Computational geometry: notes 3*

Geometric range searching: the problem

Let $\mathcal{R}$ be a family of subsets of $\mathbb{R}^d$. The elements of $\mathcal{R}$ are called *ranges*. For example, $\mathcal{R}$ might consist of all axis-parallel boxes, or all simplices, or all halfspaces. We are given a set of n points $P \subset \mathbb{R}^d$. The basic goal of the range searching is to have an algorithm that answers queries about points of P that fall into a given $R \in \mathcal{R}$. The typical queries are

Query type	Question answered
Emptiness	Is there are single point of P in R?
Counting	How many points of P are in R?
Reporting	What are the points of P that are in R?

Other kinds of queries are also possible. For example, the points might be weighted, and the goal is to find the total weight of points in R, or the maximal weight of a point in R.

The trivial algorithm to answer any of these queries takes $\Omega(n)$ steps, as it inspects every points of P. Moreover, it is evident that if the set P is given as the part of the input, then one must read every point of P, and thus spend $\Omega(n)$. However, one usually can answer queries faster if P is fixed in advance. In that case, it is possible to do *preprocessing* of P.

For example, suppose $d = 1$, and $\mathcal{R}$ is the family of all intervals. Then by sorting P we can locate the endpoints of the interval R inside P in $O(\log n)$ steps. Then the emptiness queries can be answered immediately, and so is the weight counting queries if we store the total weight of the entries to the left of a given entry. The reporting queries can be answered in further $O(k)$ time, for the total of $O(\log n + k)$, where k is the number of points that happen to fall into R.

The range searching is a common problem in computational geometry. Its most obvious occurrence is in databases. For example, a query to the population database might ask how many people falling in the given age interval live within 100 km from a given point, which is a query on $\mathbb{R}^2 \times \mathbb{R}_+$ with the range R being the cylinder[1]. More commonly the range searching appears as a subproblem in other problems in computational geometry. For example, suppose we are given

*These notes are from `http://www.borisbukh.org/CompGeomEaster11/notes3.pdf`.

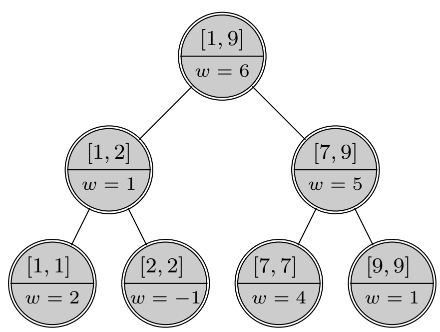
[1] We assume that the Earth is isometric to $\mathbb{R}^2$.

a set of n points P, and we wish to find all the pairs of points of P that are distance at most 1 from one another. It turns out that by preprocessing the points of P, so that one can quickly answer reporting queries for disks, one can achieve the running time of $O(n^{2-\varepsilon} + k)$.

The different kinds of queries can be treated uniformly by introducing a commutative semigroup $(S, +)$, and assigning to each point $p \in P$ a "weight" of $w(p) \in S$. The query then asks for the value of $\sum_{p \in \mathcal{R}} w(p)$. For example, for the weighted counting query the semigroup is $(\mathbb{R}, +)$, for the maximum weight query the semigroup is $(\mathbb{R}, \max)$, and for the reporting query the semigroup is $(2^P, \cup)$. We assume that the semigroup operations take $O(1)$ time.

The semigroup model is very convenient, but it fails in two ways. First, sometimes the underlying structure is richer than a semigroup. For example, above we used subtraction in the group $(\mathbb{R}, +)$ to answer weighted counting queries for the intervals in $d = 1$. Second, when dealing with reporting queries we must spend time k to simply output the answer. Thus, we can afford to spend up to $O(k)$ extra operations without affecting the asymptotic time complexity.

It is important to note that in the weighted counting in $\mathbb{R}^1$ using the subtraction in $(\mathbb{R}, +)$ was helpful, but not necessary. It is possible to solve the problem using only addition. Let $P \subset \mathbb{R}^1$, and assume for simplicity that the number of points is a power of two[2], say $n = 2^r$. We will build a rooted binary tree with $2^{r+1} - 1$ nodes. Each node corresponds to an interval of the form $(t2^i, (t+1)2^i]$, and the interval I is the interval I' if the $I \subset I'$. In the node I we store the total weight of the points belonging to I.



A tree for the four-point set with weights $w(1) = 2$, $w(2) = -1$, $w(7) = 4$ and $w(9) = 1$.

Given this data structure the following simple algorithm output the sum of weights in a given interval.

[2]If the number of points is not a power of two, from asymptotic point of view it is easier to simply add dummy points, for it costs only a factor of two.

Algorithm 1 ComputeWeight algorithm

procedure ComputeWeight(range R, tree with the root I) ▷ Outputs the total weight of the points in R
    if $R \cap I = \emptyset$ **return** 0.
    if $I \subset R$ **return** $w(I)$.
    $W \leftarrow 0$ ▷ R overlaps I partially
    for each child I' of I **do**
        $W \leftarrow W + \text{ComputeWeight}(R, I')$
    end for
    return W.
end procedure

The algorithm takes $O(\log n)$ time because it visits a node only if it partially overlaps R. Such nodes lie on two paths from the root, one for each endpoint. The storage space is proportional to the number of nodes which is $2^{r+1} - 1 = O(n)$.

Range searching: partition trees

The tree we constructed in the previous section is an example of a *partition tree*. Its nodes are sets, and in each node we store precomputed sum of the weights. Most range searching algorithms are based on the same idea.

Consider halfspace ranges in $\mathbb{R}^2$. A simple partition tree is based on division $\mathbb{R}^2$ into four parts by a pair of lines, with each part containing at most $\lceil n/4 \rceil$ points of P. Each of the parts is further divided in four subparts in the same manner, and so one. When presented with a halfspace R we can compute $w(R)$ using the ComputeWeight algorithm. The time it takes is again proportional to the number of parts that R overlaps only partially. Let $f(n)$ be the maximum number of parts that R partially overlaps for the partition tree based on n points. Among the four original parts R either misses or fully contains at least one part, which leads to the recursion

$$f(4n) \leq 3f(n)$$

Thus $f(n) \leq O(n^{\log_4 3}) = O(n^{0.792\cdots})$. As the number of nodes is $O(\sum_{k \geq 0} n/4^k) = O(n)$, the amount of storage is linear too.

The same partition tree and same algorithm ComputeWeight can be used to answer the triangle range queries. If R is a triangle, which is the intersection of halfspaces R_1, R_2 and R_3, then the nodes that ComputeWeight visits on input R are among the nodes that it visits on inputs R_1, R_2 and R_3. Hence, the running time is similarly bounded.

Range searching: $O(n^{\frac{1}{2}+\varepsilon})$ algorithm in $\mathbb{R}^2$

Next we present a method based on partition trees for answering range queries in $O(n^{\frac{1}{2}+\varepsilon})$ in $\mathbb{R}^2$. The basic result we need is the following theorem.

Theorem 16. *Let P be a set of n points in $\mathbb{R}^d$. Let $D > 0$ be an arbitrary integer. Then there exists a polynomial $f \in \mathbb{R}[x_1, dotsc, x_d]$ of degree at most D such that the set $Z(f) \overset{def}{=} \{f = 0\}$ partitions $\mathbb{R}^2$ into open regions each containing at most $O(n/D^d)$ points of P.*

Corollary 17. *For every $\varepsilon > 0$ there is an linear-storage algorithm that answers halfspace queries in $\mathbb{R}^2$ in $O(n^{\frac{1}{2}+\varepsilon})$ time.*

Proof. Let D be large, but fixed. Consider the partition tree on P obtained by iterating the partition of theorem 16. If L is any line in $\mathbb{R}^2$, then L either lies in $Z(f)$ or meets$Z(f)$ in at most D points since the restriction of f on L is a polynomial of degree at most D. Thus, L meets at most $D + 1$ regions. Since ComputeWeight on a halfspace H visits only those regions that are intersected by the line $L + \partial H$, we obtain the following recursive bound on the query time $T(n)$ for the set of n points:

$$T(n) \leq (D + 1)T(n/D^2) + c_D,$$

where c_D is the cost of locating the $D + 1$ regions that ∂f intersects. Solving the recurrence we obtain $T(n) \leq c'_D n^{\log_{D^2}(D+1)}$. If D is chosen large enough for $\log_{D^2}(D + 1) < \frac{1}{2} + \varepsilon$, we obtain the requisite running time. □

The theorem 16 is a consequence of the following lemma in the case $d = 2$.

Lemma 18. *Let $m = \binom{D+d}{d} - 1$, and suppose $P_1, \dots, P_m \subset \mathbb{R}^d$ are finite point sets. Then there exists a polynomial g such that*

$$\big|\{g < 0\} \cap P_i\big|, \big|\{g > 0\} \cap P_i\big| \leq |P_i|/2.$$

Proof. Let M be the set of all non-constant monomials of degree at most D. Then $|M| = m$. Define the embedding[3] $\pi \colon \mathbb{R}^d \to \mathbb{R}^M$ by

$$\pi(p)_\tau = \tau(p) \qquad \text{for all } \tau \in M.$$

By ham-sandwich theorem there is a hyperplane $H = \{\sum_{\tau \in M} \alpha_\tau \tau = \alpha_0\}$ such that

$$\big|\pi(P_i) \cap H^+\big|, \big|\pi(P_i) \cap H^-\big| \leq |P_i|/2.$$

Since $f(p) > 0$ if and only if $\pi(p) \in H^+$, it follows that

$$\{f > 0\} \cap P_i = \pi^{-1}\big(H^+ \cap \pi(P_i)\big),$$

and similarly for $\{f < 0\} \cap P_i$. □

[3] This embedding is called the Veronese embedding.

Proof of theorem 16. We define a sequence $\mathcal{P}_0, \mathcal{P}_1, \dots$ of partitions of P and a matching sequence of polynomials $f_0, f_1, \dots$. We start with $\mathcal{P}_0 = \{P\}$ and $f_0 = 1$. At the i'th step, for $i = 1, 2, \dots$, with aid of lemma 18 we choose g_i to be a polynomial that partitions each $P \in \mathcal{P}_{i-1}$ into sets of size at most $|P|/2$. We set $f_i = f_{i-1} g_i$ and

$$\mathcal{P}_i = \big\{P \cap \{g_i > 0\} : P \in \mathcal{P}_{i-1}\big\} \cup \big\{P \cap \{g_i < 0\} : P \in \mathcal{P}_{i-1}\big\}.$$

This way f_i partitions P into sets each of which is contained in some member of $\mathcal{P}_i$. Since $\deg g_i \le c_d |\mathcal{P}_{i-1}|^{1/d}$ and $|\mathcal{P}_i| = 2^i$ we obtain

$$\deg f_i = \sum_{j=1}^{i} \deg f_i \le \sum_{j=1}^{i} c_d 2^{(i-1)/d} = c'_d 2^{i/d} = c'_d |\mathcal{P}_i|^{1/d}.$$

If i is the largest index for which $\deg f_i \le D$, then $\deg f_i \ge D/2$, and $\mathcal{P}_i$ contains $\Omega(D^d)$ regions. $\square$

Problems

1. Suppose $P_1 \subset \mathbb{R}^2$ and $P_2 \subset \mathbb{R}^2$ are two sets of n points each. Pick points p_1 and p_2 uniformly from P_1 and P_2 respectively.
   (a) Show that the probability that the line $l = p_1 p_2$ contains fewer than βn points of P_1 on one of the sides is at most 2β.
   (b) For any $\varepsilon < 1/4$ devise a linear-time probabilistic algorithm that finds a line that splits each of P_1 and P_2 parts of size at least εn each.
   (c) (Harder) Show that for every $\beta < 1/2$ the probability the line l cuts both P_1 and P_2 into pieces no smaller than βn is at least $p = p(\beta) > 0$ for some p.
2. (a) Show that if $f \in \mathbb{R}[x, y]$ has degree D, then $Z(f)$ partitions $\mathbb{R}^2$ into at most $O(D^2)$ regions. [Pick a point in each region, and consider a curve of small degree containing each of the points.]
   (b) Give a linear-storage algorithm with query time $O(n^{2/3+\varepsilon})$ for answering halfspace queries in $\mathbb{R}^3$.
   (c) (Hard) Show that the zero set of $f \in \mathbb{R}[x_1, \dots, x_d]$ of degree D partitions $\mathbb{R}^d$ into at most $O(D^d)$ regions. Conclude that there is a linear-storage algorithm with query time $O(n^{1/(d+1)+\varepsilon})$ for answering halfspace queries in $\mathbb{R}^{d+1}$.