44 RANDOMIZATION AND DERANDOMIZATION

Otfried Cheong, Ketan Mulmuley, and Edgar Ramos

INTRODUCTION

Randomization as an algorithmic technique goes back at least to the seventies. In a seminal paper published in 1976, Rabin uses randomization to solve a geometric problem, giving an algorithm for the closest-pair problem with expected linear running time [Rab76]. Randomized and probabilistic algorithms and constructions were applied successfully in many areas of theoretical computer science. Following influential work in the mid-1980s, a significant proportion of published research in computational geometry employed randomized algorithms or proof techniques. Even when both randomized and deterministic algorithms of comparable asymptotic complexity are available, the randomized algorithms are often much simpler and more efficient in an actual implementation. In some cases, the best deterministic algorithm known for a problem has been obtained by "derandomizing" a randomized algorithm.

This chapter focuses on the randomized algorithmic *techniques* being used in computational geometry, and not so much on particular results obtained using these techniques. Efficient randomized algorithms for specific problems are discussed in the relevant chapters throughout this Handbook.

GLOSSARY

Probabilistic or "Monte Carlo" algorithm: Traditionally, any algorithm that uses random bits. Now often used in contrast to *randomized algorithm* to denote an algorithm that is allowed to return an incorrect or inaccurate result, or fail completely, but with small probability. Monte Carlo methods for numerical integration provide an example. Algorithms of this kind have started to play a larger role in computational geometry in the 21st century (Section 44.8).

Randomized or "Las Vegas" algorithm: An algorithm that uses random bits and is guaranteed to produce a correct answer; its running time and space requirements may depend on random choices. Typically, one tries to bound the expected running time (or other resource requirements) of the algorithm. In this chapter, we will concentrate on randomized algorithms in this sense.

Expected running time: The expected value of the running time of the algorithm, that is, the average running time over all possible choices of the random bits used by the algorithm. No assumptions are made about the distribution of input objects in space. When expressing bounds as a function of the input size, the worst case over all inputs of that size is given. Normally the random choices made by the algorithm are hidden from the outside, in contrast with average running time.

Average running time: The average of the running time, over all possible inputs. Some suitable distribution of inputs is assumed.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

To illustrate the difference between expected running time and average running time, consider the *Quicksort* algorithm. If it is implemented so that the pivot element is the first element of the list (and the assumed input distribution is the set of all possible permutations of the input set), then it has $O(n \log n)$ *average* running time. By providing a suitable input (here, a sorted list), an adversary can force the algorithm to perform worse than the average. If, however, Quicksort is implemented so that the pivot element is chosen at random, then it has $O(n \log n)$ *expected* running time, for any possible input. Since the random choices are hidden, an adversary cannot force the algorithm to behave badly, although it may perform poorly with some positive probability.

Randomized divide-and-conquer: A divide-and-conquer algorithm that uses a random sample to partition the original problem into subproblems (Section 44.1).

Randomized incremental algorithm: An incremental algorithm where the order in which the objects are examined is a random permutation (Section 44.2).

Tail estimate: A bound on the probability that a random variable deviates from its expected value. Tail estimates for the running time of randomized algorithms are useful but seldom available (Section 44.9).

High-probability bound: A strong tail estimate, where the probability of deviating from the expected value decreases as a fast-growing function of the input size n. The exact definition varies between authors, but a typical example would be to ask that for any $\alpha > 0$, there exists a $\beta > 0$ such that the probability that the random variable $X(n)$ exceeds $\alpha E[X(n)]$ be at most $n^{-\beta}$.

Derandomization: Obtaining a deterministic algorithm by "simulating" a randomized one (Section 44.6).

Coreset: A data set of small size that can be used as a proxy for a large data set. Algorithms can be run on the coreset to obtain a good approximation of the result for the full data set. Since a random sample captures many characteristics of a given data set, a coreset can be considered a stronger form of a random sample. Chapter 48 discusses coresets in detail, and mentions randomization frequently.

Trapezoidal map: A planar subdivision $\mathcal{T}(S)$ induced by a set S of line segments with disjoint interiors in the plane (cf. Section 33.3). $\mathcal{T}(S)$ can be obtained by passing vertical attachments through every endpoint of the given segments, extending upward and downward until each hits another segment, or extending to infinity; see Figure 44.0.1. Every face of the subdivision is a trapezoid (possibly degenerated to a triangle, or with a missing top or bottom side), hence the name.

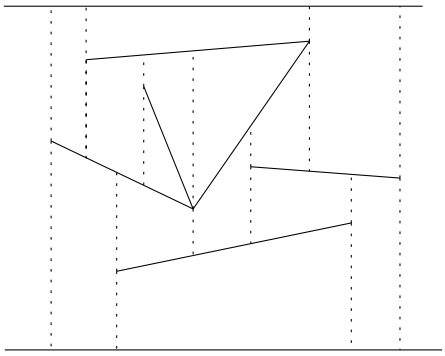

FIGURE 44.0.1
The trapezoidal map of a set of 6 *line segments.*

We will use the problem of computing the trapezoidal map of a set of line segments with disjoint interiors as a running example throughout this chapter. We assume for presentation simplicity that no two distinct endpoints have the same x-coordinate, so that every trapezoid is adjacent to at most four segments. (This can be achieved by a slight rotation of the vertical direction.)

The trapezoidal map can also be defined for intersecting line segments. In that situation, vertical attachments must be added to intersection points as well, and the map may consist of a quadratic number of trapezoids. The trapezoidal map is also called the ***vertical decomposition*** of the set of line segments. Decompositions similar to this play an important role in randomized algorithms, because most algorithms assume that the structure to be computed has been subdivided into elementary objects. (Section 44.5 explains why this assumption is necessary.)

44.1 RANDOMIZED DIVIDE-AND-CONQUER

GLOSSARY

Top-down sampling: Sampling with small, usually constant-size random samples, and recursing on the subproblems.

Cutting: A subdivision Ξ of space into simple cells Δ (of constant description complexity, most often simplices). The ***size*** of a cutting is the number of cells.

ϵ-cutting Ξ: For a set X of n geometric objects, a cutting such that every cell $\Delta \in \Xi$ intersects at most n/r of the objects in X (also called a $1/r$-cutting with $\epsilon = 1/r$ when convenient). See also Chapter 47.

Bottom-up sampling: Sampling with random samples large enough that the subproblems may be solved directly (without recursion).

Bernoulli sampling: The "standard" way of obtaining a random sample of size r from a given n-element set uses a random number generator to choose among all the possible subsets of size r, with equal probability for each subset (also obtained as the first r elements in a random permutation of n elements). In Bernoulli sampling, we instead toss a coin for each element of the set independently, and accept it as part of the sample with probability r/n. While the size of the sample may vary, its expected size is r, and essentially all the bounds and results of this chapter hold for both sampling models. Sharir showed that in fact this model can be analysed more easily than the standard one [Sha03].

Gradation: A hierarchy of samples for a set X of objects obtained by bottom-up sampling:

$$X = X_1 \supset X_2 \supset X_3 \supset \cdots \supset X_{r-1} \supset X_r = \emptyset.$$

With Bernoulli sampling, a new element can be inserted into the gradation by flipping a coin at most r times, leading to efficient dynamic data structures (Section 44.3).

Geometric problems lend themselves to solution by divide-and-conquer algorithms. It is natural to solve a geometric problem by dividing space into regions

(perhaps with a grid), and solving the problem in every region separately. When the geometric objects under consideration are distributed uniformly over the space, then gridding or "slice-and-dice" techniques seem to work well. However, when object density varies widely throughout the environment, then the decomposition has to be fine in the areas where objects are abundant, while it may be coarse in places with low object density. Random sampling can help achieve this: the density of a random sample R of the set of objects will approach that of the original set. Therefore dividing space according to the sample R will create small regions where the geometric objects are dense, and larger regions that are sparsely populated.

We can distinguish two main types of randomized divide-and-conquer algorithm, depending on whether the size of the sample is rather small or quite large.

TOP-DOWN SAMPLING

Top-down sampling is the most common form of random sampling in computational geometry. It uses a random sample of small, usually constant, size to partition the problem into subproblems. We sketch the technique by giving an algorithm for the computation of the trapezoidal map of a set of segments in the plane.

Given a set S of n line segments with disjoint (relative) interiors, we take a sample $R \subset S$ consisting of r segments, where r is a constant. We compute the trapezoidal map $\mathcal{T}(R)$ of R. It consists of $O(r)$ trapezoids. For every trapezoid $\Delta \in \mathcal{T}(R)$, we determine the ***conflict list*** S_Δ, the list of segments in S intersecting Δ. We construct the trapezoidal map of every set S_Δ recursively, clip it to the trapezoid Δ, and finally glue all these maps together to obtain $\mathcal{T}(S)$.

The running time of this algorithm can be analyzed as follows. Because r is a constant, we can afford to compute $\mathcal{T}(R)$ and the lists S_Δ naively, in time $O(r^2)$ and $O(nr)$ respectively. Gluing together the small maps can be done in time $O(n)$. But what about the recursive calls? If we denote the size of S_Δ by n_Δ, then bounding the n_Δ becomes the key issue here. It turns out that the right intuition is to assume that the n_Δ are about n/r. Assuming this, we get the recursion

$$T(n) \leq O(r^2 + nr) + O(r)T(n/r),$$

which solves to $T(n) = O(n^{1+\epsilon})$, where $\epsilon > 0$ is a constant depending on r. By increasing the value of r, ϵ can be made arbitrarily small, but at the same time the constant of proportionality hidden in the O-notation increases.

The truth is that one cannot really assume that $n_\Delta = O(n/r)$ holds for every trapezoid Δ at the same time. Valid bounds are as follows. For randomly chosen R of size r, we have:

- The ***pointwise bound:*** With probability increasing with r,

$$n_\Delta \leq C\frac{n}{r}\log r, \tag{44.1.1}$$

  for all $\Delta \in \mathcal{T}(R)$, where the constant C does not depend on r and n.

- The ***higher-moments bound:*** For any constant $c \geq 1$, there is a constant $C(c)$ (independent of r and n) such that

$$\sum_{\Delta \in \mathcal{T}(R)} (n_\Delta)^c = C(c)\left(\frac{n}{r}\right)^c |\mathcal{T}(R)|. \tag{44.1.2}$$

In other words, while the maximum n_Δ can be as much as $O((n/r)\log r)$, on the average the n_Δ behave as if they indeed were $O(n/r)$.

Both bounds can be used to prove that $T(n) = O(n^{1+\epsilon})$, with the dependence on ϵ being somewhat better using the latter bound. The difference between the two bounds becomes more marked for larger values of r, as will be detailed below. (For a more general result that subsumes these two bounds, see Theorem 44.5.2.)

The same scheme used to compute $\mathcal{T}(S)$ will also give a data structure for point location in the trapezoidal map. This data structure is a tree, constructed as follows. If the set S is small enough, simply store $\mathcal{T}(S)$ explicitly. Otherwise, take a random sample R, and store $\mathcal{T}(R)$ in the root node. Subtrees are created for every $\Delta \in \mathcal{T}(R)$. These subtrees are constructed recursively, using the sets S_Δ.

By the pointwise bound, the depth of the tree is $O(\log n)$ with high probability, and therefore the query time is also $O(\log n)$. The storage requirement is easily seen to be $O(n^{1+\epsilon})$ as above.

The algorithmic technique described in this section is surprisingly robust. It works for a large number of problems in computational geometry, and for many problems it is the only known approach to solve the problem. It does have two major drawbacks, however.

First, it seems to be difficult to remove the ϵ-term in the exponent, and truly optimal random-sampling algorithms are scarce. If the size r of the sample is a function of n, say $r = n^\delta$, then the extra factor can often be reduced to $\log^c n$. To entirely eliminate this extra factor, one needs to control the total conflict list size using problem-specific insights. Some examples where this has been achieved are listed below as applications.

Second, the practicality of this method remains to be established. If the size of the random sample is chosen too small, then the problem size may not decrease fast enough to guarantee a fast-running algorithm, or even termination. Few papers in the literature calculate this size constant, and so for most applications it remains unclear whether the size of the random sample can be chosen considerably smaller than the problem size in practice.

CUTTINGS

The only use of randomization in the above algorithm was to subdivide the plane into a number of simply-shaped regions Δ, such that every region is intersected by only a few line segments. Such a subdivision is called a *cutting* Ξ for the set X of n segments; if every $\Delta \in \Xi$ intersects at most n/r of the objects in X, it is a $1/r$-*cutting*. Cuttings are interesting in their own right, and have been studied intensively. See Section 47.5 for results on the deterministic construction of efficient cuttings, with useful properties that go beyond those of the simple cutting based on a random sample discussed above. Cuttings form the basis for many algorithms and search structures in computational geometry; see Chapter 40. As a result, many recent geometric divide-and-conquer algorithms no longer explicitly use randomization, and randomized divide-and-conquer has to some extent been replaced by divide-and-conquer based on cuttings.

In practice, however, cuttings may still be constructed most efficiently using random sampling. There are two basic techniques, which we illustrate again using a set X of n line segments with disjoint interiors in the plane.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

- ***ϵ-net based cuttings:*** The easiest way to obtain a $1/r$-cutting is to take a random sample $N \subset X$ of size $O(r \log r)$. If N is a $1/r$-net for the range space (X, Γ) (defined in Section 44.4 and Chapter 47), then the trapezoidal map of N is a $1/r$-cutting of size $O(r \log r)$. If not, we try a different sample.
- ***Splitting the excess:*** The construction based on ϵ-nets can be improved as follows. First take a random sample N of X of size $O(r)$, and compute its trapezoidal map. Every trapezoid Δ may be intersected by $O((n/r) \log r)$ segments. If we take a random sample of these segments, and form their trapezoidal map again (restricted to Δ), the pieces obtained are intersected by at most n/r segments. The size of this cutting is only $O(r)$, which is optimal.

Har-Peled [HP00] investigates the constants achievable for cuttings of lines in the plane.

BOTTOM-UP SAMPLING

In bottom-up sampling, the random sample is so large that the resulting subproblems are small enough to be solved directly. However, it is no longer trivial to compute the auxiliary structures needed to subdivide the problem. We again illustrate with the trapezoidal map.

Given a set S of n line segments, we take a sample R of size $n/2$, and compute the trapezoidal map of R recursively. For every $\Delta \in \mathcal{T}(R)$, we compute the list S_Δ of segments in $S \setminus R$ intersecting Δ. This can be done by locating an endpoint of every segment in $S \setminus R$ in $\mathcal{T}(R)$ and traversing $\mathcal{T}(R)$ from there. If we use a planar point location structure (Section 38.3), this takes time $O(n \log n + \sum_{\Delta \in \mathcal{T}(R)} n_\Delta)$. For every Δ, we then compute the trapezoidal map $\mathcal{T}(S_\Delta)$, and clip it to Δ. This can be done naively in time $O(n_\Delta^2)$. Finally, we glue together all the little maps.

The running time of the algorithm is bounded by the recursion

$$T(n) \leq T(n/2) + O(n \log n) + \sum_{\Delta \in \mathcal{T}(R)} O(n_\Delta^2).$$

The pointwise bound shows that with high probability, $n_\Delta = O(\log n)$ for all Δ. That would imply that the last term in the recursion is $O(n \log^2 n)$. Here, the higher-moments bound turns out to give a strictly better result, as it shows that the expected value of that term is only $O(n)$. The recursion therefore solves to $O(n \log^2 n)$.

Bottom-up sampling has the potential to lead to more efficient algorithms than top-down sampling, because it avoids the blow-up in problem size that manifests itself in the n^ϵ-term in top-down sampling. However, it needs more refined ingredients—as the constructions of $\mathcal{T}(R)$ and the lists S_Δ demonstrate—and therefore seems to apply to fewer problems.

As with top-down sampling, bottom-up sampling can be used for point location. These search structures have the advantage that they can often easily be made dynamic (Section 44.3).

APPLICATIONS

Proofs for the theorems above can be found in the surveys and books cited in Section 44.11, in particular [Cla92, Mul00, BCKO08, Mul93].

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

In the following we list a few advanced applications of geometric divide-and-conquer.

- Computing the diameter of a point set in $\mathbb{R}^3$ [Ram01], achieves optimality by clustering subproblems together to achieve a small boundary between subproblems.
- An optimal data structure for simplex range searching [Cha12], builds a partition tree by refining a cutting on each level.
- A *shallow cutting* is a cutting for only the "shallow" part of a structure, that is, the region of space that lies in few of the objects. Shallow cuttings are used in range searching data structures and can be computed in optimal time [CT16].

44.2 RANDOMIZED INCREMENTAL ALGORITHMS

GLOSSARY

Backwards analysis: Analyzing the time complexity of an algorithm by viewing it as running backwards in time [Sei93].

Conflict graph: A bipartite graph whose arcs represent conflicts (usually intersections) between objects to be added and objects already constructed.

History graph: A directed, acyclic graph that records the history of changes in the geometric structure being maintained. Also known as an ***influence graph***.

Many problems in computational geometry permit a natural computation by an incremental algorithm. Incremental algorithms need only process one new object at a time, which often implies that changes in the geometric data structure remain localized in the neighborhood of the new object.

As an example, consider the computation of the trapezoidal map of a set of line segments (cf. Fig. 38.3.2; for another example, see Section 26.3). To add a new line segment s to the map, one would first identify the trapezoids of the map intersected by s. Those trapezoids must be split, creating new trapezoids, some of which then must be merged along the segment s. All these update operations can be accomplished in time linear in the sum of the number of old trapezoids that are destroyed and the number of new trapezoids that are created during the insertion of s. This quantity is called the ***structural change***.

This results in a rather simple algorithm to compute the trapezoidal map of a set of line segments. Starting with the empty set, we treat the line segments one-by-one, maintaining the trapezoidal map of the set of line segments inserted so far.

However, a general disadvantage of incremental algorithms is that the total structural change during the insertions of n objects, and hence the running time of the algorithm, depends strongly on the order in which the objects are processed. In our case, it is not difficult to devise a sequence of n line segments leading to a total structural change of $\Theta(n^2)$. Even if we know that a good order of insertion exists (one that implies a small structural change), it seems difficult to determine this order beforehand. And this is exactly where randomization can help: we simply treat the n objects in random order. In the case of the trapezoidal map, we will show below that if the n segments are processed in random order, the *expected* structural change in every step of the algorithm is only constant.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

BACKWARDS ANALYSIS

An easy way to see this is via ***backwards analysis***. We first observe that it suffices to bound the number of trapezoids created in each stage of the algorithm. All these trapezoids are incident to the segment inserted in that stage. We imagine the algorithm removing the line segments from the final map one-by-one. In each step, we must bound the number of trapezoids incident to the segment s removed. Now we make two observations:

- The trapezoidal map is a planar graph, with every trapezoid incident to at most 4 segments. Hence, if there are m segments in the current set, the total number of trapezoid-segment incidences is $O(m)$.
- Since the order of the segments is a random permutation of the set of segments, each of the m segments is equally likely to be removed.

These two facts suffice to show that the expected number of trapezoids incident to s is constant. In fact, this number is bounded by the average degree of a segment in a trapezoidal map.

It follows that the expected total structural change during the course of the algorithm is $O(n)$. To obtain an efficient algorithm, however, we need a second ingredient: whenever a new segment s is inserted, we need to identify the trapezoids of the old map intersected by s. Two basic approaches are known to solve this problem: the conflict graph and the history graph.

CONFLICT GRAPH

A conflict graph is a bipartite graph whose nodes are the not-yet-added segments on one side and the trapezoids of the current map on the other side. There is an arc between a segment s and a trapezoid Δ if and only if s intersects Δ, in which case we say that s *is in conflict with* Δ.

It is possible to maintain the conflict graph during the course of the incremental algorithm. Whenever a new segment is inserted, all the conflicts of the newly-created trapezoids are found. This is not difficult, because a segment can only conflict with a newly-created trapezoid if it was previously in conflict with the old trapezoids at the same place. Thus the trapezoids intersected by the new segment s are just the neighbors of s in the conflict graph.

The time necessary to maintain the conflict graph can be bounded by summing the number of conflicts of all trapezoids created during the course of the algorithm. It follows from the higher-moments bound (Eq. 44.1.2) that the average number of conflicts of the trapezoids present after inserting the first r segments—note that these segments form a random sample of size r of S—is $O(n/r)$. Intuitively, we can assume that this is also correct if we look only at the trapezoids that are *created* by the insertion of the rth segment. Since the expected number of trapezoids created in every step of the algorithm is constant, the expected total time is $\sum_{i=1}^{n} O(n/r) = O(n \log n)$.

Note that an algorithm using a conflict graph needs to know the entire set of objects (segments in our example) in advance.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

HISTORY GRAPH

A different approach uses a history graph, which records the history of changes in the maintained structure.

In our example, we can maintain a directed acyclic graph whose nodes correspond to trapezoids constructed during the course of the algorithm. The leaves are the trapezoids of the current map; all inner nodes correspond to trapezoids that have already been destroyed (with the root corresponding to the entire plane). When we insert a segment s, we create new nodes for the newly-created trapezoids, and create a pointer from an old trapezoid to every new one that overlaps it. Hence, there are at most four outgoing pointers for every inner node of the history graph.

We can now find the trapezoids intersected by a new segment s by performing a graph search from the root, using say, depth-first search on the connected subgraph consisting of all trapezoids intersecting s. Note that this search performs precisely the same computations that would have been necessary to maintain the conflict graph during the sequence of updates, but at a different time. We can therefore consider a history graph as a lazy implementation of a conflict graph: it postpones each computation to the moment it is actually needed. Consequently, the analysis is exactly the same as for conflict graphs.

Algorithms using a history graph are ***on-line*** or ***semidynamic*** in the sense that they do not need to know about a point until the moment it is inserted.

ABSTRACT FRAMEWORK AND ANALYSIS

Most randomized incremental algorithms in the literature follow the framework sketched here for the computation of the trapezoidal map: the structure to be computed is maintained while the objects defining it are inserted in random order. To insert a new object, one first has to find a "conflict" of that object (the *location step*), then local updates in the structure are sufficient to bring it up to date (the *update step*). The cost of the update is usually linear in the size of the change in the combinatorial structure being maintained, and can often be bounded using backwards analysis. The location step can be implemented using either a conflict graph or a history graph. In both cases, the analysis is the same (since the actual computations performed are also often identical). To avoid having to prove the same bounds repeatedly for different problems, researchers have defined an axiomatic framework that captures the combinatorial essence of most randomized incremental algorithms. This framework, which uses *configuration spaces*, provides ready-to-use bounds for the expected running time of most randomized incremental algorithms. See Section 44.5.

POINT LOCATION THROUGH HISTORY GRAPH

In our trapezoidal map example, the history graph may be used as a point location structure for the trapezoidal map: given a query point q, find the trapezoid containing q by following a path from the root to a leaf node of the history graph. At each step, we continue to the child node corresponding to the trapezoid containing q.

The search time is clearly proportional to the length of the path. Backwards analysis shows that the expected length of this path is $O(\log n)$ for any fixed query

point. Even stronger, one can show that the maximum length of any search path in the history graph is $O(\log n)$ with high probability.

If point location is the goal, the history graph can be simplified: instead of storing trapezoids, the inner nodes of the graph can denote two different kinds of elementary tests ("Does a point lie to the left or right of another point?" and "Does a point lie above or below a line?"). The final result is then an efficient and practical planar point location structure [Sei91].

This observation can also lead to a somewhat different location step inside the randomized incremental algorithm. Instead of performing a graph search with the whole segment s, point location can be used to find the trapezoid containing one endpoint of s. From there, a traversal of the trapezoidal map allows locating all trapezoids intersected by s.

APPLICATIONS

The randomized incremental framework has been successfully applied to a large variety of problems. We list a number of important such applications. Details on the results can be found in the chapters dealing with the respective area, or in one of the surveys cited in Section 44.11.

- Trapezoidal decomposition formed by segments in the plane, and point location structures for this decomposition (Section 38.3).
- Triangulation of simple polygons: an optimal randomized algorithm with linear running time, and a simple algorithm with running time $O(n \log^* n)$ (Section 30.3).
- Convex hulls of points in d-dimensional space, output-sensitive convex hulls in $\mathbb{R}^3$ (Section 26.3).
- Voronoi diagrams in different metrics, including higher order and abstract Voronoi diagrams (Section 27.3).
- Linear programming in finite-dimensional space (Chapter 49).
- Generalized linear programming: optimization problems that are combinatorially similar to linear programming (Section 49.6).
- Hidden surface removal (Section 33.8 and Chapter 52).
- Constructing a single face in an arrangement of (curved) segments in the plane, or in an arrangement of triangles or surface patches in $\mathbb{R}^3$ (Sections 28.5 and 50.2); computing zones in an arrangement of hyperplanes in $\mathbb{R}^d$ (Section 28.4).

44.3 DYNAMIC ALGORITHMS

DYNAMIC RANDOMIZED INCREMENTAL

Any on-line randomized incremental algorithm can be used as a semidynamic algorithm, a dynamic algorithm that can only perform insertions of objects. The bound on the expected running time of the randomized incremental algorithm then

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

turns into a bound on the *average* running time, under the assumption that every permutation of the input is equally likely. (The relation between the two uses of the algorithms is similar to that between randomized and ordinary Quicksort as mentioned in the Introduction.)

This observation has motivated researchers to extend randomized incremental algorithms so that they can also manage deletions of objects. Then bounds on the average running time of the algorithm are given, under the assumption that the input sequence is a *random update sequence*. In essence, one assumes that for an addition, every object currently not in the structure is equally likely to be inserted, while for a deletion every object currently present is equally likely to be removed (the precise definition varies between authors; see, e.g., [Mul91c, Sch91, Cla92, DMT92]).

Two approaches have been suggested to handle deletions in history-graph based incremental algorithms. The first adds new nodes at the leaf level of the history graph for every deletion. This works for a wide variety of problems and is relatively easy to implement, but after a number of updates the history graph will become "dirty": it will contain elements that are no longer part of the current structure but which still must be traversed by the point-location steps. Therefore, the history graph needs periodic "cleaning." This can be accomplished by discarding the current graph, and reconstructing it from scratch using the elements currently present.

In the second approach, for every deletion the history graph is transformed to the state it would have been had the object never been inserted. The history graph is therefore always "clean." However, in this model deletions are more complicated, and it therefore seems to apply to fewer problems.

DYNAMIC SAMPLING AND GRADATIONS

A rather different approach permits a number of search structures based on bottom-up sampling to be dynamized surprisingly easily. Such a search structure consists of a *gradation* using Bernoulli sampling (Section 44.1): The gradation is a hierarchy of $O(\log n)$ levels. Every object is included in the first level, and is chosen independently to be in the second level with probability $\frac{1}{2}$. Every object in the second level is propagated to level 3 with probability $\frac{1}{2}$, and so forth. Whenever an object is added to or removed from the current set, the search structure is updated to the proper state. When adding an object, it suffices to flip a coin at most $\log n$ times to determine where to place the object. Using this technique, it is possible to give high-probability bounds on the search time and sometimes also on the update time [Mul91a, Mul91b, Mul93].

44.4 RANGE SPACES

"Pointwise bounds" of the form in Equation 44.1.1 can be proved in the axiomatic framework of range spaces, which then leads to immediate application to a wide variety of geometric settings. Chapter 47 also discusses range spaces, but calls them (abstract) set systems.

GLOSSARY

Range space: A pair (X, Γ), with X a universe (possibly infinite), and Γ a family of subsets of X. The elements of Γ are called ***ranges***. Typical examples of range spaces are of the form $(\mathbb{R}^d, \Gamma)$, where Γ is a set of geometric figures, such as all line segments, halfspaces, simplices, balls, etc.

Shattered: A set $A \subseteq X$ is shattered if every subset A' of A can be expressed as $A' = A \cap \gamma$, for some range $\gamma \in \Gamma$.

In the range space $(\mathbb{R}^2, \mathcal{H})$, where $\mathcal{H}$ is the set of all closed halfplanes, a set of three points in convex position is shattered. However, no set of four points is shattered. See Figure 44.4.1: whether the point set is in convex position or not, there always is a subset (encircled) that cannot be expressed as $A \cap h$ for any halfplane h.

FIGURE 44.4.1
No set of four points can be shattered by halfplanes.

In the range space $(\mathbb{R}^2, \mathcal{C})$, where $\mathcal{C}$ is the set of all convex polygons, any set of points lying on a circle is shattered.

Vapnik-Chervonenkis dimension (VC-dimension): The VC-dimension of a range space (X, Γ) is the smallest integer d such that there is no shattered subset $A \subseteq X$ of size $d + 1$. If no such d exists, the VC-dimension is said to be infinite.

Range spaces $(\mathbb{R}^d, \Gamma)$, where Γ is the set of line segments, of simplices, of balls, or of halfspaces, have finite VC-dimension. For example, the range space $(\mathbb{R}^2, \mathcal{H})$ has VC-dimension 3. The range space $(\mathbb{R}^2, \mathcal{C})$, however, has infinite VC-dimension.

Shatter function: For a range space (X, Γ), the shatter function $\pi_\Gamma(m)$ is defined as

$$\pi_\Gamma(m) = \max_{A \subset X, |A|=m} |\{A \cap \gamma \mid \gamma \in \Gamma\}| .$$

If the VC-dimension of the range space is infinite, then $\pi_\Gamma(m) = 2^m$. Otherwise the shatter function is bounded by $O(m^d)$, where d is the VC-dimension. (So the shatter function of any range space is either exponential or polynomially bounded.) If the shatter function is polynomial, the VC-dimension is finite. The order of magnitude of the shatter function is not necessarily the same as the VC-dimension; for instance, the range space $(\mathbb{R}^2, \mathcal{H})$ has VC-dimension 3 and shatter function $O(m^2)$. Since the VC-dimension is often difficult to compute, some authors have defined the *VC-exponent* as the order of magnitude of the shatter-function.

ϵ-net: A subset $N \subseteq X$ is called an ϵ-net for the range space (X, Γ) if $N \cap \gamma \neq \emptyset$ for every $\gamma \in \Gamma$ with $|\gamma|/|X| > \epsilon$ (here, $\epsilon \in [0, 1)$ and X is finite). It is often more convenient to write $1/r$ for ϵ, with $r > 1$.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

ϵ-approximation: A subset $A \subseteq X$ is called an ϵ-approximation for the range space (X, Γ) if, for every $\gamma \in \Gamma$, we have

$$\left| \frac{|A \cap \gamma|}{|A|} - \frac{|\gamma|}{|X|} \right| \leq \epsilon \, .$$

An ϵ-approximation is also an ϵ-net, but not necessarily vice versa.

Relative (p, ϵ)-approximation: An ϵ-approximation provides an absolute error on the term $|\gamma|/|X|$. In many applications we would prefer a relative error instead. This cannot be achieved by a small approximation, so a relative (p, ϵ)-approximation guarantees a relative error only when $|\gamma|/|X| \geq p$, otherwise an absolute error of ϵp [HPS11].

ϵ-NETS AND ϵ-APPROXIMATIONS

The pointwise bound translates into the abstract framework of range spaces as follows:

THEOREM 44.4.1

Let (X, Γ) be a range space with X finite and of finite VC-dimension d. Then a random sample $R \subset X$ of size $C(d) r \log r$ is a $1/r$-net for (X, Γ) with probability whose complement to 1 is polynomially small in r, where $C(d)$ is a constant that depends only on d.

This theorem forms the basis for "traditional" randomized divide-and-conquer algorithms, such as the one for the trapezoidal map of line segments sketched in Section 44.1. The pointwise bound used there follows from the theorem. Consider the range space (S, Γ), where $\Gamma := \{\gamma(\Delta) \mid \Delta \text{ an open trapezoid}\}$, and $\gamma(\Delta)$ is the set of all segments in S intersecting Δ. The VC-dimension of this range space is finite. The easiest way to see this is by looking at the shatter function. Consider a set of m line segments. Extend them to full lines, pass $2m$ vertical lines through all endpoints, and look at the arrangement of these $3m$ lines. Clearly, for any two trapezoids Δ and Δ' whose corners lie in the same faces of this arrangement we have $\gamma(\Delta) = \gamma(\Delta')$. Consequently, there are at most $O(m^8)$ different ranges, and that crudely bounds the shatter function as $O(m^8)$. Thus the VC-dimension is finite and Theorem 44.4.1 applies: with probability increasing rapidly with r, the sample R of size r is an ϵ-net for S with $\epsilon = \Omega((1/r) \log r)$. Assume this is the case, and consider some trapezoid $\Delta \in \mathcal{T}(R)$. The interior of Δ does not intersect any segment in R, so by the property of ϵ-nets, the range $\gamma(\Delta)$ can intersect at most ϵn segments of S. And so we have $n_\Delta = O((n/r) \log r)$.

The construction of ϵ-nets has been so successfully derandomized that ϵ-nets now are used routinely in deterministic algorithms. Indeed, Chapter 47, which covers ϵ-nets and ϵ-approximations in detail, hardly mentions randomization.

For general range spaces, the bound $O(r \log r)$ in Theorem 44.4.1 is the best possible. For some geometrically defined spaces the bound has been improved, see Section 47.4. For others, including the case of halfspaces in more than three dimensions, a lower bound of $\Omega(r \log r)$ has been shown [PT13].

An ϵ-approximation serves as a coreset for density estimation, see Section 48.2.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

44.5 CONFIGURATION SPACES

The framework of configuration spaces is somewhat more complicated than range spaces, but facilitates proving higher-moment bounds as in Equation 44.1.2. Terminology, axiomatics, and notation vary widely between authors. Note that the term "configuration space" is used in robotics with a different meaning (see Chapters 50 and 51).

GLOSSARY

Configuration space: A four-tuple $(X, \mathcal{T}, D, K)$. X is a finite set of geometric objects (the universe of size n). $\mathcal{T}$ is a mapping that assigns to every subset $S \subseteq X$ a set $\mathcal{T}(S)$; the elements of $\mathcal{T}(S)$ are called ***configurations***. $\Pi(X) := \bigcup_{S \subseteq X} \mathcal{T}(S)$ is the set of all configurations occurring over some subset of X. D and K assign to every configuration $\Delta \in \Pi(X)$ subsets $D(\Delta)$ and $K(\Delta)$ of X. Elements of the set $D(\Delta)$ are said to ***define*** the configuration (they are also called ***triggers***) and the elements of the set $K(\Delta)$ are said to ***kill*** the configuration (they are also said to be in ***conflict*** with the configuration and are sometimes called ***stoppers***).

Conflict size of Δ: The number of elements of $K(\Delta)$.

We will require the following axioms:

(i) The number $d = \max\{\, |D(\Delta)| \;\mid\; \Delta \in \Pi(X)\}$ is a constant (called the ***maximum degree*** or the ***dimension*** of the configuration space). Moreover, the number of configurations sharing the same defining set is bounded by a constant.

(ii) For any $\Delta \in \mathcal{T}(S)$, $D(\Delta) \subseteq S$ and $S \cap K(\Delta) = \emptyset$.

(iii) If $\Delta \in \mathcal{T}(S)$ and $D(\Delta) \subseteq S' \subseteq S$, then $\Delta \in \mathcal{T}(S')$.

(iii′) If $D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset$, then $\Delta \in \mathcal{T}(S)$.

Note that axiom (iii) follows from (iii′); see below.

EXAMPLES

1. *Trapezoidal map.* The universe X is a set of segments in the plane, and $\mathcal{T}(S)$ is the set of trapezoids in the trapezoidal map of S. The defining set $D(\Delta)$ is the set of segments that are necessary to define Δ (at most four segments suffice, so $d = 4$), and the killing set $K(\Delta)$ is the set of segments that intersect the trapezoid. It is easy to verify that conditions (i), (ii), (iii), (iii′) all hold.

2. *Delaunay triangulation.* X is a set of points in the plane (assume that no four points lie on a circle), and $\mathcal{T}(S)$ is the set of triangles of the Delaunay triangulation of S. $D(\Delta)$ consists of the vertices of triangle Δ (so $d = 3$), while $K(\Delta)$ is the set of points lying inside the circumcircle of the triangle. Again, axioms (i), (ii), (iii), (iii′) all hold.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

3. *Convex hulls in 3D.* The universe X is a set of points in 3D (assume that no four points are coplanar), and $\mathcal{T}(S)$ is the set of facets of the convex hull of S. The defining set of a facet Δ is the set of its vertices ($d = 3$), and the killing set is the set of points lying in the outer open halfspace defined by Δ. Note that there can be two configurations sharing the same defining set. Again, axioms (i)–(iii′) all hold.

4. *Single cell.* The universe X is a set of possibly intersecting segments in the plane, and $\mathcal{T}(S)$ is the set of trapezoids in the trapezoidal map of S that belongs to the cell of the line segment arrangement containing the origin (Figure 44.5.1). The defining and killing sets are defined as in the case of the trapezoidal map of the whole arrangement above. In this situation, axiom (iii′) does not hold. Whether or not a given trapezoid appears in $\mathcal{T}(S)$ depends on segments other than the ones in $D(\Delta) \cup K(\Delta)$. Axioms (i), (ii), (iii) are nevertheless valid.

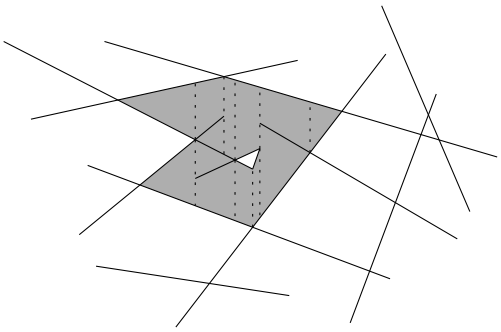

FIGURE 44.5.1
A single cell in an arrangement of line segments.

5. *LP-type problems.* LP-type problems and *violator spaces* are generalizations of linear programming. Consider as an example the problem of finding the smallest enclosing disk for a set of points in the plane. The universe X is the set of points, and $\mathcal{T}(S)$ is the unique smallest enclosing disk. If we assume general position, then a disk is defined by two or three points, its killing set is the set of points lying outside the disk. Axioms (i)–(iii′) hold.

   LP-type problems in general are defined as a set H of *constraints* and a mapping assigning a "value" $w(S)$ to each subset $S \subset H$, the goal is to find a minimal subset $B \subset H$ (a *basis*) such that $w(B) = w(H)$, see Section 49.6 for details. LP-type problems of constant combinatorial dimension (that is, the size of bases is bounded by a constant) satisfy our axioms, but degeneracies (many bases with the same value) need to be handled with care.

6. *Counterexample.* Let X be a set of line segments, and let $\mathcal{T}(S)$ be a decomposition of the arrangement that is obtained by drawing vertical extensions for faces with an even number of edges, and horizontal extensions for faces with an odd number of edges. Axioms (i) and (ii) hold, but neither (iii) nor (iii′) is satisfied.

Note that when (ii) and (iii′) both hold, then $\Delta \in \mathcal{T}(S)$ if and only if $D(\Delta) \subseteq S$ and $K(\Delta) \cap S = \emptyset$. In other words, the mapping $\mathcal{T}$ is then completely defined by the functions D and K. In fact, in the first three examples we can decide from local

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

information alone whether or not a configuration appears in $\mathcal{T}(S)$. For instance, a triangle Δ is in the Delaunay triangulation of S if and only if the vertices of Δ are in S, and no point of S lies in the circumcircle of Δ.

As mentioned above, axiom (iii) follows from (iii′), but not conversely. Axiom (iii) requires a kind of monotonicity: if Δ occurs in $\mathcal{T}(S)$ for some S, then we cannot destroy it by removing elements from S unless we remove some element in $D(\Delta)$.

We may say that the configuration spaces of the first three examples are defined *locally* and *canonically*. The fourth example is *canonical*, but *nonlocal*. The last example is not canonical and cannot be treated with the methods described here. (Fortunately, this is an artificial example with no practical use—but see the open problems below.)

HIGHER-MOMENTS AND EXPONENTIAL DECAY LEMMA

The higher-moments bound for configuration spaces generalizes the bound for trapezoidal maps, Equation 44.1.2:

THEOREM 44.5.1 *Higher-moments bound*

Let $(X, \mathcal{T}, D, K)$ be a configuration space satisfying axioms (i), (ii), (iii), and let R be a random sample of X of size r. For any constant c, we have

$$E\left[\sum_{\Delta\in\mathcal{T}(R)} |K(\Delta)|^c\right] = O((n/r)^c E[|\mathcal{T}(R)|]).$$

(Technically, rather than R, a sample R' of size $\lfloor r/2\rfloor$ should appear on the right, but $E[|\mathcal{T}(R')|] = O(E[|\mathcal{T}(R)|])$ in all cases of interest). In other words, as far as the cth-degree average is concerned, the conflict size behaves as if it were $O(n/r)$, instead of $O((n/r)\log r)$ from the pointwise bound.

Let $(X, \mathcal{T}, D, K)$ and R be as in Theorem 44.5.1. For any natural number t, we define $\mathcal{T}_t(R)$ to be the subset of configurations of $\mathcal{T}(R)$ whose conflict size exceeds the "natural" value n/r by at least the factor t:

$$\mathcal{T}_t(R) := \{\Delta \in \mathcal{T}(S) \mid |K(\Delta)| \geq tn/r\}.$$

The following exponential-decay lemma [AMS98] states that the number of such configurations decreases exponentially with t:

THEOREM 44.5.2 *Exponential decay lemma*

Let $(X, \mathcal{T}, D, K)$ be a configuration space satisfying axioms (i), (ii), (iii), and let R be a random sample of X of size r. For any t with $1 \leq t \leq r/d$ (where d is as in axiom (i)), we have

$$E[|\mathcal{T}_t(R)|] = O(2^{-t}) \cdot E[|\mathcal{T}(R')|],$$

where $R' \subseteq X$ denotes a random sample of size $\lfloor r/t\rfloor$.

The exponential decay lemma implies both the higher-moments bound, by adding over t, and the pointwise bound, by Markov's inequality.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

RANDOMIZED INCREMENTAL CONSTRUCTION

Many, if not most, randomized incremental algorithms in the literature can be analyzed using the configuration space framework. Given the set X, the goal of the randomized incremental algorithm is to compute $\mathcal{T}(X)$. This is done by maintaining $\mathcal{T}(X^i)$, for $1 \leq i \leq n$, where $X^i = \{x_1, x_2, \ldots, x_i\}$ and the x_i form a random permutation of X.

To bound the number of configurations created during the insertion of x_i into X^{i-1}, we observe that by axiom (iii) these configurations are exactly those $\Delta \in \mathcal{T}(X^i)$ with $x_i \in D(\Delta)$. The expected number of these can be bounded by

$$\frac{d}{i} E[|\mathcal{T}(X^i)|]$$

using backwards analysis. Here, d is the maximum degree of the configuration space.

The expected total change in the conflict graph or history graph can be bounded by summing $|K(\Delta)|$ over all Δ created during the course of the algorithm. Using axioms (i) to (iii′), we can derive the following bound:

$$\sum_{i=1}^{n} d^2 \frac{n-i}{i} \frac{E[|\mathcal{T}(X^i)|]}{i}.$$

(The exact form of this expression depends on the model used.) The book [Mul93] treats randomized incremental algorithms systematically using the configuration space framework (assuming axiom (iii′)).

LAZY RANDOMIZED INCREMENTAL CONSTRUCTION

In problems that have nonlocal definition, such as the computation of a single cell in an arrangement of segments, single cells in arrangements of surface patches, or zones in arrangements, the update step of a randomized incremental construction becomes more difficult. Besides the local updates in the neighborhood of the newly inserted object, there may also be global changes. For instance, when a line segment is inserted into an arrangement of line segments, it may cut the single cell being computed into several pieces, only one of which is still interesting. The technique of lazy randomized incremental construction [BDS95] deals with these problems by simply postponing the global changes to a few "clean-up" stages. Since the setting of all these problems is nonlocal, the analysis uses only axioms (i), (ii), (iii).

OPEN PROBLEM

The canonical framework of randomized incremental algorithms sketched above is sometimes too restrictive. For instance, to make a problem fit into the framework, one often has to assume that objects are in general position. While many algorithms could deal with special cases (e.g., four points on a circle in the case of Delaunay triangulations) directly, the analysis does not hold for those situations, and one has to resort to a symbolic perturbation scheme to save the analysis. Can a more relaxed framework for randomized incremental construction be given [Sei93]?

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

44.6 DERANDOMIZATION TECHNIQUES

Even when an efficient randomized algorithm for a problem is known, researchers still find it worthwhile to obtain a deterministic algorithm of the same efficiency. The reasons for doing this are varied, from scientific curiosity (what is the real power of randomness?), to practical reasons (truly random bits are quite expensive), to a preference for "deterministicity" that may not be strictly rational. Sometimes a deterministic algorithm for a given problem may be obtained by "simulating" or "derandomizing" a randomized algorithm. Derandomization has turned out to be a powerful theoretical tool: for several problems the only known worst-case optimal deterministic algorithm has been obtained by derandomization. The most famous example is computing the convex hull of n points in d-dimensional space (Section 26.3).

General derandomization techniques can be used to produce a deterministic counterpart of random sampling in both configuration spaces and range spaces. As a result, it is possible to obtain in polynomial time a sample that satisfies the higher-moment bound, or that is a net or an approximation. Taking advantage of separability and composition properties of approximations, these constructions can be made efficient. In most applications, deterministic sampling is the base of a deterministic divide-and-conquer algorithm or data structure, which is almost as efficient as the randomized counterpart.

On the other hand, incremental algorithms are considerably harder to derandomize: the convex hull algorithm mentioned above is essentially the only successful case. The problem is that in an incremental algorithm each insertion must be "globally good," while in the divide-and-conquer case, items are chosen locally in a neighborhood that shrinks as the algorithm progresses. In some cases, such as linear programming, a derandomized divide-and-conquer approach leads to a deterministic algorithm with better dependency on the dimension than previously known methods (prune-and-search), but there still remains a large gap with respect to the best randomized algorithm (which is an incremental one).

METHOD OF CONDITIONAL PROBABILITIES

The ***method of conditional probabilities*** (also called the ***Raghavan-Spencer method***) [Spe87, Rag88] implements a binary search of the probability space to determine an event with the desired properties (guaranteed by a probabilistic analysis). Given a configuration space $(X, \mathcal{T}, D, K)$, the goal is to obtain a random sample of size (approximately) r that satisfies the higher-moments bound. Let $X = \{x_1, \ldots, x_n\}$ and Ω be the probability space on $\{0,1\}^n$, and consider the probability distribution on Ω induced by selecting each component equal to 1 independently with probability $p = r/n$ (for convenience, we use Bernoulli sampling). Let $F : \Omega \to \mathbb{R}$ be the random variable that assigns to the vector $(q_1, \ldots, q_n)$ the value $\sum_{\Delta \in \mathcal{T}(R)} f(|K(\Delta) \cap X|)$, where $x_i \in R$ iff $q_i = 1$, and $f(x) = x^k$ (for the kth moment; using $f(x) = e^{c(r/n)x}$ with an appropriate constant c, one can achieve the exponential decay bound). We know that $\mathrm{E}[F] \leq M$ with $M = Cf(n/r)t(r)$, where $t(r)$ is an upper bound for $\mathrm{E}[|\mathcal{T}(R)|]$. The method is based on the following

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

relation, for $0 \leq i < n$:

$$\begin{aligned}
&\mathrm{E}[F|q_1 = v_1, \ldots q_i = v_i] \\
&\quad = \quad p \cdot \mathrm{E}[F|q_1 = v_1, \ldots q_i = v_i, q_{i+1} = 1] + \\
&\qquad (1-p) \cdot \mathrm{E}[F|q_1 = v_1, \ldots q_i = v_i, q_{i+1} = 0] \\
&\quad \geq \quad \min\{\mathrm{E}[F|q_1 = v_1, \ldots q_i = v_i, q_{i+1} = 1], \mathrm{E}[F|q_1 = v_1, \ldots q_i = v_i, q_{i+1} = 0]\}
\end{aligned}$$

If these conditional expectations can be computed *efficiently*, then this implies an efficient procedure to select v_{i+1} so that

$$\mathrm{E}[F|q_1 = v_1, \ldots q_i = v_i] \geq \mathrm{E}[F|q_1 = v_1, \ldots q_i = v_i, q_{i+1} = v_{i+1}].$$

Iterating this procedure, one finally obtains a solution $(v_1, \ldots, v_n)$ that satisfies the probabilistic bound

$$M \geq \mathrm{E}[F] \geq \mathrm{E}[F|q_1 = v_1, \ldots q_n = v_n].$$

If the locality property holds, the conditional probabilities involved can indeed be computed in polynomial time: Let $X_i = \{x_1, x_2, \ldots, x_i\}$ and $R_i = \{x_j \in X : q_j = 1, j \leq i\}$, then $\mathrm{E}[F|q_1 = v_1, \ldots q_i = v_i]$ is equal to

$$\begin{aligned}
&\sum_{\Delta \in \Pi(X)} \Pr\{\Delta \in \mathcal{T}(R)|q_1 = v_1, \ldots q_i = v_i\} f(|K(\Delta) \cap X|) \\
= &\sum_{\Delta \in \Pi(X): D(\Delta) \cap X_i \subseteq R_i, K(\Delta) \cap R_i = \emptyset} p^{|D(\Delta) \setminus S_i|} (1-p)^{|K(\Delta) \cap (X \setminus X_i)|} f(|K(\Delta) \cap X|),
\end{aligned}$$

which can be approximated with sufficient accuracy. Similarly, $(1/r)$-nets and $(1/r)$-approximations of sizes $O(r \log r)$ and $O(r^2 \log r)$ can be computed in polynomial time, see Chapter 47.

k-WISE INDEPENDENT DISTRIBUTIONS

The method of conditional probabilities is highly sequential. An approach that is more suitable for parallel algorithms is to construct a probability space of polynomial size, and to execute the algorithm on each vector of this space. This is possible, for example, when the variables q_i need only be k-wise independent rather than being fully independent: for any indices $i_1, \ldots, i_k$, and 0-1 values, $v_1, \ldots, v_k$,

$$\Pr\{q_{i_1} = v_1, \ldots, q_{i_k} = v_k\} = \Pi_{j=1}^{k} \Pr\{q_{i_j} = v_{i_j}\} = \Pi_{j=1}^{k} p^{v_j} (1-p)^{1-v_j}.$$

A probability space and distribution of size $O(n^k)$ with such k-wise independence can be computed effectively [Jof74, KM97]. Let $\rho \geq n$ be a prime number and suppose that $p_1, \ldots, p_n \in [0,1]$ satisfy $p_i = j_i/\rho$, for some integers j_i. Define a probability space with at most n^k points, as follows. For each $\langle a_0, a_1, \ldots, a_{k-1} \rangle$ in $\{0, 1, \ldots, \rho - 1\}^k$, let

$$X_i = a_0 + a_1 i + a_2 i^2 + \cdots + a_{k-1} i^{k-1} \bmod \rho,$$

for $1 \leq i \leq n$, assign probability $1/\rho^k$ and associate the vector $\langle Y_1, \ldots, Y_n \rangle$ where $Y_i = 1$ if $X_i \in \{0, 1, \ldots, j_i - 1\}$ and $Y_i = 0$ otherwise. The 0-1 probability space

defined by the vectors $\langle Y_1, \ldots, Y_n \rangle$ is a k-wise independent 0-1 probability space for $p_1, \ldots, p_n$. With this construction, arbitrary probabilities can be approximated (within a factor of 2) by an appropriate choice of ρ. Using a larger space of size $O(n^{2k})$, arbitrary probabilities can be achieved exactly [KM97].

For some randomized algorithms one can show that they still work under k-wise independency for an appropriate k. For example, a quasi-random permutation with k-wise independence suffices for the randomized incremental approach to work [Mul96] (thus $O(\log n)$ random bits suffice rather than the $\Omega(n \log n)$ bits needed to define a fully random permutation). To verify that k-wise independence suffices, a tail inequality under k-independence is used [SSS95, BR94]. Let $q_1, \ldots, q_n$ be a sequence of k-wise independent random variables in $\{0, 1\}$, with $k \geq 2$ even, let $Q = \sum_{i=1}^{n} q_i$, $\mu = \mathrm{E}[Q]$ and assume that $\mu \geq k$, then

$$\Pr\{Q = 0\} < \frac{C_k}{\mu^{k/2}} \tag{44.6.1}$$

where C_k is a constant depending on k. Let R be a $2k$-wise independent random sample from X with uniform probability p. For $\Delta \in \Pi(X)$, note that (no independence assumption needed)

$$\Pr\{D(\Delta) \subseteq R, K(\Delta) \cap R = \emptyset\} = \Pr\{D(\Delta) \subseteq R\} \cdot \Pr\{K(\Delta) \cap R = \emptyset | D(\Delta) \subseteq R\}.$$

Let d be an upper bound on $|D(\Delta)|$. The first factor can be computed using $2k$-wise independence assuming $2k \geq d$:

$$\Pr\{D(\Delta) \subseteq R\} = p^{|D(\Delta)|}.$$

To upper bound the second factor, let $t_\Delta = p|K(\Delta)|$; then using the tail bound above, for $t_\Delta \geq k$:

$$\Pr\{K(\Delta) \cap R = \emptyset | D(\Delta) \subseteq R\} \leq \frac{C}{t_\Delta^{k-d/2}},$$

since after fixing $D(\Delta) \subseteq R$, the remaining random variables are still $(2k - d)$-wise independent. Choosing k so that $c \leq k - d/2 + 2$, one can verify that the cth moment bound holds. Similarly, $1/r$-nets and $1/r$-approximations with sizes $O(rn^\delta)$ and $O(r^2 n^\delta)$ can be computed in polynomial time, where $\delta = O(1/k)$. It does not seem possible, however, to achieve the exponential decay bound with a limited-independence space of polynomial size.

For fixed k, the size of the space can be reduced if a certain deviation from k-wise independence is allowed [NN93]. Furthermore, the approach of testing all the vectors in the probability space can be combined with the approach of performing a binary search so that even a space of superpolynomial size is usable [MNN94, BRS94]. Still, these approaches do not lead to the exponential decay bound, or to nets or approximations of size matching the probabilistic analysis.

CONSTRAINT-BASED PROBABILITY SPACES

An alternative approach that is implementable in parallel constructs a probability distribution tailored to a particular algorithm and even to a specific input [Nis92, KM94, KK97, MRS01], leading to smaller probability spaces. The approach models

the sampling process using ***randomized finite automata*** (RFA), and fools the automaton using a probability distribution D_n with support of size E_0 that depends polynomially on the error and on the size of the problem. Once the probability distribution has been constructed, it is only a matter of testing the algorithm for each point in D_n.

For each configuration Δ we construct an RFA M_Δ as follows: It consists of $n+1$ levels $N_{\Delta,j}$, $0 \le j \le n$, each with two states $\langle j, \text{Yes}\rangle$ and $\langle j, \text{No}\rangle$, with transitions that reflect whether $\Delta \in \mathcal{T}(R)$: $\langle j-1, \text{No}\rangle$ is always connected to $\langle j, \text{No}\rangle$; if $x_j \in D(\Delta)$ then $\langle j-1, \text{Yes}\rangle$ is connected to $\langle j, \text{Yes}\rangle$ under $q_j = 1$ and to $\langle j, \text{No}\rangle$ under $q_j = 0$; if $x_j \in K(\Delta)$ then $\langle j-1, \text{Yes}\rangle$ is connected to $\langle j, \text{Yes}\rangle$ under $q_j = 0$ and to $\langle j, \text{No}\rangle$ under $q_j = 1$; if $x_j \notin D(\Delta) \cup K(\Delta)$ then $\langle j-1, \text{Yes}\rangle$ is connected to $\langle j, \text{Yes}\rangle$, and $\langle j-1, \text{No}\rangle$ is connected to $\langle j, \text{No}\rangle$, in either case. D_n is determined by a recursive approach in which the generic procedure fool(l, l') constructs a distribution that fools the transition probabilities between level l and l' in *all* the RFAs as follows. It computes, using fool(l, l'') and fool$(l,'' l')$ recursively, distributions D_1 and D_2, each of size at most $E_0(1+o(1))$, that fool the transitions between states in levels l and $l'' = \lfloor (l+l')/2 \rfloor$, and between states in levels l'' and l; a procedure reduce$(D_1 \times D_2)$ then combines D_1 and D_2 into a distribution D of size at most $E_0(1+o(1))$ that fools the transitions between states in levels l and l' in all the RFAs. Let $\tilde{D} = D_1 \times D_2$ be the product distribution with support$(\tilde{D}) = \{w_1 w_2 : w_i \in \text{support}(D_i)\}$ and $\Pr_{\tilde{D}}\{w_1 w_2\} = \Pr_{D_1}\{w_1\}\Pr_{D_2}\{w_2\}$, a randomized version of reduce is to retain each $w \in \tilde{D}$ with probability $q(w) = E_0/|\text{support}(\tilde{D})|$ into support(D) with $\Pr_D\{w\} = \Pr_{\tilde{D}}\{w\}/q(w)$. Thus, for all pairs of states s, t in the RFAs the transition probabilities are preserved in expectation:

$$\mathrm{E}[\Pr_D\{s \to t\}] = \sum_{w\,:\, s \xrightarrow{w} t} \frac{\Pr_{\tilde{D}}\{w\}}{q(w)} q(w) = \Pr_{\tilde{D}}\{s \to t\}, \tag{44.6.2}$$

where the sum is over all the strings w that lead from state s to state t. This selection also implies that the expected size of support(D) is $\sum_w q(w) = E_0$. This randomized combining can be derandomized using a 2-wise independent probability space. The bottom of the recursion is reached when the number of levels between l and l' is at most $\log E_0$, and then the distribution (of size E_0) is implemented by $\log E_0$ unbiased bits. E_0 depends polynomially on $1/\delta$, where δ is the relative error that is allowed for the transition probabilities. Taking δ as a small constant suffices to obtain a constant approximation of the moment bounds.

APPLICATIONS

Some interesting examples for which optimal deterministic algorithms have been obtained using derandomization are the following:

- *ϵ-nets and ϵ-approximations:* See Chapter 47.
- *Convex hulls:* The only optimal deterministic algorithm for the computation of the convex hull of n points in $\mathbb{R}^d$ space is the derandomization of a randomized-incremental algorithm. The reader is referred to [BCM99] for the details (this reference is much more readable than the original paper [Cha93]) (Chapter 26).
- *Output-sensitive convex hull in $\mathbb{R}^d$:* An optimal algorithm for $d = 3$ was obtained using derandomization [CM95]; afterward a surprisingly simple solution avoiding derandomization was found [Cha96].

- *Diameter of a point set in* $\mathbb{R}^3$*:* After a sequence of improvements, an optimal algorithm using derandomization was found [Ram01]. Currently, the best solution that avoids derandomization has a running time with an extra $\log n$ factor [Bes01].
- *Linear programming:* In $\mathbb{R}^d$, the best deterministic solution is achieved through derandomization and has running time $O(C_d n)$ with $C_d = \exp(O(d \log d))$ [CM96, Cha16]; in contrast, it is possible to achieve $C_d \approx \exp(\sqrt{d})$ with randomization [MSW96] (Chapter 49).
- *Segment intersection:* A first algorithm for reporting intersections between n line segments in $O(n)$ space and optimal time used derandomization [AGR95], followed shortly by a relatively simple algorithm that avoids derandomization [Bal95]. Optimal parallel algorithms have been obtained with derandomization and have not been matched by other approaches.

OPEN PROBLEMS

Is derandomization truly necessary to obtain an optimal algorithm in cases such as the convex hull in d dimensions or the diameter in dimension 3?

44.7 OPTIMIZATION

In geometric optimization we seek to optimize some measure on a geometric structure that satisfies given constraints, such as the length of a tour visiting given points, or the area of a convex container into which given shapes can be translated.

PARAMETRIC SEARCH AND RANDOMIZATION

Parametric search is a technique that allows us to take a decision algorithm for the problem (that is, an algorithm that takes a parameter r and tells us if a solution of quality better than r exists), and to convert it into an algorithm solving the optimization problem. Several geometric examples can be found in [CEGS93, AST94].

In many applications of parametric searching, it can be considered as a search among the vertices of an arrangement, where each vertex determines a critical value of the parameter r. If we could compute these vertices efficiently, we could use the decision algorithm to perform a binary search—but the arrangement is too large to be built explicitly.

If we can randomly sample vertices of some range of this (implicit) arrangement, then parametric search can be replaced by randomization to obtain a much simpler algorithm. A classic example is the *slope selection* problem [Mat91].

Alternatively, we may be able to guide the search using an appropriate cutting of this arrangement, obtaining a deterministic algorithm that is much simpler than using parametric search [BC98].

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

CHAN'S TECHNIQUE

Another randomized alternative to parametric search is a simple technique by Chan [Cha99]. Again, it assumes that we have an algorithm to solve the decision version of the problem, and in addition requires that we can split the problem into a constant number k of subproblems smaller by a constant fraction. Then the optimization problem can be solved as follows: Split the problem P into k subproblems $P_1, P_2, \ldots, P_k$, pick a random permutation of the P_i, and set r to the solution for P_1. For $i \in \{2, \ldots, k\}$, use the decision algorithm to determine if the solution for P_i is better than r. If it is, recursively compute the solution for P_i and set r to this solution.

The final value of r is the optimal solution for P, the algorithm also determines a constant-size subset that determines this optimum. If the running time of the decision algorithm is at least polynomial, then the expected running time of this procedure asymptotically matches the running time of the decision algorithm.

A simple example is determining the closest pair, or more generally the b-tuple minimizing some measure, in a given set of points P. Indeed, we can arbitrarily partition P into $b+1$ sets $Q_1, \ldots, Q_{b+1}$ of roughly equal size, and define the subproblem $P_i = P \setminus Q_i$. Then $k = b+1$ and each subproblem's size is $b/(b+1)$ of the original problem's size.

This technique was generalized for certain families of implicit LP-type problems [Cha04].

44.8 MONTE CARLO ALGORITHMS

Monte Carlo algorithms, that is, algorithms that are allowed to fail or report an incorrect answer with a small probability, used to be uncommon in computational geometry, but started to appear more frequently since the early 2000s (Chapter 48 also mentions several Monte Carlo algorithms).

A typical Monte Carlo algorithm in computational geometry takes a random sample R of some input set S, and proceeds under the assumption that R is an ϵ-approximation for S. This is true with high probability, but if the assumption fails, an incorrect result can be returned.

As an example, consider the problem of computing, given a simple polygon P, a point p inside P that maximizes the area of P visible from p. A naive approach would simply try many random points $p \in P$, compute their visibility region, and choose the best one. However, the probability of finding a good point can be arbitrarily small, and we can do better by using an ϵ-approximation to estimate the visibility region of a point [CEHP07]: We uniformly sample a set S of points from P. We compute the visibility region $V(s)$ of each point $s \in S$, take the arrangement of all these regions, and choose a point p^* in the most heavily covered cell of this arrangement.

This point p maximizes the number of points in S visible from p. Since with high probability, S is an ϵ-approximation for the points in P, that is, the number of points of S visible from a point p is an estimate for the area of P visible from p, this implies that the area seen by p^* is a good approximation for the largest area that any point in P can see.

The technique can be improved by observing that we do not actually need an

ϵ-approximation—we need that S is a good approximation only for the vertices of the arrangement of the visibility regions $V(s)$. We can also make the size of S dependent on the value of the optimal solution, see [CEHP07].

Similar ideas have been used in shape matching [CEHP07, ASS10, AS12]. *Relative* (p, ϵ)*-approximations* help to achieve a guaranteed relative error with a sample of small size [HPS11].

44.9 BETTER GUARANTEES

Bounds for the expected performance of randomized algorithms are usually available. Sometimes stronger results are desired. If the analysis of the algorithm cannot be extended to provide such bounds, then some techniques may help to achieve them:

Randomized space vs. deterministic space. Any randomized algorithm using expected space S and expected time T can be converted to an algorithm that uses deterministic space $2S$, and whose expected running time is at most $2T$. We simply need to maintain a count of the memory allocated by the algorithm. Whenever it exceeds $2S$, we stop the computation and restart it again with fresh choices for the random variables. The expected number of retrials is one.

Tail estimates. The knowledge that the expected running time of a given program is one second does not exclude the possibility that it sometimes takes one hour. Markov's inequality implies that the probability that this happens is at most 1/3600. While this seems innocuous, it implies that it is likely to occur if we repeat this particular computation, say, 10000 times.

For randomized incremental construction, better tail estimates can sometimes be proven [CHPR16, Lemma 3.4]. In particular, bounds are known for segment intersection in the plane [MSW93a] (see also [BCKO08, Section 6.4]) and for LP-type optimization [GW00]. Tail estimates are also available for the *space complexity* of randomized incremental construction [CMS93, MSW93b].

In all other cases, one can still apply a simple modification to the algorithm to yield a stronger bound. We run it for two seconds. If it does not finish the computation within two seconds, then we abandon the computation and restart with fresh choices for the random variables. Clearly, the probability that the algorithm does not terminate within one hour is at most 2^{-1800}. Alt et al. [AGM+96] work out this technique, which is a special case of success amplification [Hro05, Chapter 5], in detail.

44.10 PROBABILISTIC PROOF TECHNIQUES

Randomized algorithms are related to probabilistic proofs and constructions in combinatorics, which precede them historically. Conversely, the concepts developed to design and analyze randomized algorithms in computational geometry can be used as tools in proving purely combinatorial results. Many of these results are based on the following theorem:

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

THEOREM 44.10.1

Let $(X, \mathcal{T}, D, K)$ be a configuration space satisfying axioms (i), (ii), (iii), and (iii′) of Section 44.5. For $S \subseteq X$ and $0 \leq k \leq n$, let

$$\Pi^k(S) := \{\Delta \in \Pi(X) \mid |K(\Delta) \cap S| \leq k\}$$

denote the set of configurations with at most k conflicts in S.

Then $|\Pi^k(S)| = O(k^d)E[|\mathcal{T}(R)|]$, where R is a random sample of S of size n/k, and d is as in axiom (i).

Note that $\Pi^0(S) = \mathcal{T}(S)$. The theorem relates the number of configurations with at most k conflicts to those without conflict.

An immediate application is to prove a bound on the number of vertices of level at most k in an arrangement of lines in the plane (the level of a vertex is the number of lines lying above it; see Section 20.2). We define a configuration space $(X, \mathcal{T}, D, K)$ where X is the set of lines, $\mathcal{T}(S)$ is the set of vertices of the upper envelope of the lines, $D(\Delta)$ are the two lines forming the vertex Δ (so $d = 2$), and $K(\Delta)$ is the set of lines lying above Δ. Theorem 44.10.1 implies that the number of vertices of level up to k is bounded by $O(nk)$. The same argument works in any dimension.

Sharir and others have proved a number of combinatorial results using this technique [Sha94, AES99, ASS96, SS97]. They define a configuration space and need to bound $|\mathcal{T}(S)|$. They do this by proving a geometric relationship between the configurations with zero conflicts (the ones appearing in $\mathcal{T}(S)$) and the configurations with at most k conflicts. Applying Theorem 44.10.1 yields a recursion that bounds $|\mathcal{T}(S)|$ in terms of $|\mathcal{T}(R)|$. A refined approach that uses a sample of size $n - 1$ (instead of n/k) has been suggested by Tagansky [Tag96].

Sharir [Sha03] reviews this technique and gives a new proof for Theorem 44.10.1 based on the *Crossing lemma* (Chapter 28).

44.11 SOURCES AND RELATED MATERIAL

SURVEYS AND BOOKS

[BCKO08]: This textbook on computational geometry contains a gentle introduction to randomized incremental construction for several problems.

[HP11]: This monograph covers approximation algorithms in computational geometry and includes many randomized algorithms.

[Cla92, Mul00]: General surveys of randomized algorithms in computational geometry.

[Sei93]: An introduction to randomized incremental algorithms using backwards analysis.

[GS93]: Surveys computations with arrangements, including randomized algorithms.

[AS01] Surveys randomized techniques in geometric optimization problems.

[Mul93]: This monograph is an extensive treatment of randomized algorithms in computational geometry.

[Mat00]: An introduction to derandomization for geometric algorithms, with many references.

[MR95]: A book on randomized algorithms and their analysis in computer science, including derandomization techniques.

[AS16]: This monograph covers probabilistic proof techniques mostly in combinatorics, and includes some algorithmic aspects, geometric results, and derandomization.

[HP09]: A survey of ϵ-samples, approximations, and relative (p, ϵ)-approximations.

RELATED CHAPTERS

Because randomized algorithms have been used successfully in nearly all areas of computational geometry, they are mentioned throughout Parts C and D of this Handbook. Areas where randomization plays a particularly important role include:

Chapter 26: Convex hull computations
Chapter 28: Arrangements
Chapter 40: Range searching
Chapter 47: Epsilon-approximations and epsilon-nets
Chapter 48: Coresets and sketches
Chapter 49: Linear programming

REFERENCES

[AES99] P.K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29:912–953, 1999.

[AGM+96] H. Alt, L. Guibas, K. Mehlhorn, R. Karp, and A. Wigderson. A method for obtaining randomized algorithms with small tail probabilities. *Algorithmica*, 16:543–547, 1996.

[AGR95] N.M. Amato, M.T. Goodrich, and E.A. Ramos. Computing faces in segment and simplex arrangements. In *Proc. 27th ACM Sympos. Theory Comput.*, pages 672–682, 1995.

[AMS98] P.K. Agarwal, J. Matoušek, and O. Schwarzkopf. Computing many faces in arrangements of lines and segments. *SIAM J. Comput.*, 27:491–505, 1998.

[AS01] P.K. Agarwal and S. Sen. Randomized algorithms for geometric optimization problems. In S. Rajasekaran, P.M. Pardalos, J.H. Reif, and J. Rolim, editors, *Handbook of Randomized Computing*, pages 151–201, Kluwer Academic, Boston, 2001.

[AS12] H. Alt and L. Scharf. Shape matching by random sampling. *Theoret. Comput. Sci.*, 442:2–12, 2012.

[AS16] N. Alon and J.H. Spencer. *The Probabilistic Method*, 4th edition. John Wiley & Sons, New York, 2016.

[ASS96] P.K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete Comput. Geom.*, 15:1–13, 1996.

[ASS10] H. Alt, L. Scharf, and D. Schymura. Probabilistic matching of planar regions. *Comput. Geom.*, 43:99–114, 2010.

[AST94] P.K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17:292–318, 1994.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

[Bal95] I.J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Sympos. Comput. Geom.*, pages 211–219, ACM Press, 1995.

[BC98] H. Brönnimann and B. Chazelle. Optimal slope selection via cuttings. *Comput. Geom.*, 10:23–29, 1998.

[BCKO08] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*, 3rd edition. Springer, Berlin, 2008.

[BCM99] H. Brönnimann, B. Chazelle, and J. Matoušek. Product range spaces, sensitive sampling, and derandomization. *SIAM J. Comput.*, 28:1552–1575, 1999.

[BDS95] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. *Discrete Comput. Geom.*, 14:261–286, 1995.

[Bes01] S. Bespamyatnikh. An efficient algorithm for the three-dimensional diameter problem. *Discrete Comput. Geom.*, 25:235–255, 2001.

[BR94] M. Bellare and J. Rompel. Randomness-efficient oblivious sampling. In *Proc. 35th IEEE Sympos. Found. Comp. Sci.*, pages 276–287, 1994.

[BRS94] B. Berger, J. Rompel, and P.W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. *J. Comput. Syst. Sci.*, 49:454–477, 1994.

[CEGS93] B. Chazelle, H. Edelsbrunner, L.J. Guibas, and M. Sharir. Diameter, width, closest line pair and parametric searching. *Discrete Comput. Geom.*, 10:183–196, 1993.

[CEHP07] O. Cheong, A. Efrat, and S. Har-Peled. Finding a guard that sees most and a shop that sells most. *Discrete Comput. Geom.*, 37:545–563, 2007.

[Cha93] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.*, 10:377–409, 1993.

[Cha96] T.M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete Comput. Geom.*, 16:361–368, 1996.

[Cha99] T.M. Chan. Geometric applications of a randomized optimization technique. *Discrete Comput. Geom.*, 22:547–567, 1999.

[Cha04] T.M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proc. 15th ACM-SIAM Sympos. Discrete Algorithms*, pages 430–436, 2004.

[Cha12] T.M. Chan. Optimal partition trees. *Discrete Comput. Geom.*, 47:661–690, 2012.

[Cha16] T.M. Chan. Improved deterministic algorithms for linear programming in low dimensions. In *Proc. 27th ACM-SIAM Sympos. Discrete Algorithms*, pages 1213–1219, 2016.

[CHPR16] H.-C. Chang, S. Har-Peled, and B. Raichel. From proximity to utility: A Voronoi partition of Pareto optima. *Discrete Comput. Geom.*, 56:631–656, 2016.

[Cla92] K.L. Clarkson. Randomized geometric algorithms. In D.-Z. Du and F.K. Hwang, editors, *Computing in Euclidean Geometry*, vol. 1 of *Lecture Notes Series on Computing*, pages 117–162, World Scientific, Singapore, 1992.

[CM95] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Comput. Geom.*, 5:27–32, 1995.

[CM96] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21:579–597, 1996.

[CMS93] K.L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom.*, 3:185–212, 1993.

[CT16] T.M. Chan and K. Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete Comput. Geom.*, 2016.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

[DMT92] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Comput. Geom.*, 2:55–80, 1992.

[GS93] L.J. Guibas and M. Sharir. Combinatorics and algorithms of arrangements. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, vol. 10 of *Algorithms and Combin.*, pages 9–36. Springer-Verlag, Berlin, 1993.

[GW00] B. Gärtner and E. Welzl. Random sampling in geometric optimization: New insights and applications. In *Proc. 16th Sympos. Comput. Geom.*, pages 91–99, ACM Press, 2000.

[HP00] S. Har-Peled. Constructing planar cuttings in theory and practice. *SIAM J. Comput.*, 29:2016–2039, 2000.

[HP09] S. Har-Peled. Carnival of samplings: Nets, approximations, relative and sensitive. Preprint, `arXiv:0908.3716`, 2009.

[HP11] S. Har-Peled. *Geometric Approximation Algorithms*. AMS, Providence, 2011.

[HPS11] S. Har-Peled and M. Sharir. Relative (p, ϵ)-approximations in geometry. *Discrete Comput. Geom.*, 45:462–496, 2011.

[Hro05] J. Hromkovič. *Design and Analysis of Randomized Algorithms*. Springer, Berlin, 2005.

[Jof74] A. Joffe. On a set of almost deterministic k-independent random variables. *Ann. Probab.*, 2:161–162, 1974.

[KK97] D.R. Karger and D. Koller. (De)randomized construction of small sample spaces in NC. *J. Comput. Syst. Sci.*, 55:402–413, 1997.

[KM94] D. Koller and N. Megiddo. Constructing small sample spaces satisfying given constraints. *SIAM J. Discrete Math.*, 7:260–274, 1994.

[KM97] H. Karloff and Y. Mansour. On construction of k–wise independent random variables. *Combinatorica*, 17:91–107, 1997.

[Mat91] J. Matoušek. Randomized optimal algorithm for slope selection. *Inform. Process. Lett.*, 39:183–187, 1991.

[Mat00] J. Matoušek. Derandomization in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 559–595, North-Holland, Amsterdam, 2000.

[MNN94] R. Motwani, J. Naor, and M. Naor. The probabilistic method yields deterministic parallel algorithms. *J. Comput. Syst. Sci.*, 49:478–516, 1994.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MRS01] S. Mahajan, E.A. Ramos, and K.V. Subrahmanyam. Solving some discrepancy problems in NC. *Algorithmica*, 29:371–395, 2001.

[MSW93a] K. Mehlhorn, M. Sharir, and E. Welzl. Tail estimates for the efficiency of randomized incremental algorithms for line segment intersection. *Comput. Geom.*, 3:235–246, 1993.

[MSW93b] K. Mehlhorn, M. Sharir, and E. Welzl. Tail estimates for the space complexity of randomized incremental algorithms. *Comput. Geom.*, 4:185–246, 1993.

[MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[Mul91a] K. Mulmuley. Randomized multidimensional search trees: Dynamic sampling. In *7th Sympos. Comput. Geom.*, pages 121–131, ACM Press, 1991.

[Mul91b] K. Mulmuley. Randomized multidimensional search trees: Further results in dynamic sampling. In *32nd IEEE Sympos. Found. Comp. Sci.*, pages 216–227, 1991.

[Mul91c] K. Mulmuley. Randomized multidimensional search trees: Lazy and dynamic shuffling. In *32nd IEEE Sympos. Found. Comp. Sci.*, pages 180–196, 1991.

[Mul93] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, 1993.

[Mul96] K. Mulmuley. Randomized geometric algorithms and pseudorandom generators. *Algorithmica*, 16:450–463, 1996.

[Mul00] K. Mulmuley. Randomized algorithms in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 703–724, North-Holland, Amsterdam, 2000.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12:449–461, 1992.

[NN93] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22:838–856, 1993.

[PT13] J. Pach and G. Tardos. Tight lower bounds for the size of epsilon-nets. *J. Amer. Math. Soc.*, 26:645–658, 2013.

[Rab76] M.O. Rabin. Probabilistic algorithms. In J. Traub, editor, *Algorithms and Complexity*, pages 21–39. Academic Press, New York, 1976.

[Rag88] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *J. Comput. Syst. Sci.*, 37:130–143, 1988.

[Ram01] E.A. Ramos. An optimal deterministic algorithm for computing the diameter of a three-dimensional point set. *Discrete Comput. Geom.*, 26:233–244, 2001.

[Sch91] O. Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proc. 32nd IEEE Sympos. Found. Comp. Sci.*, pages 197–206, 1991.

[Sei91] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom.*, 1:51–64, 1991.

[Sei93] R. Seidel. Backwards analysis of randomized geometric algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, vol. 10 of *Algorithms and Combin.*, pages 37–68. Springer-Verlag, Berlin, 1993.

[Sha94] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete Comput. Geom.*, 12:327–345, 1994.

[Sha03] M. Sharir. The Clarkson-Shor technique revisited and extended. *Combin. Probab. Comput.*, 12:191–201, 2003.

[Spe87] J. Spencer. *Ten Lectures on the Probabilistic Method*. CBMS-NSF. SIAM, 1987.

[SS97] O. Schwarzkopf and M. Sharir. Vertical decomposition of a single cell in a three-dimensional arrangement of surfaces and its applications. *Discrete Comput. Geom.*, 18:269–288, 1997.

[SSS95] J.P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8:223–250, 1995.

[Tag96] B. Tagansky. A new technique for analyzing substructures in arrangements of piecewise linear surfaces. *Discrete Comput. Geom.*, 16:455–479, 1996.

Preliminary version (August 10, 2017). To appear in the Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Rourke, and C. D. Tóth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.