Chapter 8

Randomized Algorithms

In this chapter we explore randomized algorithms for computational geometry. We begin with the technique of *randomized incremental construction*.

8.1 Randomized Incremental Construction

In this section we describe the randomized incremental construction technique. The main idea behind this approach to geometric algorithm design is to build some structure by incrementally inserting the input objects one at a time in random order. This gives rise to very simple algorithms, and, as we will show, it gives rise to algorithms with very good expected behavior as well. But before we describe this technique in detail, let us review a few important mathematical facts.

8.1.1 Some Preliminary Facts

First, lets say a couple of words about expectation.

Expectation

Let X be a discrete random variable that takes on values $x_1, x_2, \ldots, x_n$ with probabilities $p_1, \ldots, p_n$, respectively. The *expectation* $E(X)$ is simply the weighted sum

$$E(X) = \sum_{1 \leq i \leq n} x_i p_i.$$

An important property of this definition is that it implies that the expectation of a sum of random variables is the sum of the expectations. In particular, suppose we have a second random variable Y that takes on values $y_1, y_2, \ldots, y_m$ with probabilities $q_1, \ldots, q_n$, respectively. Then $X + Y$ is a random variable that takes on the values $x_i + y_j$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$ (we are not assuming that X and Y are necessarily independent). Let $p_{i,j}$ be the probability that $X + Y$ takes the value $x_i + y_j$.

From the definition, then,

$$\begin{aligned} E(X+Y) &= \sum_i \sum_j (x_i + y_j)\, p_{i,j} \\ &= \sum_i \sum_j x_i p_{i,j} + \sum_i \sum_j y_j p_{i,j} \\ &= \sum_i x_i \sum_j p_{i,j} + \sum_j y_j \sum_i p_{i,j} \end{aligned}$$

$$\begin{aligned} &= \sum_i x_i p_i + \sum_j y_j q_j \\ &= E(X) + E(Y). \end{aligned}$$

This generalizes to show that for random variables $X_1, X_2, \ldots, X_n$, the expectation

$$E\left(\sum_k X_k\right) = \sum_k E(X_k),$$

and this is true even if the X_k are not mutually independent.

Harmonic Numbers

The harmonic numbers, defined by $H_n = \sum_{1 \le i \le n} 1/i$ appear frequently in probabalistic analysis. In Knuth, vol. I, page 74, you can find the asymptotic expression

$$H_n = \ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \epsilon, \qquad 0 < \epsilon < \frac{1}{252n^6}.$$

Here is a quick wordless proof that $b(n)/2 < H_n \le b(n)$, where $b(n)$ is the number of bits needed to write n in binary.

$$\begin{array}{lllllllll} H_n = 1 & +1/2 & +1/3 & +1/4 & +1/5 & +1/6 & +1/7 & +1/8 & +1/9 & +\cdots \\ < 1 & +1/2 & +1/2 & +1/4 & +1/4 & +1/4 & +1/4 & +1/8 & +1/8 & +\cdots \\ < 1 & +1 & & +1 & & & & +1 & & +\cdots \\ = b(n) & & & & & & & & & \\ H_n = 1 & +1/2 & +1/3 & +1/4 & +1/5 & +1/6 & +1/7 & +1/8 & +1/9 & +\cdots \\ > 1 & +1/2 & +1/4 & +1/4 & +1/8 & +1/8 & +1/8 & +1/8 & +1/16 & +\cdots \\ > 1 & +1/2 & & +1/2 & & & & +1/2 & & +\cdots \\ = b(n)/2 & & & & & & & & & \end{array}$$

Chernoff Bounds

It is often necessary in the analysis randomized algorithms to bound the sum of a set of random variables. One set of inequalities that makes this tractable is the set of Chernoff Bounds.

Let $X_1, X_2, \ldots, X_n$ be a set of mutually independent indicator random variables, such that each X_i is 1 with some probability $p_i > 0$ and 0 otherwise. Let $X = \sum_{i=1}^n X_i$ be the sum of these random variables, and let μ denote the mean of X, i.e., $\mu = E(X) = \sum_{i=1}^n p_i$.

Theorem 8.1.1: *Let X be as above. Then, for $\delta > 0$,*

$$\Pr(X > (1+\delta)\mu) < \left[\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right]^\mu, \tag{8.1}$$

and, for $0 < \delta \le 1$,

$$\Pr(X < (1-\delta)\mu) < e^{-\mu\delta^2/2}. \tag{8.2}$$

Proof: See the book *Randomized Algorithms* by Motwani and Raghavan. □

8.1.2 3-Dimensional Convex Hull Construction

The first application of the RIC technique we give is for a simple randomized algorithm for constructing the convex hull of n points in 3-space. The method we describe runs in $O(n \log n)$ expected time and uses $O(n)$ expected space. The construction is *incremental*, because it adds points one at

a time and updates the hull immediately, and *randomized*, because it chooses which point to add randomly. More importantly, the expected time and space bounds we will derive for this algorithm are taken over the random choices made by the algorithm, and is not dependent of the distribution of the input points.

Assume, to make the description easier, that our set of points in space $P = \{p_1, p_2, \ldots, p_n\}$ is in general position (no four points are coplanar). Then the faces of convex hulls of subsets of P are triangles. The algorithm maintains a tetrahedralization of the convex hull of $\{p_1, p_2, \ldots, p_k\}$, as k goes from 4 to n.

Procedure **RIChull**
  Let $\{p_1, p_2, \ldots, p_n\}$ be a random permutation of the n input points.
  Form the tetrahedron T_4 defined by $\{p_1, p_2, p_3, p_4\}$
  Choose a *center* point $c \in T_4$
  for $k = 5$ to n do
    Walk through the tetrahedra of T_{k-1} from c to p_k
    If p_k is inside a tetrahedron of T_{k-1}, then
      p_k is inside the convex hull and $T_k = T_{k-1}$
    else
      Determine the faces of T_{k-1} that p_k sees,
        starting with the last face encountered by the ray $\overrightarrow{cp}_k$.
      T_k is the tetr'ization formed by erecting tetrahedra
        from each visible face to p_k and adding them to T_{k-1}.

Space: To analyze the expected space required by this algorithm we can count the number of tetrahedra formed in going from step $k-1$ to step k. We let $|T_k|$ denote the total number of tetrahedra in tetr'ization T_k and let s_k be the number of faces on its outer surface, which is the convex hull of $\{p_1, p_2, \ldots, p_k\}$. ($|T_4| = s_4 = 4$.)

If the point p_k is not on the convex hull, then no tetrahedra are formed: $|T_k| = |T_{k-1}|$. If p_k is on the convex hull, then one tetrahedron is added for each surface face that is destroyed. The number of new surface faces created is the *degree* $d(p_k)$ of the vertex p_k on the convex hull. Therefore, the number of tetrahedra created equals

$$|T_k| - |T_{k-1}| = d(p_k) + s_{k-1} - s_k.$$

Summing the right and left hand sides of this equation over $5 \le k \le n$ gives

$$|T_n| - |T_4| = s_4 - s_n + \sum_{5 \le k \le n} d(p_k).$$

We bound the expected number of tetrahedra $E(|T_n|)$ by the expected sum of degrees,

$$E\Big(\sum_{5 \le k \le n} d(p_k)\Big) = \sum_{5 \le k \le n} E\left(d(p_k)\right) < \sum_{5 \le k \le n} 6 < 6n.$$

To bound the expectation of the kth degree $d(p_k)$, we think of running the algorithm backwards—starting from a convex hull and removing vertices until only the first four remain. The sum of degrees of all vertices on a given hull is twice the number of hull edges. Thus, the average degree is at most $6 - 12/n$; if we remove a vertex at random, we can expect to remove less than 6 surface faces. Thus, the expected space is bounded above by $6n$.

Time: The time spent constructing faces is constant per face, so we know that data structure manipulation is expected to be linear by the previous analysis. (You should convince yourself that you can store the tetr'ization in a data structure that lets you spend constant time per face.) The crucial quantity for the running time is how many faces are intersected by each segment $\overline{cp_k}$—we spend time proportional to this amount in walking from c to p_k.

Let us say that four points $(p, q, r; s)$ are a *conflict* if the segment $\overline{cs}$ intersects $\triangle pqr$. We say that a conflict *arises* if the face $\triangle pqr$ is constructed by the algorithm sometime before s is added. (This definition is a bit redundant—once s is added the algorithm cannot construct $\triangle pqr$ as a face—but that doesn't matter.)

We would like to count the expected number of conflicts that arise during an execution of the algorithm on n points. Let Z_Δ be a random variable that is 1 if the specific conflict $\Delta = (p, q, r; s)$ arises in the first n steps. The quantity that we want to know is

$$\sum_{\forall \text{ conflicts } \Delta} Z_\Delta.$$

The probability that a conflict Δ arises at step k is the probability that Δ exists at step k and did not at step $k-1$. Thus, if we let $X_{\Delta,i}$ be a random variable that is 1 if and only if conflict $\Delta = (p, q, r; s)$ exists at step i, that is, triangle $\triangle pqr$ is on the surface of the convex hull and s is outside the hull and projects onto $\triangle pqr$. In symbols,

$$Z_\Delta = \sum_{1 \le i \le n} X_{\Delta,i} \cap \overline{X_{\Delta,i-1}}.$$

We can now evaluate the expectation. First, write the intersection in terms of conditional probability.

$$\begin{aligned} \sum_{\forall \text{ conflicts } \Delta} Z_\Delta &= \sum_{\Delta} \sum_{1 \le i \le n} X_{\Delta,i} \cap \overline{X_{\Delta,i-1}} \\ &= \sum_{\Delta} \sum_{i} \Pr(X_{\Delta,i}) \cdot \Pr(\overline{X_{\Delta,i-1}} \mid X_{\Delta,i}) \end{aligned}$$

The conditional term is the probability that one of the three vertices of the triangle of Δ was chosen last of i vertices. Thus, we can bound the previous expression by

$$\sum_{\Delta} \sum_{i} \Pr(X_{\Delta,i}) \cdot \frac{3}{i}.$$

Next we change the order of summation, which allows us to re-write this as

$$\sum_{i} \frac{3}{i} \sum_{\Delta} \Pr(X_{\Delta,i}).$$

The inner sum is a fancy way to count the points outside of the convex hull of the first i points. It is surely less than n.

$$\sum_{\forall \text{ conflicts } \Delta} Z_\Delta \le \sum_{i} \frac{3}{i} n = 3nH_n = O(n \log n).$$

This shows that the expected time is $O(n \log n)$. At the present time, bounds on the variance are unknown. There are some *tail estimate* bounds on the space of the form "The probability that the space exceeds c times the expectation is at most $(c/e)^{-c}/e$."

8.1.3 Segment Intersection

We can develop an RIC algorithm and data structure for segment intersection as well. Suppose we are given n segments that have K intersections. We begin with an empty trapezoidation. At stage $k+1$, we insert segment s_{k+1} into the trapezoidation of $s_1, s_2, \ldots, s_k$—this means that we must figure out which trapezoids s_{k+1} intersects (location) and cut them into smaller trapezoids. As we will describe in a moment, we use the history of the trapezoidation to locate s_{k+1}. Our final space and running time will be $O(n \log n + K)$, expected.

The data structures that we need are:

- a trapezoidation of the k current segments. Decompose the complement of $s_1, s_2, \ldots, s_k$ into trapezoids by making vertical cuts from each endpoint and intersection point. Each trapezoid is defined by at most four segments and can have pointers to the at most four trapezoids that share vertical cuts.
- A history DAG (directed acyclic graph). When we cut a trapezoid τ into several new "child" trapezoids by introducing a new segment we push into history and give it pointers down to its "children." This is our location structure.

To insert a segment $s = s_{k+1}$, we begin by giving it to the single trapezoid at the top of the history DAG. This trapezoid has a constant number of children, so it breaks s into the appropriate number of pieces and passes s down to each child that s intersects. Thus, s is passed down until it reaches the current trapezoidation—intersections can be reported as s is passed down or in a separate phase. There s updates all trapezoids it intersects (pushing modified trapezoids up into the history DAG). The trapezoids' neighbor pointers are used to merge neighboring trapezoids.

Define a conflict Δ to be a pair consisting of a trapezoid τ, which is defined by at most four segments, and a segment s that intersects τ. Again, let Z_Δ be an indicator random variable for conflict Δ. The cost of the algorithm is exactly $\sum_{\forall \Delta} Z_\Delta$, because we have to pass s through every trapezoid that it is in conflict with.

As before, if we let $X_{\Delta,i}$ be a random variable that is 1 if and only if conflict Δ exists at step i then

$$\begin{aligned}\sum_{\forall \text{ conflicts } \Delta} Z_\Delta &= \sum_\Delta \sum_i \Pr(X_{\Delta,i}) \cdot \Pr(\overline{X_{\Delta,i-1}} \mid X_{\Delta,i}) \\ &= \sum_\Delta \sum_i \Pr(X_{\Delta,i}) \cdot \frac{4}{i} \\ &= \sum_i \frac{4}{i} \sum_\Delta \Pr(X_{\Delta,i})\end{aligned}$$

The inner sum is the number of conflicts between segments $s_{i+1}, \ldots, s_n$ and the trapezoids at stage i for a random permutation. This will be $O(n(2 + K_{i+1}/(i+1)))$, where K_{i+1} is the number of intersections in a random permutation of the first $i+1$ segments, which is $K(i(i+1))/(n(n-1))$. Thus,

$$\sum_{\forall \text{ conflicts } \Delta} Z_\Delta \leq O(\sum_i \frac{n}{i} + \frac{K}{(n-1)}) = O(n \log n + K).$$

8.1.4 Linear Programming and Generalized LP

We discussed earlier the 2-dimensional linear programming problem. In a general form, the linear programming problem is to find a vector $x \in \mathcal{R}^d$ minimizing $\vec{c} \cdot x$ subject to the linear equations $Ax \leq b$. We can express this in geometric terms. Let H_i be the hyperplane in $\mathcal{R}^d$ that is given by the ith row of A and b_i. Then we find a point p in the intersection of n d-dimensional halfplanes $H_1, H_2, \ldots, H_n$ that maximizes $p \cdot \vec{c}$ for a d-dimensional cost vector $\vec{c}$.

Let's assume we are only interested in a solution in some bounding box B: $x_i \in [min, max]$, for each coordinate index $1 \leq i \leq d$. Then we can solve a d-dimensional linear programming problem by finding the optimum vertices of the polytopes

$$P_i = B \cap \bigcap_{1 \leq j \leq i} H_j$$

by adding one hyperplane at a time.

1. Put the hyperplanes $H_1, H_2, \ldots, H_n$ in a random order.

2. Let v_0 be optimum vertex of the bounding box with respect to $\vec{c}$. There are a constant number of vertices (2^d), so this takes constant time.

3. For $i := 1$ to n do 4–6:

4. If hyperplane H_i contains v_{i-1} then the optimum vertex $v_i = v_{i-1}$.

5. Otherwise, H_i cuts v_{i-1} off the polytope P_{i-1} in forming P_i. The new optimum vertex v_i, if it exists, is contained in the hyperplane h that bounds H_i because the cost increases along the segment from v_i to v_{i-1}. To find it:

   (a) Project the cost vector $\vec{c}$ onto h to obtain $\vec{c}'$.

   (b) Recursively solve the $d-1$ dimensional linear program of maximizing $\vec{c}'$ in h subject to $B, H_1, \ldots, H_{i-1}$.

   (c) The recursion bottoms out in one dimension, where we can easily find the maximum satisfying the constraints or determine that no maximum exists.

Linear Programming Theorem. *A linear programming problem with n constraints in d dimensions is solved in $O(d!n)$ expected time.*

Proof: We can prove this by induction on d. It should be clear that the theorem is true for dimension $d = 1$.

Assume the theorem is true for dimension $d-1$; we can solve the recursive call in step 6 in $k(d-1)!(i-1)$ time for some constant k. How often do we need to do this? That is, how often is v_i different from v_{i-1}? Well, v_i is defined by d hyperplanes and it can only be different from v_{i-1} if one of these hyperplanes' constraints is chosen as H_i. This happens with probability d/i, since all orderings are equally likely. Since the expected cost of adding constraint H_i is

$$k(d-1)!(i-1) \cdot \frac{d}{i} < kd!$$

the total expected cost of the algorithm is $kd!n$. □

8.1.5 Generalized Linear Programming

We can generalize the above algorithm to solve some problems in *convex programming*—where the contraints are convex functions, not just linear functions. As an example, let's look at a devious algorithm by Emo Welzl for computing the smallest circle enclosing a set of n points. We first show that the call **Circle**$(S, \emptyset)$ correctly returns this circle and then prove that it does so in $O(|S|)$ expected time.

```
Procedure Circle(S, P) returns a circle that passes through
          the points of P and contains S, if one exists:
  If |S| = 0 or |P| = 3 then
    Return the smallest circle through P in O(1) time.
  Else
    Pick a random point p from S.
    c = Circle(S − {p}, P).
    If p ∈ c then
        Return c
    Else
        Return Circle(S − {p}, P ∪ {p}).
```

If we assume general position, as usual, then the circle enclosing S is defined by either two or three points of S. (Two points, if they define the diameter of the circle.) Because we start with P empty, **Circle**(S, P) can be called with zero, one, two or three points in P.

We prove, by induction on $|S|$, the statement "If **Circle**(S, P) is called with P being a subset of the points defining the smallest enclosing circle of $S \cup P$, then **Circle**(S, P) correctly computes the smallest enclosing circle of $S \cup P$." For the base case, we can prove it by hand for all sets with $|S| \leq 3$ and $|P| \leq 3$.

Now, we assume that the result is true for sets S' of size $n-1$ and prove it for sets S of size n. First, if $|P| = 3$, then P contains all the points defining the minimum enclosing circle of S and **Circle**(S, P) correctly returns that circle. If $|P| < 3$, then the first recursive call $c =$ **Circle**$(S - \{p\}, P)$ returns the correct circle enclosing $(S - \{p\}) \cup P$ by the induction hypothesis. If $p \in c$, then c contains $S \cup P$ and is correctly returned. Otherwise, p must be one of the points that defines the minimum enclosing circle. Therefore the second call **Circle**$(S - \{p\}, P \cup \{p\})$ returns the correct circle.

To analyze expected running time, we write recurrences that depend on the number of points in S and P. Let $T_i(n)$ denote the upper bound on the expected running time of **Circle**(S, P) with $|S| = n$ and $|P| = 3 - i$. We want to bound the expected running time $T_3(n)$ by $O(n)$.

With this notation, i is the number of points that still must be specified on the smallest enclosing circle. A call **Circle**(S, P) will generate the second recursive call **Circle**$(S - \{p\}, P \cup \{p\})$ if and only if p happens one of the i points that define the smallest enclosing circle. The probability of this is at most $i/|S|$. For some constants c and c', the expected running times satisfy

$$\begin{aligned} T_1(n) &\leq T_1(n-1) + c' + \frac{1}{n} T_0(n-1) \\ T_2(n) &\leq T_2(n-1) + c' + \frac{2}{n} T_1(n-1) \\ T_3(n) &\leq T_3(n-1) + c' + \frac{3}{n} T_2(n-1) \end{aligned}$$

Which have solutions:

$$\begin{aligned} T_1(n) &\leq c'n + c \ln n \leq (c + c')n \\ T_2(n) &\leq (2c + 3c')n \\ T_3(n) &\leq (6c + 10c')n \end{aligned}$$

By the way, this algorithm can be extended to higher dimensions and to finding smallest ellipsoids. Suppose d points are needed to define the basic object (sphere, ellipsoid, ...). Further, suppose the object defined by d points can be computed in c operations and a point-in-object test takes c' operations. (Both c and c' may hide factors of d.) Then we can show that $T_d \leq (c + c')d!n$ as follows:

$$\begin{aligned} T_d(n) &\leq T_d(n-1) + c' + \frac{d}{n} T_{d-1}(n-1) \\ &\leq (c + c'd)d!(n-1) + c' + \frac{d}{n}(c + c'(d-1))(d-1)!(n-1) \\ &\leq (c + c'd)d!n - (c + c'd)d! + c' + (c + c'd)d! - (c + c')d! \\ &\leq (c + c'd)d!n. \end{aligned}$$

8.2 Random Sampling

Another powerful technique in randomized geometric algorithm design is *random sampling*. The general scenerio is that one is given a collection S of geometric objects and asked to construct some geometric structure for S. The technique involves selecting a random sample Y of S of size r, for some parameter r, and then using Y to decompose S into subproblems to be solved recursively. The subproblem solutions are then merged in some way to define the final step in this divide-and-conquer algorithm.

8.2.1 Polygon Triangulation Revisited

We illustrate this approach using the specific problem of polygon triangulation. Recall that in this problem one is given an n-node polygon P, and one wishes to add diagonals to P so that each internal face is a triangle. As we have shown earlier, it is enough for us to produce a trapezoidal decomposition of P, where we add a vertical line interior to P up and/or down from each vertex of P until it hits an edge of P. We already have described a simple $O(n \log n)$-time method for producing a trapezoidal decomposition of a set of n segments, which need not be connected. As we show below, we can easily turn this algorithm into a simple randomized method for triangulating a simple polygon in $O(n \log \log n)$ expected time.

We begin by selecting a random sample, Y, of $r = n/\log n$ edges of P. We apply the simple plane-sweeping algorithm to produce a trapezoidal decomposition, $T(Y)$, of Y in time $O(r \log r) = O(n)$ time. We then determine, for each trapezoid τ in $T(Y)$, the set, C_τ of edges of P that intersect the interior of τ. We refer to this set as the *conflict set* for τ. We can determine C_τ by "walking" along the edges of P noting for each edge s of P the trapezoids we cross as we traverse s (we deposit the name of s in the conflict set for each such trapezoid). Since each edge of s begins where another ends, we can perform this entire walk around P in $O(n + \sum_{\tau \in T(Y)} n_\tau)$ time, where n_τ denotes the number of edges in C_τ. We complete the algorithm, then, by applying the simple plane-sweep trapezoidal decomposition algorithm to each C_τ.

Analysis. To analyze this algorithm we concentrate on two important numbers:

$$\max_{\tau \in T(Y)} n_\tau, \text{ and} \tag{8.3}$$

$$\sum_{\tau \in T(Y)} n_\tau. \tag{8.4}$$

We begin with a bound for (8.3), which holds with n-polynomial probability:

Lemma 8.2.1: $\max_{\tau \in T(Y)} n_\tau \leq c(n/r) \log r$ *with probability at least* $1 - 1/n$.

Proof: Each trapezoid τ in $T(Y)$ exists because of two facts:

1. Each of the edges defining τ's boundary are in Y. There are at least two and at most four such edges.
2. Each of the edges of C_τ are not in Y.

Moreover, we can abstractly define the complete set of $(O(n^4))$ potential trapezoids in terms of these two sets of edges, the first of which we call τ's *triggers* and the second of which we call τ's *killers* (indeed, this notion motivates our calling C_τ the "conflict set" for τ). This allows us to analyze the probability that some potential trapezoid τ is included in $T(Y)$:

$$\Pr(\tau \in T(Y)) \leq \left(\frac{r}{n}\right)^2 \left(1 - \frac{r}{n}\right)^{n_\tau}.$$

Using the fact that $(1 - x/m)^m \leq e^x$, we can bound this probability by

$$\left(\frac{r}{n}\right)^2 e^{-t_\tau},$$

where $t_\tau = n_\tau/(n/r)$, a quantity we call the *excess* for τ. Thus, the probability that a trapezoid τ with excess more than $2 \ln r$ is included in $T(Y)$ is at most $1/n^2$. Since there are $O(r) = O(n)$ trapezoids total in $T(Y)$, the probability that the (combinatorially) largest trapezoid has excess more than $2 \ln r$ is at most $1/n$. □

Having established a bound on (8.3) that holds with n-polynomial probability, we next give an expected bound on (8.4).

Lemma 8.2.2: *$E(\sum_{\tau \in T(Y)} n_\tau) \leq cn$, for some constant $c > 1$.*

Proof: We establish this lemma by showing that the expected number of trapezoids traversed by a random edge e of P is $O(1)$. We, in turn, establish this by the now-familiar "backward analysis." Let Y' denote the random sample $Y \cup \{e\}$, which has size $r+1$. Note that the number of trapezoids crossed by e in $T(Y)$ is proportional to the number of trapezoids of $T(Y')$ that are not in $T(Y)$, that is, the number of trapezoids of $T(Y')$ that would be destroyed by the removal of e from Y'. But with respect to Y' the edge e appears to be chosen at random from its $r+1$ edges. Since any trapezoid in $T(Y')$ has at most 4 triggers, this implies that any trapezoid τ in $T(Y')$ has probability at most $4/(r+1)$ of being destroyed by the removal of the random edge e from Y'. Since there are $O(r)$ edges total in Y', this implies that the expected number of edges of $T(Y')$ destroyed by the removal of e is $O(1)$. □

Returning to the analysis of our algorithm, these two lemmas establish that the expected running time of our randomized polygon trapezoidal decomposition algorithm is $O(n \log\log n)$. By being a little more clever in how we use the random sampling technique we can actually improve this running time to have an expected value of $O(n \log^* n)$, where $\log^* n$ is the familiar iterated logarithm function (defined as the smallest k so that $\log^{(k)} n < 2$, where $\log^{(1)} n = \log n$ and $\log^{(k)} n = \log\log^{(k-1)} n$). The change involves iteratively applying the random sampling technique in a kind of "globally" recursive fasion.

We begin as described above, constructing the trapezoidal decomposition $T(Y_1)$ of a random sample $Y_1 (= Y)$, whose size is $r_1 = n/\log n$. We also determine all the conflict sets for $T(Y_1)$ by traversing the polygon P, as described above. We then define a random sample Y_2 of size $r_2 = n/\log\log n$ of the edges of P. The sample Y_2, of course, defines a random sample $Y_2 \cap C_\tau$ of each C_τ with expected size $n_\tau(n/r_2) = n_\tau/\log\log n$. We then produce a trapezoidal decomposition of each set $Y_2 \cap C_\tau$ using the simple plane-sweep algorithm. The expected running time of this computation is $O(n_\tau \log n_\tau/\log\log n)$, which is $O(n_\tau)$ for each trapezoid τ in $T(Y_1)$, by Lemma 8.2.1. By then removing edges of $T(Y_1)$ that are now determined to not be a part of $T(Y_2)$ we can produce a representation of $T(Y_2)$. Given $T(Y_2)$, we can again traverse the polygon P to determine the conflict set for each trapezoid in $T(Y_2)$ in $O(n)$ expected time (by Lemma 8.2.2). We may therefore repeat this method, producing $T(Y_{i+1})$ from $T(Y_i)$ in $O(n)$ time, where each Y_i is a random sample of size $n/\log^{(i)} n$ of the edges of P. Since each iteration takes $O(n)$ expected time, and there are clearly at most $\log^* n$ iterations, the total time for producing $T(P)$ is expected to be $O(n \log^* n)$.

8.2.2 ϵ-Nets, ϵ-Approximations, and ϵ-Cuttings

The random sampling concept can actually be couched in a fairly general setting, which allows for a wide applicability to geometric problems. We explore this general formulation in this subsection.

Let $(X, \mathcal{R})$ be a *set system*, where X is an n-element *gound set* and $\mathcal{R}$ is a collection of subsets of X. Moreover, let us assume that the sets in $\mathcal{R}$, which we will call *ranges*, are generated by some geometric property of the elements in X. For example, the set X could be n points in the plane and $\mathcal{R}$ could be all combinatorially-distinct ways of intersecting points in X with disks. Or, alternatively, X could be a set of n hyperplanes in $\mathcal{R}^d$ and $\mathcal{R}$ could be all combinatorially-distinct ways of intersecting hyperplanes in X with d-simplices (the d-dimensional generalization of triangles).

For any subset $Y \subseteq X$, we let $\mathcal{R}|_Y$ denote the collection of ranges restricted to Y, i.e.,

$$\mathcal{R}|_Y = \{R \cap Y : R \in \mathcal{R}\}.$$

The *shatter function*, $\pi_{\mathcal{R}}(n)$, for $(X, \mathcal{R})$ is defined as follows:

$$\pi_{\mathcal{R}}(m) = \max\{|\mathcal{R}|_Y| : Y \subseteq X, |Y| = m\}.$$

It provides, in some sense, a measure of the recursive complexity of a set system $(X, \mathcal{R})$. Define the *VC-exponent* of $(X, \mathcal{R})$ to be smallest paramenter d such that $\pi_{\mathcal{R}}(m)$ is $O(m^d)$ (actually, this

should be defined as an infimum). This quantity is closely related to (and subsumes) the notion of *VC-dimension*, which is common in the Computational Geometry literature. For the remainder of this section we assume that the set systems we are dealing with have bounded VC-exponent (this is true, for example, for the set systems mentioned above).

A subset $Y \subseteq X$ is an *ϵ-net* for $(X, \mathcal{R})$ if, for any range $R \in \mathcal{R}$, $Y \cap R \neq \emptyset$ any time $|R| > \epsilon n$. Intuitively, the subset Y "captures" all the big ranges.

Theorem 8.2.3: *If $(X, \mathcal{R})$ has bounded VC-exponent, then there is a constant $c > 0$ such that if $Y \subset X$ is of size at least $cr \log r$, then Y is a $(1/r)$-net, with r-polynomial probability.*

Proof: Let Y be as above. Let us analyze the probability that Y is a $(1/r)$-net. Specifically, let R be some range with $|R| > n/r$. We can view the value of $|Y \cap R|$ as a random variable, which has mean $|R|s/n$, where s is the size of $|Y|$. Thus,

$$\begin{aligned} \Pr(Y \cap R = \emptyset) &= \Pr(|Y \cap R| = 0) \\ &= \Pr(|Y \cap R| \leq (1-1)|R|s/n). \end{aligned}$$

We may therefore apply a Chernoff bound to bound this quantity by

$$e^{-|R|s/2n} < e^{-s/2r}.$$

If we let d denote the VC-exponent of $(X, \mathcal{R})$, then we know that the number of combinatorially-distinct ranges, with respect to Y, is $O(s^d)$. Thus, the probability that Y is not a $(1/r)$-net is bounded by

$$c_0 s^d e^{-s/2r},$$

for some constant $c_0 \geq 1$. We can make this less than $1/r^k$, for example, by choosing $s = 4kdr \ln r \ln c_0$. □

A related notion to that of an ϵ-net is that of an *ϵ-approximation*, which is defined as a subset $Y \subseteq X$ such that for each range $R \in \mathcal{R}$,

$$\left| \frac{|Y \cap R|}{|Y|} - \frac{|R|}{|X|} \right| < \epsilon.$$

Note that an ϵ-approximation is automatically an ϵ-net, but an ϵ-net need not be an ϵ-approximation. We will not prove it here, but, using the Chernoff bounds, one can show that a random subset Y of size $\Theta(r^2 \log r)$ is a $(1/r)$-approximation with r-polynomial probability.

As an application of this ϵ-net theory, consider the following problem: one is given a set X of n hyperplanes in $\mathcal{R}^d$ and asked to produce a triangulation of space (into d-simplices) so that each simplex intersects at most n/r hyperplanes, for a given parameter $0 < r < n$. Such a triangulation is called an ϵ-cutting, for $\epsilon = 1/r$. We can easily construct a $(1/r)$-cutting by first forming a $(1/r)$-net Y of the set of hyperplanes (under d-simplex ranges), computing the arrangement of the hyperplanes in Y, and triangulating each face in this arrangement. By the zone lemma for hyperplanes, this will produce $O(|Y|^d) = O(r^d \log^d r)$ d-simplices, each of which is guaranteed to intersect at most n/r hyperplanes. That is, it will construct a $(1/r)$-cutting.