Big] &= \mathbf{Pr}\Big[\Big(\mathbf{b}(x,\rho) \subseteq P_{u,j}(x)\Big) \cap \Big(x \in V_{u,j}\Big) \cap \Big(y \notin V_{u,j}\Big)\Big]\\
&= \mathbf{Pr}\Big[\mathbf{b}(x,\rho) \subseteq P_{u,j}(x)\Big] \cdot \mathbf{Pr}\Big[x \in V_{u,j}\Big] \cdot \mathbf{Pr}\Big[y \notin V_{u,j}\Big] \geq 1/8,
\end{aligned}$$

since those three events are independent. Notice, that if $\mathcal{E}_j$ happens, than $f_{u,j}(x) \geq \rho$ and $f_{u,j}(y) = 0$.

Let X_j be an indicator variable which is 1 if $\mathcal{E}_i$ happens, for $j = 1, \ldots, \beta$. Let $Z = \sum_j X_j$, and we have $\mu = \mathbf{E}[Z] = \mathbf{E}\Big[\sum_j X_j\Big] \geq \beta/8$. Thus, the probability that only $\beta/16$ of $\mathcal{E}_1, \ldots, \mathcal{E}_\beta$ happens, is

$\mathbf{Pr}[Z < (1 - 1/2)\,\mathbf{E}[Z]]$. By the Chernoff inequality, we have $\mathbf{Pr}[Z < (1 - 1/2)\,\mathbf{E}[Z]] \leq \exp\Bigl(-\mu 1/(2 \cdot 2^2)\Bigr) = \exp(-\beta/64) \leq 1/n^{10}$, if we set $c = 640$.

Thus, with high probability

$$\|F(x) - F(y)\| \geq \sqrt{\sum_{j=1}^{\beta}\Bigl(f_{u,j}(x) - f_{u,j}(y)\Bigr)^2} \geq \sqrt{\rho^2 \frac{\beta}{16}} = \sqrt{\beta}\frac{\rho}{4} = \phi \cdot \frac{\sqrt{\beta}}{256 \ln n}.$$

On the other hand, $\left|f_{i,j}(x) - f_{i,j}(y)\right| \leq \mathbf{d}(x, y) = \phi \leq 64\rho \ln n$. Thus,

$$\|F(x) - F(y)\| \leq \sqrt{\alpha\beta(64\rho \ln n)^2} \leq 64\sqrt{\alpha\beta}\rho \ln n = \sqrt{\alpha\beta} \cdot \phi.$$

Thus, setting $G(x) = F(x)\frac{256 \ln n}{\sqrt{\beta}}$, we get a mapping that maps two points of distance ϕ from each other to two points with distance in the range $\left[\phi, \phi \cdot \sqrt{\alpha\beta} \cdot \frac{256 \ln n}{\sqrt{\beta}}\right]$. Namely, $G(\cdot)$ is an embedding with distortion $O(\sqrt{\alpha} \ln n) = O(\sqrt{\ln \Phi} \ln n)$.

The probability that G fails on one of the pairs, is smaller than $(1/n^{10}) \cdot \binom{n}{2} < 1/n^8$. In particular, we can check the distortion of G for all $\binom{n}{2}$ pairs, and if any of them fail (i.e., the distortion is too big), we restart the process. ■

18.5.2 The unbounded spread case

Our next task, is to extend Theorem 18.5.2 to the case of unbounded spread. Indeed, let $(\mathcal{X}, d)$ be a n-point metric, such that $\operatorname{diam}(\mathcal{X}) \leq 1/2$. Again, we look on the different resolutions $r_1, r_2, \ldots$, where $r_i = 1/2^{i-1}$. For each one of those resolutions r_i, we can embed this resolution into β coordinates, as done for the bounded case. Then we concatenate the coordinates together.

There are two problems with this approach: (i) the number of resulting coordinates is infinite, and (ii) a pair x, y, might be distorted a "lot" because it contributes to all resolutions, not only to its "relevant" resolutions.

Both problems can be overcome with careful tinkering. Indeed, for a resolution r_i, we are going to modify the metric, so that it ignores short distances (i.e., distances $\leq r_i/n^2$). Formally, for each resolution r_i, let $G_i = (\mathcal{X}, \widehat{E_i})$ be the graph where two points x and y are connected if $\mathbf{d}(x, y) \leq r_i/n^2$. Consider a connected component $C \in G_i$. For any two points $x, y \in C$, we have $\mathbf{d}(x, y) \leq n(r_i/n^2) \leq r_i/n$. Let $\mathcal{X}_i$ be the set of connected components of G_i, and define the distances between two connected components $C, C' \in \mathcal{X}_i$, to be $\mathbf{d}_i(C, C') = \mathbf{d}(C, C') = \min_{c \in C, c' \in C'} \mathbf{d}(c, c')$.

It is easy to verify that $(\mathcal{X}_i, \mathbf{d}_i)$ is a metric space (see Exercise 18.7.2). Furthermore, we can naturally embed $(\mathcal{X}, \mathbf{d})$ into $(\mathcal{X}_i, \mathbf{d}_i)$ by mapping a point $x \in \mathcal{X}$ to its connected components in $\mathcal{X}_i$. Essentially $(\mathcal{X}_i, \mathbf{d}_i)$ is a snapped version of the metric $(\mathcal{X}, d)$, with the advantage that $\Phi((\mathcal{X}, \mathbf{d}_i)) = O(n^2)$. We now embed $\mathcal{X}_i$ into $\beta = O(\log n)$ coordinates. Next, for any point of $\mathcal{X}$ we embed it into those β coordinates, by using the embedding of its connected component in $\mathcal{X}_i$. Let E_i be the embedding for resolution r_i. Namely, $E_i(x) = (f_{i,1}(x), f_{i,2}(x), \ldots, f_{i,\beta}(x))$, where $f_{i,j}(x) = \min(\mathbf{d}_i(x, \mathcal{X} \setminus V_{i,j}), 2r_i)$. The resulting embedding is $F(x) = \oplus E_i(x) = (E_1(x), E_2(x), \ldots,)$.

Since we slightly modified the definition of $f_{i,j}(\cdot)$, we have to show that $f_{i,j}(\cdot)$ is nonexpansive. Indeed, consider two points $x, y \in \mathcal{X}_i$, and observe that

$$\left|f_{i,j}(x) - f_{i,j}(y)\right| \leq \left|\mathbf{d}_i(x, V_{i,j}) - \mathbf{d}_i(y, V_{i,j})\right| \leq \mathbf{d}_i(x, y) \leq \mathbf{d}(x, y),$$

as a simple case analysis[⑤] shows.

[⑤]Indeed, if $f_{i,j}(x) < \mathbf{d}_i(x, V_{i,j})$ and $f_{i,j}(y) < \mathbf{d}_i(x, V_{i,j})$ then $f_{i,j}(x) = 2r_i$ and $f_{i,j}(y) = 2r_i$, which implies the above inequality. If $f_{i,j}(x) = \mathbf{d}_i(x, V_{i,j})$ and $f_{i,j}(y) = \mathbf{d}_i(x, V_{i,j})$ then the inequality trivially holds. The other option is handled in a similar fashion.

For a pair $x, y \in \mathcal{X}$, and let $\phi = \mathbf{d}(x, y)$. To see that $F(\cdot)$ is the required embedding (up to scaling), observe that, by the same argumentation of Theorem 18.5.2, we have that with high probability

$$\|F(x) - F(y)\| \geq \phi \cdot \frac{\sqrt{\beta}}{256 \ln n}.$$

To get an upper bound on this distance, observe that for i such that $r_i > \phi n^2$, we have $E_i(x) = E_i(y)$. Thus,

$$\begin{aligned}
\|F(x) - F(y)\|^2 &= \sum_i \|E_i(x) - E_i(y)\|^2 = \sum_{i, r_i < \phi n^2} \|E_i(x) - E_i(y)\|^2 \\
&= \sum_{i, \phi/n^2 < r_i < \phi n^2} \|E_i(x) - E_i(y)\|^2 + \sum_{i, r_i < \phi/n^2} \|E_i(x) - E_i(y)\|^2 \\
&= \beta\phi^2 \lg\left(n^4\right) + \sum_{i, r_i < \phi/n^2} (2r_i)^2 \beta \leq 4\beta\phi^2 \lg n + \frac{4\phi^2\beta}{n^4} \leq 5\beta\phi^2 \lg n.
\end{aligned}$$

Thus, $\|F(x) - F(y)\| \leq \phi\sqrt{5\beta \lg n}$. We conclude, that with high probability, $F(\cdot)$ is an embedding of $\mathcal{X}$ into Euclidean space with distortion$\left(\phi\sqrt{5\beta \lg n}\right) / \left(\phi \cdot \frac{\sqrt{\beta}}{256 \ln n}\right) = O(\log^{3/2} n)$.

We still have to handle the infinite number of coordinates problem. However, the above proof shows that we care about a resolution r_i (i.e., it contributes to the estimates in the above proof) only if there is a pair x and y such that $r_i/n^2 \leq \mathbf{d}(x, y) \leq r_i n^2$. Thus, for every pair of distances there are $O(\log n)$ relevant resolutions. Thus, there are at most $\eta = O(n^2 \beta \log n) = O(n^2 \log^2 n)$ relevant coordinates, and we can ignore all the other coordinates. Next, consider the affine subspace h that spans $F(P)$. Clearly, it is $n - 1$ dimensional, and consider the projection $G : \mathbb{R}^\eta \to \mathbb{R}^{n-1}$ that projects a point to its closest point in h. Clearly, $G(F(\cdot))$ is an embedding with the same distortion for P, and the target space is of dimension $n - 1$.

Note, that all this process succeeds with high probability. If it fails, we try again. We conclude:

Theorem 18.5.3 (Low quality Bourgain theorem.) *Given a n-point metric M, one can embed it into Euclidean space of dimension $n - 1$, such that the distortion of the embedding is at most $O(\log^{3/2} n)$.*

Using the Johnson-Lindenstrauss lemma, the dimension can be further reduced to $O(\log n)$. In fact, being more careful in the proof, it is possible to reduce the dimension to $O(\log n)$ directly.

18.6 Bibliographical notes

The partitions we use are due to Calinescu *et al.* [CKR01]. The idea of embedding into spanning trees is due to Alon *et al.* [AKPW95], which showed that one can get a probabilistic distortion of $2^{O\left(\sqrt{\log n \log\log n}\right)}$. Yair Bartal realized that by allowing trees with additional vertices, one can get a considerably better result. In particular, he showed [Bar96] that probabilistic embedding into trees can be done with polylogarithmic average distortion. He later improved the distortion to $O(\log n \log\log n)$ in [Bar98]. Improving this result was an open question, culminating in the work of Fakcharoenphol *et al.* [FRT03] which achieve the optimal $O(\log n)$ distortion.

Interestingly, if one does not care about the optimal distortion, one can get similar result (for embedding into probabilistic trees), by first embedding the metric into Euclidean space, then reduce the dimension by the Johnson-Lindenstrauss lemma, and finally, construct an HST by constructing a quadtree over the points. The "trick" is to randomly translate the quadtree. It is easy to verify that this yields $O(\log^4 n)$ distortion. See the survey by Indyk [Ind01] for more details. This random shifting of quadtrees is a powerful technique that was used in getting several result, and it is a crucial ingredient in Arora [Aro98] approximation algorithm for Euclidean TSP.

Our proof of Lemma 18.3.1 (which is originally from [FRT03]) is taken from [KLMN04]. The proof of Theorem 18.5.3 is by Gupta [Gup00].

A good exposition of metric spaces is available in Matoušek [Mat02].

18.7 Exercises

Exercise 18.7.1 (Clustering for HST.) Let $(\mathcal{X}, \mathbf{d})$ be a HST defined over n points, and let $k > 0$ be an integer. Provide an algorithm that computes the optimal k-median clustering of $\mathcal{X}$ in $O(k^2 n)$ time.

[**Hint:** Transform the HST into a tree where every node has only two children. Next, run a dynamic programming algorithm on this tree.**]**

Exercise 18.7.2 (Partition induced metric.)

(a) Give a counter example to the following claim: Let $(\mathcal{X}, \mathbf{d})$ be a metric space, and let P be a partition of $\mathcal{X}$. Then, the pair $(P, \mathbf{d}')$ is a metric, where $\mathbf{d}'(C, C') = \mathbf{d}(C, C') = \min_{x \in C, y \in C'} \mathbf{d}(x, y)$ and $C, C' \in P$.

(b) Let $(\mathcal{X}, \mathbf{d})$ be a n-point metric space, and consider the set $U = \left\{ i \,\middle|\, 2^i \leq \mathbf{d}(x, y) \leq 2^{i+1}, \text{ for } x, y \in \mathcal{X} \right\}$. Prove that $|U| = O(n)$. Namely, there are only n different resolutions that "matter" for a finite metric space.

Exercise 18.7.3 (Computing the diameter via embeddings.)

(a) (h:1) Let ℓ be a line in the plane, and consider the embedding $f : \mathbb{R}^2 \to \ell$, which is the projection of the plane into ℓ. Prove that f is 1-Lipschitz, but it is not K-bi-Lipschitz for any constant K.

(b) (h:3) Prove that one can find a family of projections $\mathcal{F}$ of size $O(1/\sqrt{\varepsilon})$, such that for any two points $x, y \in \mathbb{R}^2$, for one of the projections $f \in \mathcal{F}$ we have $\mathbf{d}(f(x), f(y)) \geq (1 - \varepsilon)\mathbf{d}(x, y)$.

(c) (h:1) Given a set P of n in the plane, given a $O(n/\sqrt{\varepsilon})$ time algorithm that outputs two points $x, y \in P$, such that $\mathbf{d}(x, y) \geq (1 - \varepsilon)\mathrm{diam}(P)$, where $\mathrm{diam}(P) = \max_{z,w \in P} \mathbf{d}(z, w)$ is the diameter of P.

(d) (h:2) Given P, show how to extract, in $O(n)$ time, a set $Q \subseteq P$ of size $O(\varepsilon^{-2})$, such that $\mathrm{diam}(Q) \geq (1 - \varepsilon/2)\mathrm{diam}(P)$. (Hint: Construct a grid of appropriate resolution.)

In particular, give an $(1 - \varepsilon)$-approximation algorithm to the diameter of P that works in $O(n + \varepsilon^{-2.5})$ time. (There are slightly faster approximation algorithms known for approximating the diameter.)

Acknowledgments

The presentation in this write-up follows closely the insightful suggestions of Manor Mendel.

Chapter 19

Tail Inequalities

"Wir müssen wissen, wir werden wissen" (We must know, we shall know)
—— David Hilbert

19.1 Markov Inequality

Theorem 19.1.1 (Markov Inequality) *For a non-negative variable X, and $t > 0$, we have:*

$$\mathbf{Pr}[X \geq t] \leq \frac{\mathbf{E}[X]}{t}.$$

Proof: Assume that this is false, and there exists $t_0 > 0$ such that $\mathbf{Pr}[X \geq t_0] > \frac{\mathbf{E}[X]}{t_0}$. However,

$$\begin{aligned}\mathbf{E}[X] &= \sum_x x \cdot \mathbf{Pr}[X = x] = \sum_{x<t_0} x \cdot \mathbf{Pr}[X = x] + \sum_{x \geq t_0} x \cdot \mathbf{Pr}[X = x] \\ &\geq 0 + t_0 \cdot \mathbf{Pr}[X \geq t_0] > 0 + t_0 \cdot \frac{\mathbf{E}[X]}{t_0} = \mathbf{E}[X],\end{aligned}$$

a contradiction. ■

Theorem 19.1.2 (Chebychev inequality) *Let X be a random variable with $\mu_x = \mathbf{E}[X]$ and σ_x be the standard deviation of X. That is $\sigma_X^2 = \mathbf{E}\left[(X - \mu_x)^2\right]$. Then, $\mathbf{Pr}[|X - \mu_X| \geq t\sigma_X] \leq \frac{1}{t^2}$.*

Proof: Note that

$$\mathbf{Pr}[|X - \mu_X| \geq t\sigma_X] = \mathbf{Pr}\left[(X - \mu_X)^2 \geq t^2\sigma_X^2\right].$$

Set $Y = (X - \mu_X)^2$. Clearly, $E\left[Y\right] = \sigma_X^2$. Now, apply Markov inequality to Y. ■

Definition 19.1.3 Variables X, Y are *independent* if for any x, y we have:

$$\mathbf{Pr}[(X = x) \cap (Y = y)] = \mathbf{Pr}[X = x] \cdot \mathbf{Pr}[Y = y].$$

The following is easy to verify:

Claim 19.1.4 *If X and Y are independent, then $\mathbf{E}[XY] = \mathbf{E}[X]\,\mathbf{E}[Y]$.*
If X and Y are independent then $Z = e^X$, $W = e^Y$ are also independent variables.

19.2 Tail Inequalities

19.2.1 The Chernoff Bound — Special Case

Theorem 19.2.1 *Let $X_1, \ldots, X_n$ be n independent random variables, such that* $\mathbf{Pr}[X_i = 1] = \mathbf{Pr}[X_i = -1] = \frac{1}{2}$*, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^{n} X_i$. Then, for any $\Delta > 0$, we have*

$$\mathbf{Pr}[Y \geq \Delta] \leq e^{-\Delta^2/2n}.$$

Proof: Clearly, for an arbitrary t, to specified shortly, we have

$$\mathbf{Pr}[Y \geq \Delta] = \mathbf{Pr}[\exp(tY) \geq \exp(t\Delta)] \leq \frac{\mathbf{E}[\exp(tY)]}{\exp(t\Delta)},$$

the first part follows by the fact that $\exp(\cdot)$ preserve ordering, and the second part follows by the Markov inequality.

Observe that

$$\begin{aligned}
\mathbf{E}[\exp(tX_i)] &= \frac{1}{2}e^t + \frac{1}{2}e^{-t} = \frac{e^t + e^{-t}}{2} \\
&= \frac{1}{2}\left(1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots\right) \\
&+ \frac{1}{2}\left(1 - \frac{t}{1!} + \frac{t^2}{2!} - \frac{t^3}{3!} + \cdots\right) \\
&= \left(1 + \quad + \frac{t^2}{2!} + \quad + \cdots + \frac{t^{2k}}{(2k)!} + \cdots\right),
\end{aligned}$$

by the Taylor expansion of $\exp(\cdot)$. Note, that $(2k)! \geq (k!)2^k$, and thus

$$\mathbf{E}[\exp(tX_i)] = \sum_{i=0}^{\infty} \frac{t^{2i}}{(2i)!} \leq \sum_{i=0}^{\infty} \frac{t^{2i}}{2^i(i!)} = \sum_{i=0}^{\infty} \frac{1}{i!}\left(\frac{t^2}{2}\right)^i = \exp\left(t^2/2\right),$$

again, by the Taylor expansion of $\exp(\cdot)$. Next, by the independence of the X_is, we have

$$\begin{aligned}
\mathbf{E}[\exp(tY)] &= \mathbf{E}\left[\exp\left(\sum_i tX_i\right)\right] = \mathbf{E}\left[\prod_i \exp(tX_i)\right] = \prod_{i=1}^{n} \mathbf{E}[\exp(tX_i)] \\
&\leq \prod_{i=1}^{n} e^{t^2/2} = e^{nt^2/2}.
\end{aligned}$$

We have

$$\mathbf{Pr}[Y \geq \Delta] \leq \frac{\exp\left(nt^2/2\right)}{\exp(t\Delta)} = \exp\left(nt^2/2 - t\Delta\right).$$

Next, by minimizing the above quantity for t, we set $t = \Delta/n$. We conclude,

$$\mathbf{Pr}[Y \geq \Delta] \leq \exp\left(\frac{n}{2}\left(\frac{\Delta}{n}\right)^2 - \frac{\Delta}{n}\Delta\right) = \exp\left(-\frac{\Delta^2}{2n}\right).$$

■

By the symmetry of Y, we get the following:

Corollary 19.2.2 *Let $X_1, \ldots, X_n$ be n independent random variables, such that* $\mathbf{Pr}[X_i = 1] = \mathbf{Pr}[X_i = -1] = \frac{1}{2}$*, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^{n} X_i$. Then, for any $\Delta > 0$, we have*

$$\mathbf{Pr}[|Y| \geq \Delta] \leq 2e^{-\Delta^2/2n}.$$

Corollary 19.2.3 *Let $X_1, \ldots, X_n$ be n independent* coin flips, *such that* $\mathbf{Pr}[X_i = 0] = \mathbf{Pr}[X_i = 1] = \frac{1}{2}$*, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^{n} X_i$. Then, for any $\Delta > 0$, we have*

$$\mathbf{Pr}\left[\left|Y - \frac{n}{2}\right| \geq \Delta\right] \leq 2e^{-2\Delta^2/n}.$$

Remark 19.2.4 Before going any further, it is might be instrumental to understand what this inequalities imply. Consider then case where X_i is either zero or one with probability half. In this case $\mu = \mathbf{E}[Y] = n/2$. Set $\delta = t\sqrt{n}$ ($\sqrt{\mu}$ is approximately the standard deviation of X if $p_i = 1/2$). We have by

$$\mathbf{Pr}\left[\left|Y - \frac{n}{2}\right| \geq \Delta\right] \leq 2\exp\left(-2\Delta^2/n\right) = 2\exp\left(-2(t\sqrt{n})^2/n\right) = 2\exp\left(-2t^2\right).$$

Thus, Chernoff inequality implies exponential decay (i.e., $\leq 2^{-t}$) with t standard deviations, instead of just polynomial (like the Chebychev's inequality).

19.2.2 The Chernoff Bound — General Case

Here we present the Chernoff bound in a more general settings.

Question 19.2.5 *Let*

1. *$X_1, \ldots, X_n$ - n independent Bernoulli trials, where*

   $$\mathbf{Pr}[X_i = 1] = p_i, \text{ and } \mathbf{Pr}[X_i = 0] = q_i = 1 - p_i.$$

   Each X_i is known as a Poisson trials.

2. *$X = \sum_{i=1}^{b} X_i$. $\mu = \mathbf{E}[X] = \sum_i p_i$.*

   Question*: Probability that $X > (1 + \delta)\mu$?*

Theorem 19.2.6 *For any $\delta > 0$, we have* $\mathbf{Pr}[X > (1 + \delta)\mu] < \left(\dfrac{e^\delta}{(1 + \delta)^{1+\delta}}\right)^\mu$.

Or in a more simplified form, for any $\delta \leq 2e - 1$,

$$\mathbf{Pr}[X > (1 + \delta)\mu] < \exp\left(-\mu\delta^2/4\right), \tag{19.1}$$

and

$$\mathbf{Pr}[X > (1 + \delta)\mu] < 2^{-\mu(1+\delta)}, \tag{19.2}$$

for $\delta \geq 2e - 1$.

Proof: We have $\mathbf{Pr}[X > (1+\delta)\mu] = \mathbf{Pr}\left[e^{tX} > e^{t(1+\delta)\mu}\right]$. By the Markov inequality, we have:

$$\mathbf{Pr}[X > (1+\delta)\mu] < \frac{\mathbf{E}\left[e^{tX}\right]}{e^{t(1+\delta)\mu}}$$

On the other hand,

$$\mathbf{E}\left[e^{tX}\right] = \mathbf{E}\left[e^{t(X_1+X_2\ldots+X_n)}\right] = \mathbf{E}\left[e^{tX_1}\right]\cdots\mathbf{E}\left[e^{tX_n}\right].$$

Namely,

$$\mathbf{Pr}[X > (1+\delta)\mu] < \frac{\prod_{i=1}^n \mathbf{E}\left[e^{tX_i}\right]}{e^{t(1+\delta)\mu}} = \frac{\prod_{i=1}^n\left((1-p_i)e^0 + p_i e^t\right)}{e^{t(1+\delta)\mu}} = \frac{\prod_{i=1}^n(1+p_i(e^t-1))}{e^{t(1+\delta)\mu}}.$$

Let $y = p_i(e^t - 1)$. We know that $1 + y < e^y$ (since $y > 0$). Thus,

$$\begin{aligned}\mathbf{Pr}[X > (1+\delta)\mu] &< \frac{\prod_{i=1}^n \exp(p_i(e^t-1))}{e^{t(1+\delta)\mu}}. = \frac{\exp\left(\sum_{i=1}^n p_i(e^t-1)\right)}{e^{t(1+\delta)\mu}} \\ &= \frac{\exp\left((e^t-1)\sum_{i=1}^n p_i\right)}{e^{t(1+\delta)\mu}} = \frac{\exp((e^t-1)\mu)}{e^{t(1+\delta)\mu}} = \left(\frac{\exp(e^t-1)}{e^{t(1+\delta)}}\right)^{\mu} \\ &= \left(\frac{\exp(\delta)}{(1+\delta)^{(1+\delta)}}\right)^{\mu},\end{aligned}$$

if we set $t = \log(1+\delta)$.

For the proof of the simplified form, see Section 19.2.3. ■

Definition 19.2.7 $F^+(\mu,\delta) = \left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right]^{\mu}$.

Example 19.2.8 Arkansas Aardvarks win a game with probability $1/3$. What is their probability to have a winning season with n games. By Chernoff inequality, this probability is smaller than

$$F^+(n/3, 1/2) = \left[\frac{e^{1/2}}{1.5^{1.5}}\right]^{n/3} = (0.89745)^{n/3} = 0.964577^n.$$

For $n = 40$, this probability is smaller than 0.236307. For $n = 100$ this is less than 0.027145. For $n = 1000$, this is smaller than $2.17221 \cdot 10^{-16}$ (which is pretty slim and shady). Namely, as the number of experiments is increases, the distribution converges to its expectation, and this converge is exponential.

Theorem 19.2.9 *Under the same assumptions as Theorem 19.2.6, we have:*

$$\mathbf{Pr}[X < (1-\delta)\mu] < e^{-\mu\delta^2/2}.$$

Definition 19.2.10 $F^-(\mu,\delta) = e^{-\mu\delta^2/2}$.

$\Delta^-(\mu,\varepsilon)$ - what should be the value of δ, so that the probability is smaller than ε.

$$\Delta^-(\mu,\varepsilon) = \sqrt{\frac{2\log 1/\varepsilon}{\mu}}$$

For large δ:

$$\Delta^+(\mu,\varepsilon) < \frac{\log_2(1/\varepsilon)}{\mu} - 1$$

Values	Probabilities	Inequality	Ref
$-1,+1$	$\mathbf{Pr}[X_i = -1] =$ $\mathbf{Pr}[X_i = 1] = \frac{1}{2}$	$\mathbf{Pr}[Y \geq \Delta] \leq e^{-\Delta^2/2n}$	Theorem 19.2.1
		$\mathbf{Pr}[Y \leq -\Delta] \leq e^{-\Delta^2/2n}$	Theorem 19.2.1
		$\mathbf{Pr}[\|Y\| \geq \Delta] \leq 2e^{-\Delta^2/2n}$	Corollary 19.2.2
$0,1$	$\mathbf{Pr}[X_i = 0] =$ $\mathbf{Pr}[X_i = 1] = \frac{1}{2}$	$\mathbf{Pr}\left[\left\|Y - \frac{n}{2}\right\| \geq \Delta\right] \leq 2e^{-2\Delta^2/n}$	Corollary 19.2.3
0,1	$\mathbf{Pr}[X_i = 0] = 1 - p_i$ $\mathbf{Pr}[X_i = 1] = p_i$	$\mathbf{Pr}[Y > (1+\delta)\mu] < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$	Theorem 19.2.6
	For $\delta \leq 2e - 1$	$\mathbf{Pr}[Y > (1+\delta)\mu] < \exp\left(-\mu\delta^2/4\right)$	Theorem 19.2.6
	$\delta \geq 2e - 1$	$\mathbf{Pr}[Y > (1+\delta)\mu] < 2^{-\mu(1+\delta)}$	
	For $\delta \geq 0$	$\mathbf{Pr}[Y < (1-\delta)\mu] < \exp\left(-\mu\delta^2/2\right)$	Theorem 19.2.9

Table 19.1: Summary of Chernoff type inequalities covered. Here we have n variables $X_1, \ldots, X_n$, $Y = \sum_i X_i$ and $\mu = \mathbf{E}[Y]$.

19.2.3 A More Convenient Form

Proof: **(of simplified form of Theorem 19.2.6)** Eq. (19.2) is just Exercise 19.4.1. As for Eq. (19.1), we prove this only for $\delta \leq 1/2$. For details about the case $1/2 \leq \delta \leq 2e - 1$, see [MR95]. By Theorem 19.2.6, we have

$$\mathbf{Pr}[X > (1+\delta)\mu] < \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu = \exp(\mu\delta - \mu(1+\delta)\ln(1+\delta)).$$

The Taylor expansion of $\ln(1+\delta)$ is

$$\delta - \frac{\delta^2}{2} + \frac{\delta^3}{3} - \frac{\delta^4}{4} + \cdot \geq \delta - \frac{\delta^2}{2},$$

for $\delta \leq 1$. Thus,

$$\begin{aligned}\mathbf{Pr}[X > (1+\delta)\mu] &< \exp\left(\mu\left(\delta - (1+\delta)\left(\delta - \delta^2/2\right)\right)\right) = \exp\left(\mu\left(\delta - \delta + \delta^2/2 - \delta^2 + \delta^3/2\right)\right)\\ &\leq \exp\left(\mu\left(-\delta^2/2 + \delta^3/2\right)\right) \leq \exp\left(-\mu\delta^2/4\right),\end{aligned}$$

for $\delta \leq 1/2$. ■

19.3 Bibliographical notes

The exposition here follows more or less the exposition in [MR95]. The special symmetric case (Theorem 19.2.1) is taken from [Cha01], although the proof is only very slightly simpler than the generalized form, it does yield a slightly better constant, and it would be useful when discussing discrepancy.

An orderly treatment of probability is outside the scope of our discussion. The standard text on the topic is the book by Feller [Fel91]. A more accessible text might be any introductory undergrad text on probability, in particular [MN98] has a nice chapter on the topic.

Exercise 19.4.2 (without the hint) is from [Mat99].

19.4 Exercises

Exercise 19.4.1 [2 Points] Prove that for $\delta > 2e - 1$, we have

$$F^+(\mu, \delta) < \left[\frac{e}{1+\delta}\right]^{(1+\delta)\mu} \leq 2^{-(1+\delta)\mu}.$$

Exercise 19.4.2 [10 Points] Let $S = \sum_{i=1}^{n} S_i$ be a sum of n independent random variables each attaining values $+1$ and -1 with equal probability. Let $P(n, \Delta) = \mathbf{Pr}[S > \Delta]$. Prove that for $\Delta \leq n/C$,

$$P(n, \Delta) \geq \frac{1}{C} \exp\left(-\frac{\Delta^2}{Cn}\right),$$

where C is a suitable constant. That is, the well-known Chernoff bound $P(n, \Delta) \leq \exp(-\Delta^2/2n))$ is close to the truth.

[**Hint:** Use Stirling's formula. There is also an elementary solution, using estimates for the middle binomial coffieicnets [MN98, pages 83–84], but this solution is considerably more involved and yields unfriendly constants.**]**

Exercise 19.4.3 (Tail inequality for geometric variables.) Let $X_1, \ldots, X_m$ be m independent random variables with geometric distribution with probability p (i.e., $\mathbf{Pr}[X_i = j] = (1-p)^{j-1}p$). Let $Y = \sum_i X_i$, and let $\mu = \mathbf{E}[Y] = m/p$. Prove that

$$\mathbf{Pr}[Y \geq (1+\delta)\mu] \leq \exp\left(-\frac{m\delta^2}{8}\right).$$

Bibliography

[AB99] M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge, 1999.

[Ach01] D. Achlioptas. Database-friendly random projections. In *Proc. 20th ACM Sympos. Principles Database Syst.*, pages 274–281, 2001.

[AEIS99] A. Amir, A. Efrat, P. Indyk, and H. Samet. Efficient algorithms and regular data structures for dilation, location and proximity problems. In *Proc. 40th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 160–170, 1999.

[AHV04] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. Assoc. Comput. Mach.*, 51(4):606–635, 2004.

[AKPW95] N. Alon, R. M. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the *k*-server problem. *SIAM J. Comput.*, 24(1):78–100, February 1995.

[AM94] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. *Discrete Comput. Geom.*, 11:393–418, 1994.

[AM98] S. Arya and D. Mount. ANN: library for approximate nearest neighbor searching. `http://www.cs.umd.edu/~mount/ANN/`, 1998.

[AM02] S. Arya and T. Malamatos. Linear-size approximate Voronoi diagrams. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 147–155, 2002.

[AM04] S. Arya and D. M. Mount. Computational geometry: Proximity and location. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, chapter 63. CRC Press LLC, Boca Raton, FL, 2004. to appear.

[AMM02] S. Arya, T. Malamatos, and D. M. Mount. Space-efficient approximate Voronoi diagrams. In *Proc. 34th Annu. ACM Sympos. Theory Comput.*, pages 721–730, 2002.

[AMN$^+$98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Mach.*, 45(6), 1998.

[APV02] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for *k*-line center. In *Proc. 10th Annu. European Sympos. Algorithms*, pages 54–63, 2002.

[Aro98] S. Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. *J. Assoc. Comput. Mach.*, 45(5):753–782, Sep 1998.

[AS00] N. Alon and J. H. Spencer. *The probabilistic method*. Wiley Inter-Science, 2nd edition, 2000.

[Aur91] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23:345–405, 1991.

[Bal97] K. Ball. An elementary introduction to modern convex geometry. In *Flavors of geometry*, volume MSRI Publ. 31. Cambridge Univ. Press, 1997. `http://www.msri.org/publications/books/Book31/files/ball.pdf`.

[Bar96] Y. Bartal. Probabilistic approximations of metric space and its algorithmic application. In *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 183–193, October 1996.

[Bar98] Y. Bartal. On approximating arbitrary metrices by tree metrics. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 161–168. ACM Press, 1998.

[Bar02] A. Barvinok. *A course in convexity*, volume 54 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2002.

[BEG94] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comput. Syst. Sci.*, 48:384–409, 1994.

[BH01] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *J. Algorithms*, 38:91–109, 2001.

[BHI02] M. Bădoiu, S. Har-Peled, and P. Indyk. Approximate clustering via coresets. In *Proc. 34th Annu. ACM Sympos. Theory Comput.*, pages 250–257, 2002.

[BM58] G. E.P. Box and M. E. Muller. A note on the generation of random normal deviates. *Annl. Math. Stat.*, 28:610–611, 1958.

[Cha98] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete Comput. Geom.*, 20:359–373, 1998.

[Cha01] B. Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cambridge University Press, New York, 2001.

[Cha02] T. M. Chan. Closest-point problems simplified on the ram. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 472–473. Society for Industrial and Applied Mathematics, 2002.

[Cha04] T. M. Chan. Faster coreset constructions and data stream algorithms in fixed dimensions. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 152–159, 2004.

[CK95] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to *k*-nearest-neighbors and *n*-body potential fields. *J. Assoc. Comput. Mach.*, 42:67–90, 1995.

[CKR01] G. Calinescu, H. Karloff, and Y. Rabani. Approximation algorithms for the 0-extension problem. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 8–16. Society for Industrial and Applied Mathematics, 2001.

[Cla83] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 226–232, 1983.

[Cla93] K. L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3th Workshop Algorithms Data Struct.*, volume 709 of *Lect. Notes in Comp. Sci.*, pages 246–252. Springer-Verlag, 1993.

[Cla94] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 160–164, 1994.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press / McGraw-Hill, Cambridge, Mass., 2001.

[CS00] N. Cristianini and J. Shaw-Taylor. *Support Vector Machines*. Cambridge Press, 2000.

[dBvKOS00] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[Dey98] T. K. Dey. Improved bounds for planar *k*-sets and related problems. *Discrete Comput. Geom.*, 19(3):373–382, 1998.

[DG99] S. Dasgupta and A. Gupta. An elementary proof of the Johnson-Lindenstrauss lemma. Technical Report TR-99-006, International computer science institute, 1999.

[DNIM04] M. Datar, Immorlica N, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 253–262, 2004.

[Dud74] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approx. Theory*, 10(3):227–236, 1974.

[Dun99] C. A. Duncan. *Balanced Aspect Ratio Trees*. Ph.D. thesis, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, 1999.

[EGS05] D. Eppstein, M. T. Goodrich, and J. Z. Sun. The skip quadtree: a simple dynamic data structure for multidimensional data. In *Proc. 21st Annu. ACM Sympos. Comput. Geom.*, pages 296–305. ACM, June 2005.

[EK89] O. Egecioglu and B. Kalantari. Approximating the diameter of a set of points in the Euclidean space. *Inform. Process. Lett.*, 32:205–211, 1989.

[Fel71] W. Feller. *An Introduction to Probability Theory and its Applications*, volume II. John Wiley & Sons, NY, 1971.

[Fel91] W. Feller. *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, NY, 1991.

[FG88] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444, 1988.

[FH06] J. Fischer and S. Har-Peled. On coresets for clustering and related problems. manuscript, 2006.

[FM88] P. Frankl and H. Maehara. The johnson-lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory B*, 44:355–362, 1988.

[For97] S. Fortune. Voronoi diagrams and Delaunay triangulations. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 20. CRC Press LLC, Boca Raton, FL, 1997.

[FRT03] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proc. 35th Annu. ACM Sympos. Theory Comput.*, pages 448–455, 2003.

[Gar02] R. J. Gardner. The brunn-minkowski inequality. *Bull. Amer. Math. Soc.*, 39:355–405, 2002.

[GK92] P. Gritzmann and V. Klee. Inner and outer j-radii of convex bodies in finite-dimensional normed spaces. *Discrete Comput. Geom.*, 7:255–280, 1992.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin Heidelberg, 2nd edition, 1988. 2nd edition 1994.

[Gon85] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[GRSS95] M. Golin, R. Raman, C. Schwarz, and M. Smid. Simple randomized algorithms for closest pair problems. *Nordic J. Comput.*, 2:3–27, 1995.

[Gup00] A. Gupta. *Embeddings of Finite Metrics*. PhD thesis, University of California, Berkeley, 2000.

[Har01] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 94–103, 2001.

[HM03] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing disc. In *Proc. 11th Annu. European Sympos. Algorithms*, volume 2832 of *Lect. Notes in Comp. Sci.*, pages 278–288. Springer-Verlag, 2003.

[HM04] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-median clustering and their applications. In *Proc. 36th Annu. ACM Sympos. Theory Comput.*, pages 291–300, 2004.

[HÜ05] S. Har-Peled and A. Üngör. A time-optimal delaunay refinement algorithm in two dimensions. In *Proc. 21st Annu. ACM Sympos. Comput. Geom.*, pages 228–236, 2005.

[HW87] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.

[Ind01] P. Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 10–31, 2001. Tutorial.

[Ind04] P. Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 39, pages 877–892. CRC Press LLC, Boca Raton, FL, 2nd edition, 2004.

[JL84] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mapping into hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[Joh48] F. John. Extremum problems with inequalities as subsidary conditions. *Courant Anniversary*, pages 187–204, 1948.

[KLMN04] R. Krauthgamer, J. R. Lee, M. Mendel, and A. Naor. Measured descent: A new embedding method for finite metric spaces. In *Proc. 45th Annu. IEEE Sympos. Found. Comput. Sci.*, page to appear, 2004.

[KOR00] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 2(30):457–474, 2000.

[Leo98] S. J. Leon. *Linear Algebra with Applications*. Prentice Hall, 5th edition, 1998.

[Mac50] A.M. Macbeath. A compactness theorem for affine equivalence-classes of convex regions. *Canad. J. Math*, 3:54–61, 1950.

[Mag01] A. Magen. Dimensionality reductions that preserve volumes and distance to affine spaces, and its algorithmic applications. Submitted to STOC 2002, 2001.

[Mat90] J. Matoušek. Bi-lipschitz embeddings into low-dimensional euclidean spaces. *Comment. Math. Univ. Carolinae*, 31:589–600, 1990.

[Mat92] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[Mat95] J. Matoušek. On enclosing *k* points by a circle. *Inform. Process. Lett.*, 53:217–221, 1995.

[Mat99] J. Matoušek. *Geometric Discrepancy*. Springer, 1999.

[Mat02] J. Matoušek. *Lectures on Discrete Geometry*. Springer, 2002.

[Mil04] G. L. Miller. A time efficient Delaunay refinement algorithm. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 400–409, 2004.

[MN98] J. Matoušek and J. Nešetřil. *Invitation to Discrete Mathematics*. Oxford Univ Pr, 1998.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, 1995.

[O'R85] J. O'Rourke. Finding minimal enclosing boxes. *Internat. J. Comput. Inform. Sci.*, 14:183–199, 1985.

[OvL81] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[PS89] D. Peleg and A. Schäffer. Graph spanners. *J. Graph Theory*, 13:99–116, 1989.

[Rab76] M. O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, New York, NY, 1976.

[Rup93] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 83–92, 1993.

[Sam89] H. Samet. *Spatial Data Structures: Quadtrees, Octrees, and Other Hierarchical Methods*. Addison-Wesley, Reading, MA, 1989.

[Smi00] M. Smid. Closest-point problems in computational geometry. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier Science Publishers B. V. North-Holland, Amsterdam, 2000.

[SSS02] Y. Sabharwal, N. Sharma, and S. Sen. Improved reductions of nearest neighbors search to plebs with applications to linear-sized approximate voronoi decopositions. In *Proc. 22nd Conf. Found. Soft. Tech. Theoret. Comput. Sci.*, pages 311–323, 2002.

[Tou83] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON '83*, pages A10.02/1–4, 1983.

[Üng04] A. Üngör. Off-centers: A new type of steiner points for computing size-optimal quality-guaranteed delaunay triangulations. In *Latin Amer. Theo. Inf. Symp.*, pages 152–161, 2004.

[Vai86] P. M. Vaidya. An optimal algorithm for the all-nearest-neighbors problem. In *Proc. 27th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 117–122, 1986.

[Van97] R. J. Vanderbei. *Linear programming: Foundations and extensions*. Kluwer, 1997.

[VC71] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

[WVTP97] M. Waldvogel, G. Varghese, J. Turener, and B. Plattner. Scalable high speed ip routing lookups. In *Proc. ACM SIGCOMM 97*, Octeber 1997.

[YAPV04] H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan. Practical methods for shape fitting and kinetic data structures using core sets. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 263–272, 2004.

Index