Low-Entropy Computational Geometry

Wolfgang Johann Heinrich Mulzer

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

Advisor: Bernard Chazelle

June 2010

Abstract

The worst-case model for algorithm design does not always reflect the real world: inputs may have additional structure to be exploited, and sometimes data can be imprecise or become available only gradually. To better understand these situations, we examine several scenarios where additional information can affect the design and analysis of geometric algorithms.

First, we consider hereditary convex hulls: given a three-dimensional convex polytope and a two-coloring of its vertices, we can find the individual monochromatic polytopes in linear expected time. This can be generalized in many ways, eg, to more than two colors, and to the offline-problem where we wish to preprocess a polytope so that any large enough subpolytope can be found quickly. Our techniques can also be used to give a simple analysis of the self-improving algorithm for planar Delaunay triangulations by Clarkson and Seshadhri [58].

Next, we assume that the point coordinates have a bounded number of bits, and that we can do standard bit manipulations in constant time. Then Delaunay triangulations can be found in expected time $O(n\sqrt{\log\log n})$. Our result is based on a new connection between quadtrees and Delaunay triangulations, which also lets us generalize a recent result by Löffler and Snoeyink about Delaunay triangulations for imprecise points [110].

Finally, we consider randomized incremental constructions when the input permutation is generated by a bounded-degree Markov chain, and show that the resulting running time is almost optimal for chains with a constant eigenvalue gap.

Acknowledgments

Without an advisor, there cannot be a thesis, and therefore I would like to kick off this acknowledgments section with a big shout-out to Bernard Chazelle, without whose perseverance, curiosity, guidance and never-wavering support this thesis would not have been possible.

I would also like to thank the other members of my thesis committee for providing valuable feedback and helping improve the quality of this thesis: Sanjeev Arora, Boaz Barack, Moses Charikar, David Dobkin, and Bob Tarjan.

I am also grateful for the wonderful time I spent with the theory group at Freie Universität Berlin, and would especially like to thank my advisors there, Christian Knauer and Günter Rote.

Science never happens in a vacuum, and I would like to thank my co-authors, with whom it has been (and still is) a pleasure to work: Nir Ailon, Tetsuo Asano, Kevin Buchin, Bernard Chazelle, Kenneth L. Clarkson, Esther Ezra, Christian Knauer, Ding Liu, Maarten Löffler, Pat Morin, Günter Rote, C. Seshadhri, and Yajun Wang.

I wish to thank David Eppstein, Jeff Erickson, and Mikkel Thorup for their stimulating and insightful advice concerning the problems presented in Chapter 5, and Alistair Sinclair for helpful discussions related to the material in Chapter 7. I would also like to thank the anonymous referees of the various preliminary versions of the results in this thesis, for their helpful comments and pointers to the literature.

I would like to thank the Computer Science department at Princeton University, and especially Melissa Lawson, whose handling of all the administrative work made my PhD studies flow smoothly and seamlessly.

I gratefully acknowledge a Wallace Memorial Fellowship in Engineering, as well as NSF grants CCR-0306283, CCF-0634958, and CCF-0832797 for providing financial support for my studies at Princeton University.

Finally, I would like to thank my family for their love and support, my parents Johann and Ingeborg Mulzer, my brother Michael and my sister Johanna, and my aunt Gudrun Thormann for providing me with a postcard collection that has become the envy of the department.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

In the past thirty years, computational geometry has come a long way in understanding the fundamental problems of geometric computing, and many simple, efficient, and optimal algorithms have been discovered [23, 31, 122, 128]. With a solid foundation and a well-developed toolbox in place, we can now venture beyond the basics and explore the additional structure behind these problems: What are the assumptions behind the classic results? Are these assumptions justified? What if they do not hold? What makes a problem hard? How can additional structure help? What is the role of randomness? What other notions of optimality besides worst-case performance can be used fruitfully? A better understanding of these issues can lead to deeper insight into the underlying problems and help us design improved algorithms for cases where old lower bounds do not apply.

Traditional algorithms assume worst-case inputs, that are known exactly, with infinite precision; they also often require perfect randomness. Take the example of *sorting*, one of the best-studied problems in theoretical computer science [61, 102, 136]. All undergraduates learn about the basic sorting algorithms, and they are taught that sorting needs $\Omega(n \log n)$ steps in the binary decision tree model. They also learn about the popular quicksort algorithm, and that it yields an optimal expected running time when the input array is permuted randomly. However, the $\Omega(n \log n)$ lower bound is deceptive, and there are many ways in which additional structure in the inputs can help to get around it:

- If the elements come from a totally ordered universe U, we can build a data structure to sort any subset $S \subseteq U$ in time $O(|S| \log\log |U|)$ [71, 72, 114]. We call this an *hereditary* result, because S inherits enough structure from U that S can be sorted faster.
- Suppose we want to sort n w-bit numbers, for some $w \geq \log n$, and that our computational model supports standard bit operations (`and`, `or`, `xor`, etc) in constant time. Then it is possible to sort in expected time $O(n\sqrt{\log\log n})$,

irrespective of w [88]. This line of research was advocated by Fredman and Willard [80, 81], who called it the study of *transdichotomous algorithms.*

- The inputs could be *restricted.* For example, suppose we are given a set $\mathcal{R}$ of n intervals such that every number is contained in at most k of them. Then $\mathcal{R}$ can be preprocessed into a linear space data structure such that given a set S with exactly one point from each interval, we can sort S in time $O(n \log k)$.

Similarly, the assumption that quicksort has access to perfect randomness often does not hold. However, it can be shown that an $O(1)$-wise independent random permutation suffices to achieve optimal expected performance [123].

In computational geometry, there are many problems for which a reduction from sorting yields an $\Omega(n \log n)$ lower bound. This makes it natural to ask how the three kinds of structure—hereditary, transdichotomous, and restricted inputs—affect the difficulty of these problems. Furthermore, many geometric algorithms are randomized, and we would like to know in what way imperfect randomness can influence the running time. We call this whole body of questions the study of *low-entropy computational geometry.* Here, low entropy can have two different meanings:

1. **Low entropy in the inputs.** Through the additional structure in the inputs, we derive less information from any specific problem instance, and the information theoretic lower bounds from the decision tree model do not apply any longer.

2. **Low entropy in the algorithm.** When using an imperfect random source, the algorithm has less entropy at its disposal, with potentially adverse effects on the expected running time.

These two kinds of low entropy are wide-spread. Of course, complexity theory [14] has spent a lot of effort studying the limitations (or lack thereof) of algorithms with an imperfect source of randomness, and our concern here will be to understand specific problems with specific kinds of randomness, rather than to develop a general theory. Furthermore, there are many situations in which an algorithm is confronted with inputs that have low entropy. For example, *hidden Markov models* [131], which stipulate a strong local coherence between individual inputs, are often used to model real life data such as speech, web-surfing, or robot motion; and also when processing videos or image data, we often need to deal with inputs which are locally very similar.

The dual nature of low entropy will be the theme that connects the different results to be presented. In the next few sections, we will describe these results in more detail and give an outline of what the reader can expect from the coming chapters.

1.1 The utility of additional structure

As mentioned above, we will consider three different kinds of additional structure in the inputs that can lead to faster algorithms: hereditary structure, transdichotomous models, and restricted inputs.

Hereditary structure and convex hulls in $\mathbb{R}^3$. Let $\mathcal{P}$ be a convex polytope in $\mathbb{R}^3$ and S a subset of its vertices. How much information does $\mathcal{P}$ give away about the convex hull of S, that is, how much structure does the convex hull of S *inherit* from $\mathcal{P}$? In Chapter 3, we will see that the hull of S can be found in $O(|\mathcal{P}|)$ time. Hence, if S is large enough, $\mathcal{P}$ tells us everything we need to know. This extends previous work about Delaunay triangulations (DTs) by Chazelle et al. [47], and algorithms for similar settings were also obtained by van Kreveld et al. [106] and Chan [38].

There are many interesting ways to extend our result, for example, we give algorithms for splitting a 3D polytope into more than two parts and for completing a partial Delaunay triangulation in almost optimal time, providing a Delaunay analogue to a result by Bar-Yehuda and Chazelle [19].

Transdichotomous Delaunay triangulations. As we already said, *transdichotomous algorithms* offer a way to go beyond the classic decision tree model. Ever since Fredman and Willard [80, 81] advocated this model in 1990, there has been a series of increasingly faster integer sorting algorithms, culminating in Han and Thorup's result from 2002 which gives an algorithm to sort n integers in expected time $O(n\sqrt{\log\log n})$ [88]. In computational geometry general transdichotomous results have long remained elusive, since it was not clear how to generalize the one-dimensional sorting algorithms to higher dimensions. Therefore, authors would focus on problems of an *orthogonal* flavor, where lines and line segments can have only a bounded number of different slopes and where one-dimensional techniques can be applied. This situation changed when Chan and Pătraşcu [40,41] first showed how to overcome these limitations. In particular, they achieved expected running time $n2^{O(\sqrt{\log\log n})}$ for planar DTs. In Chapter 5, we describe how to improve this bound to $O(n\sqrt{\log\log n})$, using Han and Thorup's result [88]. We show that given a quadtree for a planar point set, its DT can be found in $O(n)$ time, so ultimately, Delaunay computation reduces to sorting. For this, we need well-known tools, like the Morton-curve [118] and well-separated pair decompositions [35], as well as a new variant of geometric sampling with dependencies, inspired by work of Amenta et al. [9].

Restricted inputs. Suppose we want to find a planar DT, but the input is *restricted*: we know a set of planar regions $\mathcal{R}$ such that each point comes from exactly one region of $\mathcal{R}$. Can we preprocess $\mathcal{R}$ for faster Delaunay computation? This question has received some attention in the study of imprecise input models [89,106,110], and Löffler and Snoeyink [110] showed that the DT can be found in linear time if $\mathcal{R}$ consists of disjoint unit disks.

In Chapter 6, we will rederive this result with a much simpler (but randomized) algorithm, and show how to extend it to more general classes of input regions $\mathcal{R}$, with optimal parameters. For example, if $\mathcal{R}$ consists of (not necessarily unit) disks, such that each point in the plane is covered by at most k disks, the DT can be found in time $O(n \log k)$ after preprocessing, which is optimal. The proof is again based on the connection between quadtrees and Delaunay triangulations, as well as *guarding sets* [24] and a carefully balanced data structure to obtain linear space.

1.2 Imperfect randomness

Randomized incremental construction (RIC) is a classic tool in computational geometry [31,59,122]: to compute a certain geometric structure we randomly permute its constituent parts and insert them one by one. The resulting algorithm is optimal, and Mulmuley [123] showed that this even holds for $O(1)$-wise independent random permutations.

Thus, RICs need high *local* entropy: every k-subset should be completely random. However, there is a very common and natural type of random sources without this property: Markov chains. These model time-dependent natural processes, such as speech, web-surfing, robot movement, etc. A permutation obtained by a random walk on a labeled graph only generates a tiny amount of randomness per step, and therefore the basic needs of the RIC paradigm seem to be violated. However, Chapter 7 shows that even in this case RICs are almost optimal, up to poly-logarithmic factors. We employ spectral techniques to give new bounds on the first-passage time of Markov chains, and also a new Clarkson-Shor type bound that supports dependent sampling.

1.3 Previous publications

The results covered in this thesis, or preliminary versions thereof, have appeared previously in the following publications (listed in chronological order):

- B. Chazelle and W. Mulzer. *Markov Incremental Constructions.* In Discrete and Computational Geometry (DCG) 42(3), pp. 399–420, 2009. Preliminary version in SoCG 2008.

- B. Chazelle and W. Mulzer. *Computing Hereditary Convex Structures.* In Proceedings of the 25th Annual ACM Symposium on Computational Geometry (SoCG), pp. 61–70, 2009.

- K. Buchin, M. Löffler, P. Morin, and W. Mulzer. *Delaunay Triangulation of Imprecise Points Simplified and Extended.* Proceedings of the 11th Algorithms and Data Structures Symposium (WADS), pp. 131–143, 2009.

- K. Buchin and W. Mulzer. *Delaunay Triangulations in $O(\texttt{sort}(n))$ Time and More.* Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 139–148, 2009.

- N. Ailon, B. Chazelle, K. L. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. *Self-Improving Algorithms.* `arXiv:0907.0884`, 2009.

Chapter 2

Preliminaries and Background

Before going in medias res, let us take a moment to review some basic notions from computational geometry [23, 31, 122, 128] such as Delaunay triangulations and convex hulls, and to recall some facts from geometric sampling theory [59, 122] and about different computational models [14].

2.1 Definitions and notation

Throughout, we will use the *Vinogradov notation*, $f \ll g$ for $f(n) = O(g(n))$ and $f \gg g$ for $f(n) = \Omega(g(n))$. In the following paragraphs, we will review some basic concepts from computational geometry.

Delaunay triangulations, Voronoi diagrams, and convex hulls Let $P \subseteq \mathbb{R}^2$ be a finite point set. A *geometric* graph with vertex set P is a planar graph $G = (V, E)$ that is embedded in the plane such that the vertices in V correspond bijectively to the points in P, and such that the edges in E are represented by line segments. A *triangulation* of P is a maximal geometric graph with vertex set P. All internal faces of a triangulation are triangles. A *Delaunay triangulation* of P, denoted by $\mathrm{DT}(P)$, is a triangulation of P which has the *empty circle property*: for any triangle f in $\mathrm{DT}(P)$, the (open) circumcircle of f does not contain any points in P. If P is in *general position*, ie, if P contains no three points on a common line and no four points on a common circle, then $\mathrm{DT}(P)$ is uniquely determined.

A related structure is the *Voronoi diagram* of P, $\mathcal{V}(P)$. This is a subdivision of the plane into cells $\mathcal{C}_p, p \in P$, where the cell $\mathcal{C}_p$ contains all points $q \in \mathbb{R}^2$ such that there exists no point $r \in P \setminus p$ with $\|q - r\|_2 < \|q - p\|_2$. The points contained in at least two cells $\mathcal{C}_p$ constitute the edges of $\mathcal{V}(P)$, and the points in at least three cells form its vertices. If P is in general position, then $\mathcal{V}(P)$ is the graph theoretic dual of $\mathrm{DT}(P)$, ie, the vertices of $\mathcal{V}(P)$ correspond to the facets in $\mathrm{DT}(P)$, and the

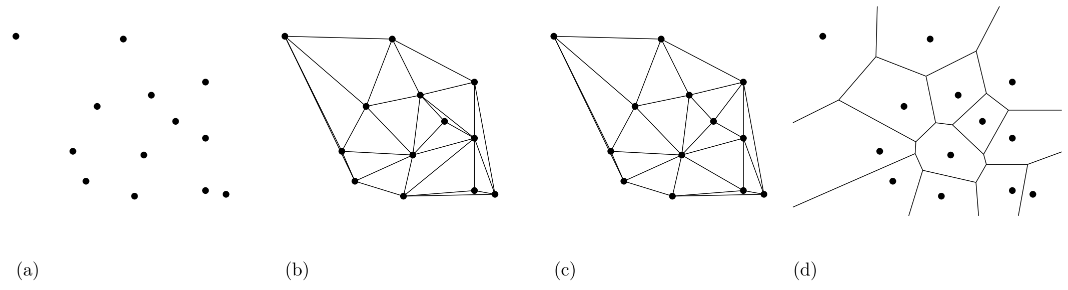


Figure 2.1: (a) A planar point set P; (b) a triangulation of P; (c) the Delaunay triangulation of P; and (d) (part of) the Voronoi diagram for P.

facets of $\mathcal{V}(P)$ correspond to the vertices of $\mathrm{DT}(P)$. Please see Figure 2.1 for an example of a Delaunay triangulation and a Voronoi diagram.

Finally, for a finite point set $P \subseteq \mathbb{R}^3$, the *convex hull* of P, $\operatorname{conv} P$, is the minimum convex set containing P.[1] The boundary of $\operatorname{conv} P$ consists of *vertices* (a subset of P), *edges*, and *facets*. We denote the edges of $\operatorname{conv} P$ by $E[P]$ and the facets by $F[P]$. For a point $p \in P$, let $\deg_P p$ be the number of edges in $E[P]$ incident to p, the *degree* of p (with respect to P). Throughout, we will assume that convex hulls are given in a standard planar graph representation, eg, a DCEL [23, Chapter 2.2]. Our point sets will usually be in *general convex position* (gcp), ie, every three points in P are linearly independent and $p \notin \operatorname{conv}(P \setminus p)$ for every $p \in P$. In particular, $\operatorname{conv} P$ is simplicial[2], and all the points in P are vertices of $\operatorname{conv} P$.

There is a well-known connection between planar Delaunay triangulations and convex hulls in $\mathbb{R}^3$. Namely, let $P \in \mathbb{R}^2$ be a planar point set, and let $\widehat{P}$ be obtained by projecting P onto the three-dimensional unit paraboloid, ie, by mapping $p = (p_x, p_y) \in P$ to $\hat{p} = (p_x, p_y, p_x^2 + p_y^2)$. Then $\mathrm{DT}(P)$ can be found by computing the part of $\operatorname{conv} \widehat{P}$ visible from the negative z-axis and by projecting the result back onto the plane $z = 0$. See Figure 2.2 for a two-dimensional illustration.

Halfspaces and conflicts We will need some results from classic geometric random sampling theory [59, 122]; see Section 2.2. For this, we quickly review the notion of *conflict sets*. Given a point set $P \subseteq \mathbb{R}^3$, an edge $e \in E[P]$, and a point $p \notin \operatorname{conv} P$, we say that p *can see* e in $\operatorname{conv} P$ or that e *is visible* from p, if the triangle spanned by e and p intersects $\operatorname{conv} P$ only in e. All the planes we consider are *oriented*, that is, one of the two halfspaces defined by a plane h is designated the *left halfspace* of h, h^+, and the other one is designated the *right halfspace* of h, h^-. We use the convention that every supporting plane of $\operatorname{conv} P$ is oriented such that

[1]A set $A \subseteq \mathbb{R}^3$ is *convex*, if for any $\lambda \in [0, 1]$, we have that $p, q \in A$ implies $\lambda p + (1 - \lambda)q \in A$.
[2]That is, all facets of $\operatorname{conv} P$ are triangles.

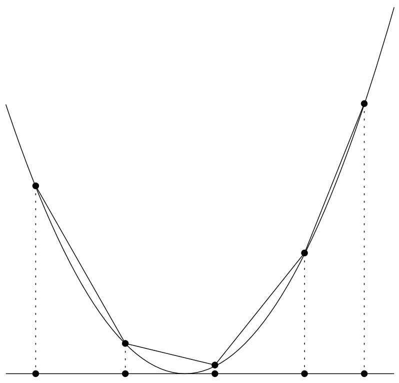


Figure 2.2: We can find DT P by projecting P onto the unit paraboloid in three dimensions, computing the lower convex hull, and projecting the result back down.

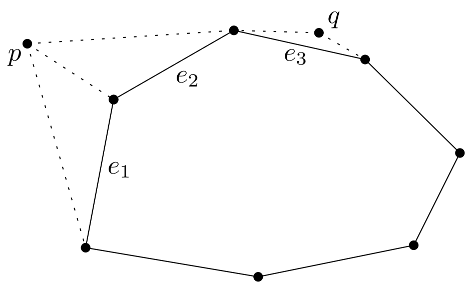


Figure 2.3: The point p is in conflict with e_1, and e_2, but not with e_3. The point q only conflicts with e_3.

P lies in the right halfspace h^-. Let $Q \subseteq P$, $f \in F[Q]$, and h_f be the supporting plane for f. A point $p \in P$ is *in conflict* with f if p lies in h_f^+, see Figure 2.3. Let $B_f \subseteq P$ denote the points in conflict with f, and b_f the size of B_f. Conversely, for a point $p \in P$, we let $D_p \subseteq F[Q]$ denote the set of facets in conflict with p, and let d_p be its size. The sets B_f and D_p are the *conflict sets* of f and p, and b_f and d_p are the *conflict sizes*. By double counting,

$$\sum_{f \in F[Q]} b_f = \sum_{p \in P} d_p. \tag{2.1}$$

Duality We will also need the notion of *geometric duality*. Given a point

$$p = (p_x, p_y, p_z) \in \mathbb{R}^3 \setminus \vec{0},$$

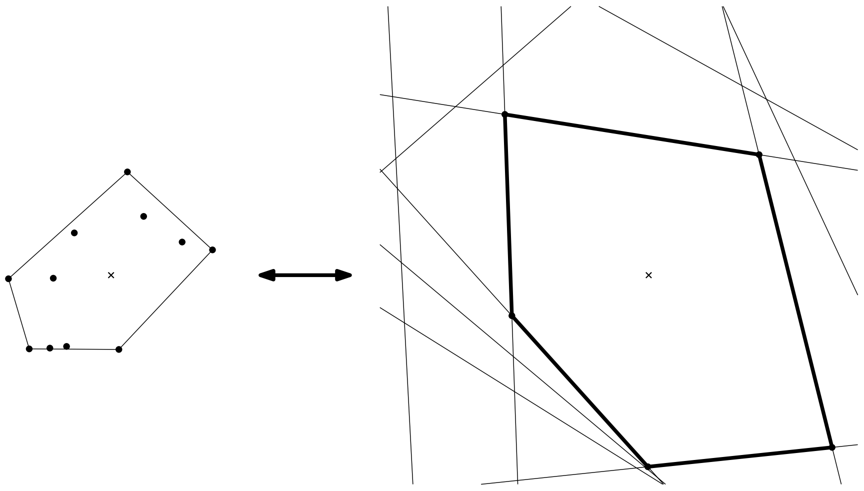


Figure 2.4: The problems of computing the convex hull of a point set and computing the intersection of a set of halfspaces are dual to each other.

the dual plane to p, p^*, is the plane defined by $p^* : 1 = p_x x + p_y y + p_z z$. And similarly, if $h : 1 = ax + by + cz$ is a plane in $\mathbb{R}^3$ that does not pass through the origin, the dual point to h, denoted by h^*, is (a, b, c). This duality establishes an equivalence between convex hulls and halfspace intersection. Namely, let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^3$ such that $\operatorname{conv} P$ contains the origin in its interior, and let $H = \{h_1, \dots, h_n\}$ be a set of halfspaces such that h_i is bounded by p_i^* and contains the origin. Then the facets, edges, and vertices of $\operatorname{conv} P$ are in one-to-one correspondence with the vertices, edges, and facets of $\bigcap_{i=1}^n h_i$, and this correspondence can be found in linear time. Therefore, convex hull computation and halfspace intersection are equivalent problems. See Figure 2.4 for a two-dimensional example.

2.2 Geometric sampling: the toolbox by Clarkson and Shor

We review a few tools from geometric random sampling theory [59, 122]. Our presentation follows Ramos [130]. Let $P \subseteq \mathbb{R}^3$ with $|P| = n$ and $K \subseteq P$ with $|K| = k$. Given a triple $\mathbf{u} = (p_1, p_2, p_3) \in P^3$, let $h_\mathbf{u}$ be the plane spanned by $\mathbf{u}$, oriented such that the set of vectors $\{u_2 - u_1, u_3 - u_1, p - u_1\}$ has positive determinant for $p \in h_\mathbf{u}^+$. A point $p \in P$ *conflicts* with $\mathbf{u}$ if p lies in $h_\mathbf{u}^+$. Let $B_\mathbf{u}$ denote the set of all points in P that conflict with $\mathbf{u}$, and $b_\mathbf{u} = |B_\mathbf{u}|$.

Lemma 2.2.1. *Fix $p \in (0, 1]$ and $t \geq 1$. Let $S \subseteq P \setminus K$ be a random subset of size $p(n-k)$ and let $S' \subseteq P \setminus K$ be a random subset of size $p'(n-k)$ for $p' = p/t$.*

Suppose that $p'(n-k) \geq 4$. Fix $\boldsymbol{u} = (p_1, p_2, p_2) \in P^3$, and let $f_{\boldsymbol{u}}$ be the facet defined by $\boldsymbol{u}$. Then

$$\Pr[f_{\boldsymbol{u}} \in F[S \cup K]] \ll t^3 \exp\left(-\frac{(t-1)pb_{\boldsymbol{u}}}{t}\right) \Pr[f_{\boldsymbol{u}} \in F[S' \cup K]]. \tag{2.2}$$

Proof. Let $\sigma = \Pr[f_{\mathbf{u}} \in F[S \cup K]]$ and $\sigma' = \Pr[f_{\mathbf{u}} \in F[S' \cup K]]$. Note that $f_{\mathbf{u}}$ appears in $F[S \cup K]$ precisely if $\mathbf{u} \subseteq S \cup K$ and $B_{\mathbf{u}} \cap (S \cup K) = \emptyset$. If $K \cap B_{\mathbf{u}} \neq \emptyset$, then $\sigma = \sigma' = 0$, and the lemma holds. Thus, we may assume that K and $B_{\mathbf{u}}$ are disjoint. Let $m = n - k$ and let $d_{\mathbf{u}}$ denote $3 - |K \cap \mathbf{u}|$, the number of points in $\mathbf{u}$ not in K. Since there are $\binom{m-b_{\mathbf{u}}-d_{\mathbf{u}}}{pm-d_{\mathbf{u}}}$ ways of choosing a pm-subset from $P \setminus K$ that avoids all elements in $B_{\mathbf{u}}$ and contains all the relevant points of $\mathbf{u}$, we have

$$\begin{aligned}\sigma = \binom{m-b_{\mathbf{u}}-d_{\mathbf{u}}}{pm-d_{\mathbf{u}}} \bigg/ \binom{m}{pm} &= \frac{\prod_{j=0}^{pm-d_{\mathbf{u}}-1}(m-b_{\mathbf{u}}-d_{\mathbf{u}}-j)}{\prod_{j=0}^{pm-d_{\mathbf{u}}-1}(pm-d_{\mathbf{u}}-j)} \bigg/ \frac{\prod_{j=0}^{pm-1}(m-j)}{\prod_{j=0}^{pm-1}(pm-j)} \\ &= \prod_{j=0}^{d_{\mathbf{u}}-1} \frac{pm-j}{m-j} \cdot \prod_{j=0}^{pm-d_{\mathbf{u}}-1} \frac{m-b_{\mathbf{u}}-d_{\mathbf{u}}-j}{m-d_{\mathbf{u}}-j} \\ &\leq p^{d_{\mathbf{u}}} \prod_{j=0}^{pm-d_{\mathbf{u}}-1} \left(1 - \frac{b_{\mathbf{u}}}{m-d_{\mathbf{u}}-j}\right).\end{aligned}$$

Similarly, we get

$$\sigma' = \prod_{j=0}^{d_{\mathbf{u}}-1} \frac{p'm-j}{m-j} \cdot \prod_{j=0}^{p'm-d_{\mathbf{u}}-1} \left(1 - \frac{b_{\mathbf{u}}}{m-d_{\mathbf{u}}-j}\right),$$

and since $p'm \geq 4$ and $j \leq 2$ (in the first product), it follows that

$$\sigma' \geq \left(\frac{p'}{2}\right)^{d_{\mathbf{u}}} \prod_{j=0}^{p'm-d_{\mathbf{u}}-1} \left(1 - \frac{b_{\mathbf{u}}}{m-d_{\mathbf{u}}-j}\right).$$

Therefore, since $p' = p/t$,

$$\begin{aligned}\frac{\sigma}{\sigma'} &\leq 8\left(\frac{p}{p'}\right)^{d_{\mathbf{u}}} \prod_{j=p'm-d_{\mathbf{u}}}^{pm-d_{\mathbf{u}}-1} \left(1 - \frac{b_{\mathbf{u}}}{m-d_{\mathbf{u}}-j}\right) \\ &\leq 8t^3 \left(1 - \frac{b_{\mathbf{u}}}{m}\right)^{(t-1)pm/t} \leq 8t^3 \exp\left(-\frac{(t-1)pb_{\mathbf{u}}}{t}\right),\end{aligned}$$

as desired. □

The lemma implies a Chernoff-type bound for the conflict size of a random sample.

Lemma 2.2.2. *Fix $p \in (0,1]$ and let $S \subseteq P$ be a random subset of size pn. Fix $t \geq 1$ such that $t \leq pn/4$ and let $F_{\geq t} = \{f \in F[S] \mid b_f \geq t/p\}$. Then*

$$\mathbf{E}\left[|F_{\geq t}|\right] \ll t^2 e^{-t} pn.$$

Proof. Let $S' \subseteq P$ be a random subset of size pn/t. Since $pn/t \geq 4$, we have

$$\begin{aligned}\mathbf{E}\left[|F_{\geq t}|\right] &= \sum_{\substack{\mathbf{u} \in P^3 \\ b_{\mathbf{u}} \geq t/p}} \Pr\left[f_{\mathbf{u}} \in F[S]\right] \\ &\ll \sum_{\substack{\mathbf{u} \in P^3 \\ b_{\mathbf{u}} \geq t/p}} t^3 \exp\left(-\frac{(t-1)pb_{\mathbf{u}}}{t}\right) \Pr\left[f_{\mathbf{u}} \in F[S']\right] \qquad \text{(by (2.2))} \\ &\ll t^3 e^{-t} \mathbf{E}\left[|F[S']|\right] \ll t^2 e^{-t} pn,\end{aligned}$$

because $\mathbf{E}\left[|F[S']|\right] \ll pn/t$. □

Next, we want to bound the average conflict size. For this, we first determine the average for a particular function, from which we then deduce bounds for a large class of well-behaved functions.

Lemma 2.2.3. *Fix $p \in (0,1]$ and let $S \subseteq P \setminus K$ be a random subset of size $p(n-k)$. Then*

$$\mathbf{E}\left[\sum_{f \in F[S \cup K]} \exp\left(\frac{pb_f}{2}\right)\right] \ll p(n-k) + k. \tag{2.3}$$

Proof. We may assume that $p(n-k)/2 \geq 4$, because otherwise $pb_f = O(1)$ for every $f \in F[S \cup K]$ (as all these f have $b_f \leq n-k$ and $\operatorname{conv}(S \cup K)$ has $O(p(n-k)+k)$ facets) and the lemma would hold trivially. Let $S' \subseteq P \setminus K$ be a random subset of size $p(n-k)/2$. We have

$$\begin{aligned}\mathbf{E}\left[\sum_{f \in F[S \cup K]} \exp\left(\frac{pb_f}{2}\right)\right] &= \sum_{\mathbf{u} \in P^3} \Pr[f_{\mathbf{u}} \in F[S \cup K]] \exp\left(\frac{pb_{\mathbf{u}}}{2}\right) \\ &\ll \sum_{\mathbf{u} \in P^3} \Pr[f_{\mathbf{u}} \in F[S' \cup K]] \qquad \text{(by (2.2))} \\ &= \mathbf{E}\left[|F[S' \cup K]|\right] \ll p(n-k) + k.\end{aligned}$$

□

Using this bound, we can show that the sum of every well-behaved function over the conflict sizes of a random sample gives the value one would expect. This remains true if a few points from P are always included in the sample.

Lemma 2.2.4. *Fix $p \in (0,1]$ and let $S \subseteq P \setminus K$ be a random subset of size $p(n-k)$. Let g be a function such that $g(tn) \ll e^t g(n)$ for all $t \geq 0$. Then*

$$\mathbf{E}\left[\sum_{f \in F[S \cup K]} g(b_f)\right] \ll (p(n-k)+k) \cdot g(1/p).$$

In particular, choosing $k = 0$ and $g : n \mapsto n^\gamma$ for $\gamma \geq 0$, we have

$$\mathbf{E}\left[\sum_{f \in F[S]} b_f^\gamma\right] \ll np^{1-\gamma}, \tag{2.4}$$

and choosing $g : n \mapsto n \log n$, we get

$$\mathbf{E}\left[\sum_{f \in F[S \cup K]} b_f \log b_f\right] \ll \left(n - k + \frac{k}{p}\right) \log \frac{1}{p}. \tag{2.5}$$

Proof. We have

$$\begin{aligned}
\mathbf{E}\left[\sum_{f \in F[S \cup K]} g(b_f)\right] &= \mathbf{E}\left[\sum_{f \in F[S \cup K]} g\left(\frac{pb_f}{2} \cdot \frac{2}{p}\right)\right] \\
&\ll \exp(2) g\left(\frac{1}{p}\right) \cdot \mathbf{E}\left[\sum_{f \in F[S \cup K]} \exp\left(\frac{pb_f}{2}\right)\right] \\
&\ll (p(n-k)+k) \cdot g(1/p). && \text{(by (2.3))}
\end{aligned}$$

□

The following lemma is a standard application of the geometric divide-and-conquer technique [46, 55, 59] and asserts that a convex hull can be computed faster if a random partial hull and the corresponding conflict information are known.

Lemma 2.2.5. *Fix $p \in (0,1]$ and let $S \subseteq P \setminus K$ be a subset of size $p(n-k)$. Suppose that $\operatorname{conv}(S \cup K)$ and the conflict sets $B_f \subseteq P$ for $f \in F[S \cup K]$ are available. Then we can find $\operatorname{conv} P$ in expected time $\sum_{f \in F[S \cup K]} b_f \log b_f$. In particular, if S is a random subset, the running time is $O\left((n-k+k/p)\log(1/p)\right)$.*

Proof. Let $\widetilde{S} = S \cup K$. Without loss of generality, we assume that $\operatorname{conv} \widetilde{S}$ contains the origin. Instead of $\operatorname{conv} P$ we compute $(P^*)^\cap$, the intersection of the halfspaces dual to the points in P. For this, we first obtain $(\widetilde{S}^*)^\cap$, which takes linear time, since $\operatorname{conv} \widetilde{S}$ is known. The vertices of $(\widetilde{S}^*)^\cap$ correspond to the facets of $\operatorname{conv} \widetilde{S}$. In particular, each vertex f of $(\widetilde{S}^*)^\cap$ has a conflict list B_f^* of size b_f. We compute a tetrahedralization $\mathcal{T}$ of $(\widetilde{S}^*)^\cap$ as follows: for each facet g of $(\widetilde{S}^*)^\cap$, determine the vertex f_g incident to g with minimum b_{f_g}.[3] The vertex f_g is called the *apex* of g. Triangulate g by adding line segments from the apex to all other vertices of g. Finally, extend this triangulation to a tetrahedralization by lifting it to the origin. This takes linear time.

The conflict set of a simplex s is precisely $B_s^* = B_{f_1}^* \cup B_{f_2}^* \cup B_{f_3}^*$, where f_1, f_2, f_3 are the vertices of s other than the origin. Let $b_s = |B_s|$. We determine the intersection of the halfspaces in B_s^* and clip it to s. Then we glue the parts together to obtain $(P^*)^\cap$, and hence $\operatorname{conv} P$. This takes time $O\left(\sum_{s\in\mathcal{T}} b_s \log b_s\right)$. Consider a simplex $s \in \mathcal{T}$ and let f_s, f_1, f_2 be its vertices other than the origin. Here f_s denotes the apex of the facet of $(\widetilde{S}^*)^\cap$ that contains a facet of s, and we call f_s also the *apex* of s. By definition, we have $b_s = f_s + f_1 + f_2 \leq 2(f_1 + f_2)$, and hence $b_s \log b_s \ll f_1 \log f_1 + f_2 \log f_2$. By general position, every vertex of $(\widetilde{S}^*)^\cap$ has degree 3 and thus appears in only constantly many simplices of $\mathcal{T}$ as a non-apex. Hence, $\sum_{s\in\mathcal{T}} b_s \log b_s \ll \sum_{f\in F[\widetilde{S}]} b_f \log b_f$, as claimed. Now, if S is a random sample, this sum is proportional to $(n - k + k/p)\log(1/p)$, by Lemma 2.2.4(2.5). $\square$

Finally, we would like to investigate the expected conflict size in a given cell of a lower envelope.[4] Let $H = \{h_1, \ldots, h_n\}$ be a set of planes in $\mathbb{R}^3$, and let ℓ be a fixed vertical line. Let $p \in (0, 1]$, and let $S \subseteq H$ be a random sample which contains every h_i with probability p. Let T_S be the canonical triangulation[5] of the lower envelope of S, and Δ_ℓ be the downward vertical prism defined by the facet f_ℓ of T_S that is intersected by ℓ. We want to bound b_ℓ, the number of planes in H that intersect Δ_ℓ.

Lemma 2.2.6. *We have* $\mathbf{E}[b_\ell] \ll 1/p$.

Proof. First, note that every possible facet of T_S is defined by 4, 5, or 6 planes in H, see Figure 2.5. Fix $d \in \{4, 5, 6\}$, and A_d be the event that f_ℓ is defined by d planes. We will show that $\mathbf{E}[b_\ell \mid A_d] \ll 1/p$, from which the result follows by the law of total probability. For $k = 0, \ldots, n$, let r_k denote the number of d-tuples in

[3] Take the lexicographically smallest if there is more than one such vertex.

[4] The *lower envelope* of a set of planes H in $\mathbb{R}^3$ is the two-dimensional surface obtained from H by taking the lowest intersection of the planes in H with every possible vertical line.

[5] In the *canonical triangulation*, all vertices of a facet are connected to the lexicographically smallest vertex in that facet.

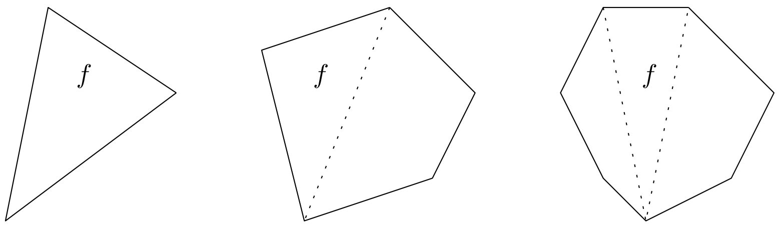


Figure 2.5: Every possible facet f of T_S is defined by 4, 5, or 6 planes.

H^d that define a facet that contains ℓ and whose corresponding downward vertical prism is intersected by exactly k planes in H, and set $r_{\leq k} = \sum_{j=0}^{k} r_j$.

Claim 2.2.7. *We have* $r_{\leq k} \ll (k+2)^d$.

Proof. Take a sample $S' \subseteq H$ by including every plane with probability $1/(k+2)$.[6] Clearly, ℓ intersects only one facet of $T_{S'}$, the canonical triangulation of the lower envelope of S'. Now let $\mathbf{u}$ be a d-tuple defining a facet intersected by ℓ, and let $B_{\mathbf{u}}$ be the planes intersecting the corresponding downward prism. Define $b_{\mathbf{u}} = |B_{\mathbf{u}}|$. Since the probability that $\mathbf{u}$ defines the facet containing ℓ is $(k+2)^{-d}(1-1/(k+2))^{b_{\mathbf{u}}}$, we can write the expected number of those facets as

$$1 \geq \sum_{\mathbf{u}} \left(\frac{1}{k+2}\right)^d \left(1 - \frac{1}{k+2}\right)^{b_{\mathbf{u}}} \geq \sum_{\substack{\mathbf{u} \\ b_{\mathbf{u}} \leq k}} \left(\frac{1}{k+2}\right)^d \left(1 - \frac{1}{k+2}\right)^{k} \gg (k+2)^{-d} r_{\leq k},$$

since $(1-1/(k+2))^k \geq 1/e$. It follows that $r_{\leq k} \ll (k+2)^d$, as claimed. $\square$

Now we can bound $\mathbf{E}\left[b_\ell \mid A_d\right]$, using summation by parts,

$$\begin{aligned}
&\sum_{\mathbf{u}} b_{\mathbf{u}} p^d (1-p)^{b_{\mathbf{u}}} \\
&= p^d \cdot \sum_{k=0}^{n} k r_k (1-p)^k && \text{(group by } b_{\mathbf{u}}\text{)} \\
&= p^d \left(\sum_{k=0}^{n-1} r_{\leq k} \left(k(1-p)^k - (k+1)(1-p)^{k+1} \right) + r_{\leq n} n (1-p)^n \right) && \text{(sum by parts)} \\
&\ll p^d \left(\sum_{k=0}^{n-1} (k+2)^d (1-p)^k (pk - (1-p)) + (n+2)^{d+1} (1-p)^n \right) && \text{(Claim 2.2.7)} \\
&\ll p^{d+1} \sum_{k=0}^{n-1} k^{d+1} (1-p)^k + p^d n^{d+1} (1-p)^n,
\end{aligned}$$

[6]We sample with probability $1/(k+2)$ instead of $1/k$ in order to avoid problems with the corner cases $k = 0, 1$.

since $p \leq 1$. Now, by forming groups of $1/p$ consecutive summands and upper-bounding the t-th group by $p^{-1}(t/p)^{d+1}(1-p)^{(t-1)/p}$, we get

$$\sum_{\mathbf{u}} b_{\mathbf{u}} p^d (1-p)^{b_{\mathbf{u}}} \leq p^d \sum_{t=1}^{\infty} (t/p)^{d+1}(1-p)^{(t-1)/p} + p^d n^{d+1}(1-p)^n$$
$$\leq p^{-1} \sum_{t=1}^{\infty} t^{d+1} e^{-t+1} + p^d n^{d+1}(1-p)^n \ll p^{-1},$$

since a simple calculation shows that $p^{d+1}n^{d+1}(1-p)^n \ll 1$ for $p \in (0,1]$. This completes the proof. □

2.3 On computational models

In the following chapters, we will encounter results for a variety of computational models, so let us take a moment to look at them in more detail. We will deal with the real RAM, the word RAM, the pointer machine, and algebraic computation trees.

Real RAM. The standard machine model in computational geometry is the *real RAM.* Here, data is represented as an infinite sequence of storage cells. These cells can be of two different types: they can store real numbers or integers. The model supports standard operations on these numbers in constant time, including addition, multiplication, and elementary functions like square-root, sine or cosine. Furthermore, the integers can be used as indices to memory locations. Integers can be converted to real numbers in constant time, but we need to be careful about the reverse direction. The *floor* function can be used to truncate a real number to an integer, but if we were allowed to use it arbitrarily, the real RAM could solve PSPACE-complete problems in polynomial time [132]. Therefore, we usually have only a restricted floor function at our disposal, and in this thesis it will be banned altogether.

Word RAM. The *word RAM* is essentially a real RAM without support for real numbers. However, on a real RAM, the integers are usually treated as atomic, whereas the word RAM allows for powerful bit-manipulation tricks. More precisely, the word RAM represents the data as a sequence of w-bit words, where $w = \Omega(\log n)$. Data can be accessed arbitrarily, and standard operations, such as Boolean operations (`and`, `xor`, `shl`, ...), addition, or multiplication take constant time. There are many variants of the word RAM, depending on precisely which instructions are supported in constant time. The general consensus seems to be that any function

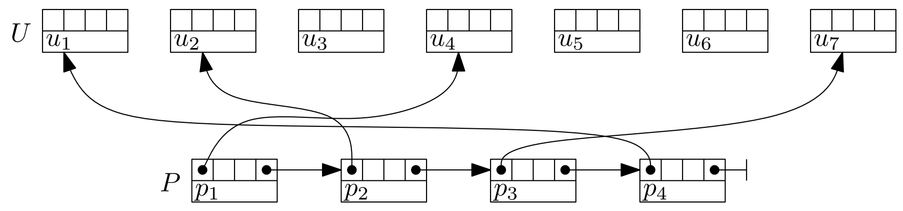


Figure 2.6: Representing a subset P of a universe U on a pointer machine.

in AC^0 is acceptable [12, 145].[7] However, it is always preferable to rely on a set of operations as small, and as non-exotic, as possible. Note that multiplication is not in AC^0 [82], but nevertheless is usually included in the word RAM instruction set [12, 80].

Pointer Machine. The *pointer machine* model [101, 103, 133, 143] disallows the use of constant time table lookup, and is therefore a restriction of the (real) RAM model. The data structure is modeled as a directed graph G with bounded out-degree. Each node in G represents a *record*, with a bounded number of pointers to other records and a bounded number of (real or integer) data items. The algorithm can access data only by following pointers from the inputs (and a bounded number of global entry records); random access is not possible. The data can be manipulated through the usual real RAM operations, but without support for the floor function, for reasons mentioned above.

In Chapter 3, we will consider pointer machine algorithms for subsets of a universe U of points that is known in advance. This is represented as follows: for each point in U there is a record storing its coordinates, and the input subsets are provided as a linked list of records, each pointing to the record for the corresponding input. This gives the elements in the input data a certain identity which can be exploited, see Figure 2.6. The output (a convex hull or a Delaunay triangulation) is provided as a DCEL [23, Chapter 2.2].

Algebraic Computation Tree. *Algebraic computation trees* (ACTs) [14, 22] are the computational geometry analogue of binary decision trees, and like these they are mainly used for proving lower bounds. Let $x_1, \dots, x_n \in \mathbb{R}$ be the inputs. An ACT is a binary tree with two different kinds of nodes: *computation nodes* and

[7]AC^0 is the class of all functions $f : \{0,1\}^* \to \{0,1\}^*$ that can be computed by a family of circuits $(C_n)_{n \in \mathbb{N}}$ with the following properties: (i) each C_n has n inputs; (ii) there exist constants a, b, such that C_n has at most an^b gates, for $n \in \mathbb{N}$; (iii) there is a constant d such that for all n the length of the longest path from an input to an output in C_n is at most d (ie, the circuit family has bounded depth); (iv) each gate has an arbitrary number of incoming edges (ie, the fan-in is unbounded).

branch nodes. A computation node v has one child and is labeled with an expression of the type $y_v = y_u \oplus y_w$, where $\oplus \in \{+, -, *, /, \sqrt{\cdot}\}$ is a operation and y_u, y_w is either an input variable $x_1, \ldots, x_n$ or corresponds to a computation node that is an ancestor of v. A branch node has degree 2 and is labeled by $y_u = 0$ or $y_u > 0$, where again y_u is either an input or a variable corresponding to an ancestor. A family of algebraic computation trees $(T_n)_{n \in \mathbb{N}}$ solves a computational problem (like Delaunay triangulation or convex hulls computation), if for each $n \in \mathbb{N}$, the tree T_n accepts inputs of size n, and if for any such input $x_1, \ldots, x_n$ the corresponding path in T_n (where the children of the branch nodes are determined according the conditions they represent) constitutes a computation which represents the answer in the variables y_v encountered during the path.

Part I

The Utility of Additional Structure

Chapter 3

Hereditary Structure

We begin by exploring how hereditary structure in the inputs can lead to faster algorithms. Suppose we are given a planar n-point set and its Delaunay triangulation (DT). Then we can find the DT of any given subset in linear time[1], as shown by Chazelle et al. [47]. Since planar Delaunay triangulations are a special case of three-dimensional convex hulls (see Section 2.1), it is natural to ask whether a similar result can be proven for the convex hull of an arbitrary subset of the vertices of a convex 3-polytope, and we will soon see that the answer is affirmative. We formulate the problem in a *hereditary* setting by assuming that the vertices of a convex polytope $\mathcal{P}$ in $\mathbb{R}^3$ are colored red and blue. The problem is then to "split" $\mathcal{P}$ and compute both monochromatic convex hulls. We show how to do this in linear time, which answers the main open question of Chazelle et al. [47]. This result can also be interpreted as saying that the convex hull problem in $\mathbb{R}^3$ loses its $\Omega(n \log n)$-hardness if it is embedded in a larger polytope. In other words, computationally speaking, a convex polytope "gives away" the convex hull of any of its subsets. We will also discuss how to extend our result in several interesting ways:

- **Multiple colors.** If the vertices of $\mathcal{P}$ are colored with χ colors, we are able to compute the convex hulls of all the color classes in $O(n(\log\log n)^2)$ time. If the coloring is random, we can do it in linear time. We emphasize that the result holds for any $\chi \in \{1, \ldots, n\}$. Note that a straightforward application of binary splitting yields an $O(n \log \chi)$ time algorithm.

- **Data structure version.** The splitting algorithm needs time linear in the size of $\mathcal{P}$, but suppose we want to find the convex hulls for many different subsets of the vertices of $\mathcal{P}$. In this case, we can preprocess $\mathcal{P}$ so that the convex hull of any vertex set S can be found in time $O(|S|(\log\log|\mathcal{P}|)^2)$.

[1]All our algorithms are randomized, so the complexity is to be understood in the expected sense. Note that the expectation is only over the randomness used by the algorithm and that the complexity bounds hold for every input.

- **Colorings induced by halfspaces.** Consider the coloring induced by halfspace range queries: given a query plane, compute the convex hull of the points on one side. We describe how to do so in time $O(k + \log n)$, where k is the output size; the data structure requires $O(n \log n)$ storage.

- **Colorings with few connected components.** Suppose the blue vertices form k connected components in the skeleton graph of $\mathcal{P}$. Then we can find their convex hull in time $O(n \log^* n + k \log k)$, where n now is the size of the subset. Our result has this intriguing corollary: given a DT $\mathcal{T}$, the DT of any set S of n vertices and edges in $\mathcal{T}$ can be computed in time $O(n \log^* n + k \log k)$, where k is the number of connected components formed by S within $\mathcal{T}$. We actually prove a slightly more general result. It is well known that the convex hull of two convex polytopes can be stitched together in linear time [45]. We consider the case of k disjoint convex polytopes with a total of n vertices. If the vertices of each polytope form a connected component in the convex hull of their union, we can compute their common convex hull in $O(n \log^* n + k \log k)$ time. This assumption is motivated by a lower bound of $\Omega(n \log k)$ for the general case.

Previous Work. The study of hereditary structure is part of a broader attempt to understand *what makes what hard.* To compute the DT of n points in the plane requires $\Omega(n \log n)$ time, but knowing that the points are the vertices of a convex polygon cuts down the complexity to linear [2, 49]. Given a spanning subgraph of degree at most d, the DT can be completed in time $O(nd \log^* n)$ [63]. In fact, at the cost of a more complicated algorithm, it can be done in linear time [52, 100]. Furthermore, Djidjev and Lingas have proven linearity for any set of points forming a monotone chain in both x and y directions [67]. This might suggest that the hardness of DT is really confined to sorting. Of course, we know this is not true: in the general Euclidean case, sorting does not help (though it does in ℓ_∞ [50]). Ranking the points in any one direction still leaves us with a $\Theta(n \log n)$ complexity [67]. The simplicity of a polygon is known to "linearize" many problems that otherwise exhibit $\Omega(n \log n)$ lower bounds, eg, polygon triangulation [7,44,138], medial axis [51], or constrained Delaunay triangulation [52, 100].

Hereditary algorithms are nothing new. Given a subset of a simple polygon, Chan [38] has shown how to compute its convex hull in linear time[2] and how to triangulate it in $O(n \log^* n)$ time. Van Kreveld, Löffler, and Mitchell [106] improved the latter result by proving that any subset of a given triangulation can in fact be triangulated in linear time. To appreciate the difficulty of obtaining general hereditary algorithms, let us mention the example of hereditary trapezoidal decompositions [38, 99]. Kirkpatrick, Klawe, and Tarjan [99] gave an algorithm for removing

[2]Here, *linear time* means linear in the size of the whole structure, not just the subset.

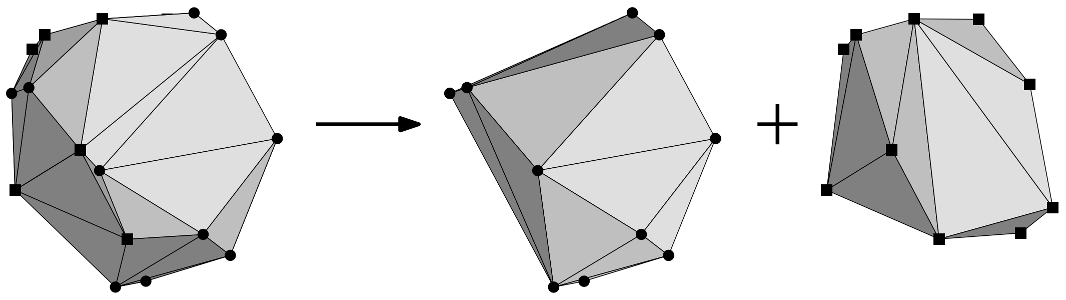


Figure 3.1: Given their joint convex hull, we can find the red and blue hulls in linear time.

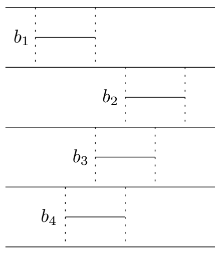


Figure 3.2: General hereditary trapezoidal decompositions are hard.

a *hole-free* subset of line segments in a trapezoidal decomposition in linear time, where hole-freeness is a property that is necessary to ensure that the subset does not obscure too much information. They also give an example that for general hereditary trapezoidal decompositions no improvement is possible (see also [38]). Consider the line segments in Figure 3.2, and their trapezoidal decomposition. Suppose we would like to find the trapezoidal decomposition of b_1, b_2, b_3, b_4. To achieve this, we essentially have no choice but to sort their endpoints from scratch, since the long line segments obscure all information. This means that sometimes trapezoidal decompositions do not give away anything about some of their subsets, unlike convex hulls. There are many other situations in which additional "hereditary" information brings no benefits: if P is a point set in $\mathbb{R}^3$, sorting P in a bounded number of directions does not help in computing its convex hull [137]; nor does knowing the convex hull of P help in finding its diameter [76].

3.1 Splitting polytopes

We are given an n-point set $P \subseteq \mathbb{R}^3$ in gcp. Let $B \subseteq P$ and let $R = P \setminus B$. The points in B are called *blue*, the points in R are called *red*. Given the convex hull

Algorithm 3.1 Splitting a bichromatic convex hull.

```
SplitHull(conv $P$)

  1. If $P$ contains no red points, return conv $P$.
  2. If there exists a red point $r$ in $P$ with $\deg_P r \leq d_0$ (with a suitable constant
     $d_0$), then return SplitHull(conv $(P \setminus r)$).
  3. Take random blue points $b \in B$ until (i) $\deg_P b \leq 6$; and (ii) there exists a
     blue edge $e$ in conv $(P \setminus b)$ that is visible from $b$.
  4. Call SplitHull(conv $(P \setminus b)$) to compute conv $(B \setminus b)$.
  5. Using $e$ as a starting edge, insert $b$ into conv $(B \setminus b)$ and return conv $B$.
```

conv P, we show that it is possible to obtain the individual hulls conv B and conv R in linear time.

Theorem 3.1.1. *Let $P \subseteq \mathbb{R}^3$ be a set of n points in gcp, colored red and blue. Given* conv P*, the convex hull of the blue points can be computed in $O(n)$ expected time.*

An edge of conv P is called *blue* if both of its endpoints are blue, and *red* if both of its endpoints are red, otherwise it is *bichromatic*. Blue, red, and bichromatic facets are defined similarly. The splitting is performed by a recursive algorithm `SplitHull` that receives the convex hull and a two-coloring of P. Please refer to Algorithm 3.1. `SplitHull` can be seen as a generalization of Chew's algorithm for Voronoi diagrams of convex polygons [49], and it is also reminiscent of Dobkin and Kirkpatrick's hierarchy [68, 69]. It first tries to delete a red point of small degree. If this is not possible, it removes blue points until there is a red point of small degree again. Later, these blue points must be reinserted into the recursively computed blue hull. In order to do this efficiently, we must be careful about which blue points we delete, so that we have a landmark from where to start the conflict location. `SplitHull` is easily shown to be correct.

Lemma 3.1.2. *`SplitHull`(*conv P*) computes* conv B*.*

Proof. The proof is by straightforward induction on $|P|$. We only comment on Step 5. Let $B^- = B \setminus b$ and $P^- = P \setminus b$. If e is a blue edge visible from b in conv P^-, then the same holds in conv B^-: since e has both endpoints in B^-, a supporting plane for e in conv P^- supports e also in conv B^-, and since conv $B^- \subseteq$ conv P^-, the triangle spanned by b and e intersects conv B^- only in e. Thus, we can walk from e to determine b's conflict set D_b and replace D_b by new facets incident to

b. This takes time $O(|D_b|)$ [23, Chapter 11.2]. When implementing the algorithm, care must be taken that the pointer to e obtained in Step 3 is not invalidated by the recursive call in Step 4. We can easily do it as follows: when deleting a blue edge in Step 4, retain the corresponding record in memory and reuse it when the edge is recreated in Step 5. □

The bulk of the analysis lies in bounding the running time.

Lemma 3.1.3. *The expected time needed for one invocation of* `SplitHull` *is constant, not counting the time for the recursive calls.*

Proof. We argue that each step takes constant expected time. This clearly holds for Step 1: just use a counter for the number of red points. Step 2 is also easy: keep a linked list L for the red points with degree at most d_0. During preprocessing, determine the degrees and initialize L accordingly. When the hull is altered in Steps 2 and 4, update the degrees and L. Since all relevant vertices have bounded degree, this takes constant time. The most interesting part lies in the analysis of Step 3. We show that there is a good chance of sampling a point with the required properties.

Lemma 3.1.4. *Let $\widetilde{B}$ be the subset of the blue points b with the following properties: (i) $\deg_P b \leq 6$; and (ii) b is a vertex of a blue facet of* $\operatorname{conv} P$ *or $E[P \setminus b] \setminus E[P]$ contains at least one blue edge.*[3] *There exists a constant d_0 such that if all red points have degree at least d_0, then $|\widetilde{B}| \geq |P|/5$.*

Proof. Call a blue point *pleasant* if it satisfies the properties in the lemma, and *ghastly* otherwise. By Euler's formula, a large fraction of blue points has degree at most 6. If a blue point b is ghastly and has degree at most 6, then either (a) b is incident to a facet with a red edge; or (b) b's neighborhood has only bichromatic edges and to delete b from $\operatorname{conv} P$ creates no blue edge. We bound the number of points satisfying (a) and (b) separately and then finish the analysis with a union bound.

In the following, we will assume that d_0 is a large enough constant. By gcp, we have $|E[P]| = 3n - 6$.[4] Let B' be the set of blue points b with $\deg_P b \leq 6$. Since $\operatorname{conv} P$ is three-connected [112, Theorem 5.3.3], and since all red nodes have degree at least $d_0 \geq 7$, we get

$$6n - 12 = \sum_{p \in B'} \deg p + \sum_{p \in P \setminus B'} \deg p \geq 3|B'| + 7(n - |B'|).$$

[3]Recall that $E[P], F[P]$ denote the edges and facets of $\operatorname{conv} P$ (see Section 2.1).

[4]Since all the points are on the hull, Euler's formula [31, Theorem 7.2.1] yields $n - |E[P]| + |F[P]| = 2$, and since all facets are triangles, we have $2|[E[P]| = 3|F[P]|$.

Thus

$$|B'| > n/4. \tag{3.1}$$

Similarly,

$$6n - 12 = \sum_{p \in R} \deg p + \sum_{p \in P \setminus R} \deg p \geq d_0|R| + 3(n - |R|) = (d_0 - 3)|R| + 3n.$$

so $|R| < 4n/d_0$ (for $d_0 \geq 12$). Let E_R denote the set of red edges in $\operatorname{conv} P$. Since every red edge of $\operatorname{conv} P$ is an edge of $\operatorname{conv} R$,

$$|E_R| \leq |E[R]| = 3|R| - 6 < 12n/d_0. \tag{3.2}$$

For $b \in B'$, let Γ_b be the simple polygon formed by b's neighbors in $\operatorname{conv} P$, and let C be the set of points $b \in B'$ such that Γ_b contains a red edge (this corresponds to the property (a) mentioned at the beginning of the proof). Since an edge is incident to two facets, for each $e \in E_R$ there are at most two points $p, q \in C$ such that e is in Γ_p and Γ_q. Hence, by (3.2),

$$|C| \leq 2|E_R| < 24n/d_0. \tag{3.3}$$

Now, let $D \subseteq B'$ be the set of points b such that Γ_b has no monochromatic edge. For any such b, $\deg_P b$ is even and red and blue points alternate along Γ_b. Let $E_b = E[P \setminus b] \setminus E[P]$. We say that b *creates* E_b. Note that E_b contains only diagonals of Γ_b. Any edge e is created by at most two points in D: if e is occluded in $\operatorname{conv} P$ by exactly one edge, it is created by the endpoints of this edge; if e is occluded by two or more edges, it can only be created by a point incident to all of them; see Figure 3.3. Furthermore, every $b \in D$ creates at least one monochromatic

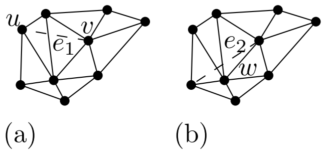


Figure 3.3: (a) The edge e_1 is occluded by exactly one edge and is created by u and v; (b) the edge e_2 is occluded by two edges and is created only by w.

edge, since every triangulation of a two-colored simple polygon contains at least one monochromatic diagonal[5]; see Figure 3.4. Let D' be the set of points in D that do not create a blue edge (these are the points with property (b)). By the previous discussion and (3.2),

$$|D'| \leq 2|E[R]| < 24n/d_0. \tag{3.4}$$

[5]Since the dual graph of this triangulation is a tree [23, Section 3.1], and every tree contains at least one leaf, corresponding to a triangle between two adjacent edges.

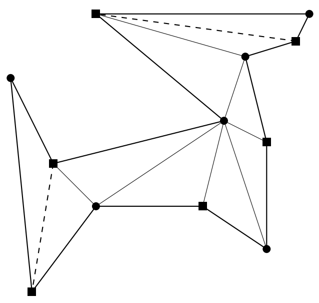


Figure 3.4: Every triangulation of a two-colored simple polygon contains at least one monochromatic diagonal (shown in dashed).

To conclude, we observe that all the points in the set $B' \setminus (C \cup D')$ are pleasant and that by (3.1, 3.3, 3.4) it contains at least $(1/4 - 48/d_0)\, n > n/5$ points, for d_0 large enough. $\square$

By Lemma 3.1.4 we expect at most five iterations in Step 3, each taking constant time, since all points under consideration have bounded degree. The same holds for Step 4 without the recursive call, as $\deg_P b \leq 6$. Finally, we use backwards analysis to handle Step 5. Take $\widetilde{B}$ as in Lemma 3.1.4. Because $|\widetilde{B}| > |B|/5$, the average degree of a point in $\widetilde{B}$ is less than 30, by Euler's formula. Hence, to delete a random point $b \in \widetilde{B}$ from $\operatorname{conv} B$ takes constant expected time, and this is exactly the cost of inserting b into $\operatorname{conv}(B \setminus b)$ [23, Chapter 11.2]. $\square$

Theorem 3.1.1 follows from Lemmas 3.1.2 and 3.1.3, since the number of recursive calls is $O(n)$.

3.2 Handling multiple colors

Now, we extend `SplitHull` to handle more than two colors: for a point set $P \subseteq \mathbb{R}^3$, let $c : P \to \{1, \dots, \chi\}$ be a *coloring* of P. For $i \in \{1, \dots, \chi\}$, we let $C_i = c^{-1}(i)$ denote the points that are colored i, the ith *color class*. First, we note an easy consequence of `SplitHull`:

Proposition 3.2.1. *Let $P \subseteq \mathbb{R}^3$ be an n-point set in gcp, and let $c : P \to \{1, \dots, \chi\}$ be a coloring of P. Given $\operatorname{conv} P$, we can find the convex hulls $\operatorname{conv} C_1, \dots, \operatorname{conv} C_\chi$ in expected time $O(n \log \chi)$.*

Proof. If $\chi = 1$, there is nothing to do, and if $\chi = 2$, we can just use `SplitHull`. Otherwise, let $t = \lfloor \chi/2 \rfloor$, and let $C_a = \bigcup_{i=1}^{t} C_i$ and $C_b = \bigcup_{i=t+1}^{\chi} C_i$. Use `SplitHull` to obtain $\operatorname{conv} C_a$ and $\operatorname{conv} C_b$, and then recurse. Since the total work in each round is $O(n)$ and the number of colors halves in each step, the result follows. $\square$

In the following sections we shall see how to improve upon Proposition 3.2.1. We will first consider *random* colorings, where the coloring c is called *random*, if each point p is colored uniformly and independently with a color in $\{1, \dots, \chi\}$. For such colorings, we can split the convex hull in expected linear time, no matter the value of χ. After that, we will look at the harder case of arbitrary colorings. For these, we will see a splitting algorithm with $O(n\sqrt{\log n})$ running time, which will later be improved to $O(n(\log\log n)^2)$, as an application of the data structure version of the splitting theorem presented in Section 3.3.

3.2.1 Random colorings

The goal of this section is to prove the following theorem.

Theorem 3.2.2. *Let $P \subseteq \mathbb{R}^3$ be a set of n points in gcp, and let $c : P \to \{1, \dots, \chi\}$ be a random coloring of P. Given $\operatorname{conv} P$, we can compute the convex hulls $\operatorname{conv} C_1, \dots, \operatorname{conv} C_\chi$ in $O(n)$ expected time (the expectation is over the coloring and the random choices of the algorithm).*

The algorithm for Theorem 3.2.2 is called `RandMultiSplit`. See Algorithm 3.2. It receives the convex hull and a coloring of P as input, and it computes the convex hull of a random sample $S \subseteq P$ into which the points of each color class are then inserted separately. As we will see below, this can be done quickly because c is random. Finally, it uses `SplitHull` to remove the points from S.

Algorithm 3.2 Splitting random colorings.

```
RandMultiSplit(conv $P$) (* see Figure 3.5 *)

  1. Pick a random sample $S \subseteq P$ of size $n/\chi$ and compute $\operatorname{conv} S$.
  2. For each $p \in P$, determine a facet $f_p \in F[S]$ in conflict with $p$.
  3. For each color $i$:
     (a) Insert all points of $C_i$ into $\operatorname{conv} S$.
     (b) Extract $\operatorname{conv} C_i$ from $\operatorname{conv}(C_i \cup S)$.
```

Clearly, the algorithm correctly computes the $\operatorname{conv} C_i$. We bound the running time of each step. Using `SplitHull`, Step 1 requires $O(n)$ time. The analysis of Step 2 needs more work.

Lemma 3.2.3. *Step 2 takes $O(n)$ expected time.*

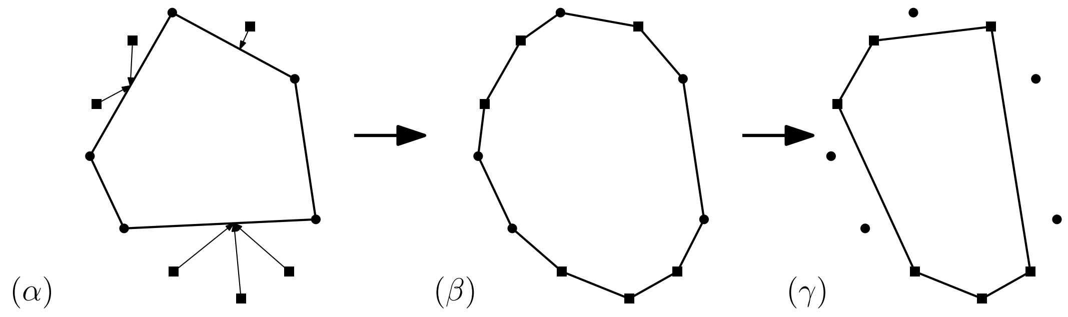


Figure 3.5: Splitting random colorings: the algorithm (α) computes $\operatorname{conv} S$ and conflict facets for C_i, (β) inserts C_i into $\operatorname{conv} S$, and (γ) extracts $\operatorname{conv} C_i$. The points in C_i are shown as boxes, S as circles.

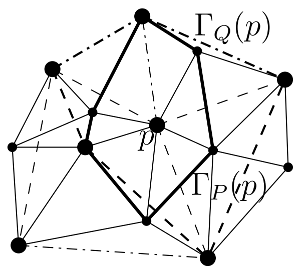


Figure 3.6: Claim 3.2.4: the facets $F[Q]$ are shown dashed, $F[P]$ solid. Merge $\Gamma_P(p)$ with $\Gamma_Q(p)$ to determine its conflict facets.

Proof. For $Q \subseteq P$ and $p \in Q$, let $\Gamma_Q(p)$ denote the neighbors of p in $\operatorname{conv} Q$. First, we show how to compute the conflict facets for points that are neighbors in $\operatorname{conv} P$ of a point in Q.

Claim 3.2.4. *Let $Q \subseteq P$ and $p \in Q$. Assume that both $\operatorname{conv} Q$ and $\operatorname{conv} P$ are available. In $O(\deg_Q p + \deg_P p)$ time, we can compute a conflict facet $f_q \in F[Q]$ for every neighbor $q \in \Gamma_P(p)$ of p.*

Proof. Consider an *overlay* of $\operatorname{conv} Q$ and $\operatorname{conv} P$, ie, a central projection of their vertices and edges onto the unit sphere centered at a point $O \in \operatorname{conv} Q$. Let $q \in \Gamma_P(p)$ and let $f \in F[Q]$ be the facet incident to p that is intersected by the line segment pq in the overlay. Then q is in conflict with f. To see this, let h_f be the plane supporting f. If q did not conflict with f, then q would lie in h_f^- and at least part of the line segment pq would be strictly inside $\operatorname{conv} Q$. But then pq could not be an edge of $\operatorname{conv} P$, as $\operatorname{conv} Q \subseteq \operatorname{conv} P$. Thus, conflict facets for $\Gamma_P(p)$ can be computed by merging the cyclically ordered lists $\Gamma_P(p)$ and $\Gamma_Q(p)$ with respect to some overlay of the hulls; see Figure 3.6. This takes time $O(\deg_Q p + \deg_P p)$. □

Algorithm 3.3 Determining the conflict facets in a subset.

```
SubsetConflictWalk($\mathrm{conv}\, S, \mathrm{conv}\, P$)

  1. Let queue be a queue with the elements in $S$.
  2. While queue $\neq \emptyset$.
     (a) Let $p$ be the next point in queue.
     (b) If $p \notin S$, insert $p$ into $\mathrm{conv}\, S$, using a previously computed conflict facet
         $f_p$ for $p$ as a starting point.
     (c) For each neighbor $q \in \Gamma_P(p)$, find a conflict facet $\tilde{f}_q$ in $\mathrm{conv}\,(S \cup p)$, using
         Claim 3.2.4.
     (d) Using the $\tilde{f}_q$'s, find conflict facets $f_q \in F[S]$ for all $q \in \Gamma_P(p)$. If $q \in$
         $\Gamma_P(p)$ has not been encountered yet, insert it into queue.
```

The conflict facets for P can now be found by breadth-first search, using the algorithm `SubsetConflictWalk`. Please refer to Algorithm 3.3. Step 2 of the algorithm maintains the invariant that a conflict facet $f_p \in F[S]$ is known for each $p \in$ `queue` $\setminus S$. Using standard techniques, Step 2b takes $O(d_p)$ time, where d_p is the conflict size of p in $\mathrm{conv}\, S$ [23, Chapter 11.2].[6] Furthermore, by Claim 3.2.4, the conflict facets of $\Gamma_P(p)$ can be found in $O(\deg_{S\cup p} p + \deg_P p)$ time. Finally, Step 2d takes time $O(\deg_P p)$: every facet $\tilde{f} \in F[S \cup p]$ shares at least one edge e with an $f \in F[S]$, and if q can see e in $\mathrm{conv}\, S$, it conflicts with at least one facet adjacent to e. Thus, f_q can be computed from $\tilde{f}_q$ in constant time. It follows that the total running time of `SubsetConflictWalk` is proportional to

$$\mathbf{E}\left[\sum_{p\in P}\left(d_p + \deg_{S\cup p} p + \deg_P p\right)\right].$$

Now, since[7] $\deg_{S\cup p} p \ll d_p$ for $p \notin S$, this is proportional to

$$\mathbf{E}\left[\sum_{p\in S}\deg_S p + \sum_{p\in P\setminus S} d_p + \sum_{p\in P}\deg_P p\right] \ll \mathbf{E}\left[\frac{n}{\chi} + \sum_{f\in F[S]} b_f + n\right],$$

by (2.1) in Appendix 2.1. The lemma follows, since $\mathbf{E}\left[\sum_{f\in F[S]} b_f\right] \ll n$ by Lemma 2.2.4(2.4) (b_f is the conflict size of f). □

[6] $d_p = 0$ if $p \in S$.

[7] Recall that we use the Vinogradov notation $f \ll g$ for $f = O(g)$ and $f \gg g$ for $f = \Omega(g)$.

Now we consider Step 3 of `RandMultiSplit`. Fix a color i, and for each $f \in F[S]$, let $a_f = |C_i \cap B_f|$. Since the coloring is random, conditioned on b_f, the size a_f is distributed like a sum of independent Bernoulli random variables with mean b_f/χ. By standard moment bounds [46, Lemma A.1], $\mathbf{E}_c[a_f^2] \ll (b_f/\chi)^2$. By Lemma 2.2.5, Step 3a takes time $\mathbf{E}_{S,c}\left[\sum_{f\in F[S]} a_f \log a_f\right]$, and by Lemma 2.2.4(2.4), we get

$$\mathbf{E}_{S,c}\left[\sum_{f\in F[S]} a_f \log a_f\right] \ll \mathbf{E}_S\left[\sum_{f\in F[S]} \mathbf{E}_c\left[a_f^2\right]\right] \ll \mathbf{E}_S\left[\frac{1}{\chi^2}\sum_{f\in F[S]} b_f^2\right] \ll \frac{\chi n}{\chi^2} = \frac{n}{\chi}.$$

Using `SplitHull` in Step 3b, $\operatorname{conv} C_i$ can now be computed in time $O(|C_i| + n/\chi)$. There are χ colors, so Step 3 takes total time proportional to $\sum_i |C_i| + \chi \cdot (n/\chi) \ll n$, and Theorem 3.2.2 follows.

3.2.2 Arbitrary colorings

We now consider arbitrary colorings. For the random colorings in the previous section, we could exploit the fact that each color is spread uniformly over the polytope in order to design a simple divide and conquer algorithm that decomposes each color class into subsets of expected constant size. This is no longer possible for arbitrary colorings, because now the distribution of color classes can be highly irregular. Therefore, we need a more sophisticated scheme to partition the color classes.

Here, we will prove the following theorem, which serves as a basis to the data structure version in Section 3.3. There, we will see how Theorem 3.2.5 can be improved through a more complicated bootstrapping scheme.

Theorem 3.2.5. *Let $P \subseteq \mathbb{R}^3$ be a set of n points in gcp, and let $c : P \to \{1, \dots, \chi\}$ be an arbitrary coloring of P. Given* $\operatorname{conv} P$*, we can compute* $\operatorname{conv} C_1, \dots, \operatorname{conv} C_\chi$ *in $O\left(n\sqrt{\log n}\right)$ expected time.*

We begin with a useful sampling lemma.

Lemma 3.2.6. *Let $Q \subseteq \mathbb{R}^3$ be an m-point set in gcp, and let $\mu \in (0, 1)$ be a constant. There exists a constant α_0 such that the following holds: let $\alpha \in \{\alpha_0, \dots, \mu m\}$. Given* $\operatorname{conv} Q$*, in $O(m)$ time we can compute subsets $S, R \subseteq Q$ and a partition $R_1, \dots, R_\beta$ of R such that*

1. *$|S| = \alpha$, $|R| \gg m$, and* $\max_i |R_i| \ll m(\log \alpha)/\alpha$.
2. *For each R_i, there exists a facet $f_i \in F[S]$ such that all points in R_i are in conflict with f_i.*

3. Every point in R conflicts with constantly many facets of $\operatorname{conv} S$.

4. The conflict sets for two points $p \in R_i$, $q \in R_j$, $i \neq j$, are disjoint and no conflict facet of p shares an edge with a conflict facet of q.

Furthermore, the convex hulls $\operatorname{conv} S$, $\operatorname{conv} R_1, \dots, \operatorname{conv} R_\beta$, $\operatorname{conv}(Q \setminus (R \cup S))$ *can be computed in expected $O(m)$ time.*

Proof. We call a subset $S \subseteq Q$ *decent* if it has two properties: (i) $\sum_{f \in F[S]} b_f \ll m$; and (ii) $\max_{f \in F[S]} b_f \ll m(\log \alpha)/\alpha$, where b_f denotes the conflict size of f.

Claim 3.2.7. *A decent subset $S \subseteq Q$ of size α together with* $\operatorname{conv} S$ *and the conflict sets B_f, $f \in F[S]$, can be found in expected time $O(m)$.*

Proof. Let S be a random α-subset of Q. We claim that S is decent with probability at least 1/2. To see this, we first use Lemma 2.2.4(2.4) with $\gamma = 1$ to obtain $\mathbf{E}\left[\sum_{f \in F[S]} b_f\right] \ll m$. By Markov's inequality, it follows that $\sum_{f \in F[S]} b_f \ll m$ with probability at least 3/4. Furthermore, using Lemma 2.2.2 with $pn = \alpha$ and $t = 2\log\alpha$, we get[8] $\mathbf{E}\left[|F_{\geq 2\log\alpha}|\right] \ll (\log^2\alpha)/\alpha$, and if α_0 is large enough, this expected value is less than 1/4. Hence, by Markov's inequality, the probability that there exists a facet with conflict size at least $2m(\log\alpha)/\alpha$ is at most 1/4. So, we have $\max_{f \in F[S]} b_f \ll m(\log\alpha)/\alpha$ with probability at least 3/4, and the claimed probability follows from a union bound.

Furthermore, a decent sample can be verified in $O(m)$ time: by the proof of Lemma 3.2.3, we can find the conflict sets B_f and D_p in time $O\left(m + \sum_{f \in F[S]} b_f\right)$. Hence, we can run the algorithm of Lemma 3.2.3 on the sample S. If the number of steps exceeds cm, for a certain constant c that comes from the proof of Lemma 3.2.3, we abort the computation and reject the sample, since it cannot be decent. Otherwise, we can check in $O(m)$ time that $\max_{f \in F[S]} b_f \ll m(\log\alpha)/\alpha$, as required. Consequently, since a sample is decent with constant probability, repeated sampling yields the desired result. $\square$

Now let S be a decent sample, and let B_f, $f \in F[S]$, denote its conflict sets. By (2.1) and Property (i) of a decent sample, we have $\sum_{p \in Q} d_p \ll m$, and hence there exists a constant λ such that the set $X = \{p \in Q \mid d_p > \lambda\}$ has cardinality at most $(1-\mu)m/2$. Let $R' = Q \setminus (S \cup X)$, $B'_f = B_f \cap R'$ and $b'_f = |B'_f|$ for $f \in F[S]$. By definition, all points in R' conflict with at most λ facets. We now prune $F[S]$ to obtain a subset $\mathcal{F}$ of facets whose conflict sets constitute the desired partition. For $f, g \in F[S]$, let $\delta(f,g)$ denote the BFS-distance between f and g in the dual graph of $\operatorname{conv} S$;[9] see Figure 3.7. The pruning is done by a greedy algorithm `PruneFS`,

[8]Note that by choosing α_0 large enough, we can ensure that $t = 2\log\alpha \leq \alpha/4 = pn/4$.

[9]More precisely, the *BFS-distance* (BFS = **B**readth **F**irst **S**earch) between f and g is the length of a shortest path between f and g in the graph with vertex set $F[S]$ in which two vertices are adjacent precisely if the corresponding facets share an edge in $\operatorname{conv} S$.

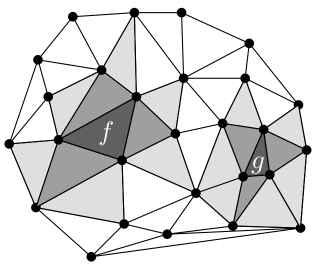


Figure 3.7: The pruning step: remove all facets at distance at most 2λ from a facet with maximum conflict size. The points in B'_f, B'_g conflict only with the darker facets at distance at most $\lambda = 1$.

Algorithm 3.4 Pruning the conflict facets.

```
PruneFS

  1. Let $\mathcal{F} = \emptyset$ and let queue be a priority queue containing the facets in $F[S]$.
  2. While queue $\neq \emptyset$:
      (a) Let $f$ be a facet in queue with maximum $b'_f$, and let $N_f = \{f' \in F[S] \mid \delta(f, f') \leq 2\lambda\} \cap$ queue.
      (b) Let queue = queue $\setminus N_f$ and $\mathcal{F} = \mathcal{F} \cup \{f\}$.
```

which iteratively takes the facet with the largest conflict size and discards all of its neighbors. For details see Algorithm 3.4. Clearly, `PruneFS` takes $O(m)$ time. Let $f_1, \ldots, f_\beta$ be the facets in $\mathcal{F}$ as computed by `PruneFS`, and let $R_1, \ldots, R_\beta$ be the corresponding conflict sets with respect to R'. Set $R = \bigcup_{i=1}^{\beta} R_i$.

Claim 3.2.8. *We have $|R| \gg m$, the R_i constitute a partition of R, and for $p \in R_i, q \in R_j$, $i \neq j$, we have $D_p \cap D_q = \emptyset$ and no facet in D_p shares an edge with a facet in D_q.*

Proof. To see that $|R| \gg m$, note that $|N_f| \ll 1$ and $b'_{f'} \leq b'_f$ for every $f' \in N_f$. Thus, we have $b'_f \gg \sum_{f' \in N_f} b'_{f'}$, and therefore

$$|R| = \sum_{f \in \mathcal{F}} b'_f \gg \sum_{f \in \mathcal{F}} \sum_{f' \in N_f} b'_{f'} \geq |R'| \gg m.$$

To see that $(R_i)_{1 \leq i \leq \beta}$ is a partition, consider two sets R_i, R_j with $i \neq j$, and let f_i, f_j be the corresponding facets. Any point $p \in R$ has $|D_p| \leq \lambda$, and D_p is connected in the dual graph of $\operatorname{conv} S$. By construction, we have $\delta(f_i, f_j) > \lambda$, so there cannot be a point in conflict with both f_i and f_j. It follows that $R_i \cap R_j = \emptyset$, since R_i and R_j are the conflict sets of f_i and f_j. Similarly, we see that D_p, D_q are disjoint for $p \in R_i$, $q \in R_j$, and no facet in D_p is adjacent to a facet in D_q, because $\delta(f_i, f_j) > 2\lambda$ and D_p, D_q are connected with size at most λ.

Furthermore, to see that $|R| \gg m$, note that $|N_f| \ll 1$ and $b'_{f'} \leq b'_f$ for every $f' \in N_f$. Thus, we have $b'_f \gg \sum_{f' \in N_f} b'_{f'}$, and therefore

$$|R| = \sum_{f \in \mathcal{F}} b'_f \gg \sum_{f \in \mathcal{F}} \sum_{f' \in N_f} b'_{f'} \geq |R'| \geq m - \alpha - (1-\mu)m/2 \geq$$
$$m - \mu m - (1-\mu)m/2 = (1-\mu)m/2 \gg m.$$

□

By now, we have established statements 1–4 of Lemma 3.2.6. It remains to show how to find all the convex hulls quickly. First, using `SplitHull`, we can compute $\operatorname{conv} S$, $\operatorname{conv}(R \cup S)$ and $\operatorname{conv}(Q \setminus (R \cup S))$ in time $O(m)$. It remains to consider the R_i's.

Claim 3.2.9. *For $i = 1, \ldots, \beta$, the convex hull $\operatorname{conv} R_i$ can be computed in $O(|R_i|)$ time.*

Proof. Consider an R_i, and let f_i be the corresponding facet in $\operatorname{conv} S$. First, note that the subgraph of $\operatorname{conv}(R \cup S)$ induced by R_i is connected, because $R_i = R \cap h^+_{f_i}$. Let Γ denote the points in $(R \cup S) \setminus R_i$ that are adjacent in $\operatorname{conv}(R \cup S)$ to a point in R_i. We have $\Gamma \subseteq S$: if there were two points $p \in R_i$, $q \in R_j$, $i \neq j$, such that pq is an edge of $\operatorname{conv}(R \cup S)$ then pq would also be an edge of $\operatorname{conv}(S \cup \{p, q\})$. This implies that either $D_p \cap D_q \neq \emptyset$ or that there are facets $f' \in D_p$, $f'' \in D_q$ such that f' and f'' share an edge. Both are impossible by Claim 3.2.8.

Next, we claim that $|\Gamma| = O(1)$: if $p \in R_i$ is adjacent to a point $q \in S$, then it follows that pq is also an edge of $\operatorname{conv}(S \cup \{p\})$, and hence D_p contains a facet incident to q. Since $|\bigcup_{p \in R_i} D_p| = O(1)$ and since each facet is incident to three points, the claim follows.

Now we compute $\operatorname{conv}(R_i \cup \Gamma)$ in $O(|R_i|)$ time as follows: let F_1 be the set of facets in $F[R \cup S]$ incident to R_i and let F_2 be the set of facets in $F[R_i \cup \Gamma]$ incident to R_i. We have $F_1 = F_2$. Clearly, $F_1 \subseteq F_2$ by the definition of Γ and since $R_i \cup \Gamma \subseteq R \cup S$. If there were a facet $f \in F_2 \setminus F_1$, the half-space spanned by f would contain only points in $(R \cup S) \setminus (R_i \cup \Gamma)$. However, this would mean that in $\operatorname{conv}(R \cup S)$ all the vertices of f are adjacent to a point in $(R \cup S) \setminus (R_i \cup \Gamma)$, contradicting the choice of Γ. The facets in F_1 can be extracted from $\operatorname{conv}(R \cup S)$ in time $O(|R_i \cup \Gamma|)$, and the convex hull of $R_i \cup \Gamma$ can be completed in the same time, since the remaining facets involve only points in Γ, which has constant size. Now $\operatorname{conv} R_i$ can be extracted from $\operatorname{conv}(R_i \cup \Gamma)$ in linear time, either by using `SplitHull` or by naively removing the points in Γ one by one. □

Algorithm 3.5 Splitting arbitrary colorings.

```
MultiSplit(conv $P$)

  1. For all colors i with $|C_i| \leq 2^{\sqrt{\log n}}$, find $\operatorname{conv} C_i$ directly. Let $K$ denote the
     remaining colors and $Q = \bigcup_{i \in K} C_i$. Use SplitHull to determine $\operatorname{conv} Q$.

  2. Use Lemma 3.2.6 with $\alpha = 2^{\sqrt{\log n}}$ to obtain $S, R \subseteq Q$, a partition of $R_1, \dots, R_\beta$
     of $R$, and their convex hulls.

  3. Call MultiSplit($\operatorname{conv}(Q \setminus (S \cup R))$) to find the hulls $\operatorname{conv}(C_i \cap (Q \setminus (S \cup R)))$.

  4. For $j = 1, \dots, \beta$, call MultiSplit($\operatorname{conv} R_j$) to find the hulls $\operatorname{conv}(C_i \cap R_j)$.

  5. For $i \in K$ do

     (a) For $j = 1, \dots, \beta$, merge $\operatorname{conv}(C_i \cap R_j)$ into $\operatorname{conv}(S)$. This yields $\operatorname{conv}(S \cup
         (C_i \cap R))$.
     (b) Use SplitHull to extract $\operatorname{conv}(C_i \cap (S \cup R))$.
     (c) Compute the union of $\operatorname{conv}(C_i \cap (S \cup R))$ and $\operatorname{conv}(C_i \cap (Q \setminus (S \cup R)))$
         to obtain $\operatorname{conv} C_i$.
```

This concludes the proof of Lemma 3.2.6. □

Now, the splitting is performed by the algorithm `MultiSplit`. Please refer to Algorithm 3.5. For the recursion to work, we need to avoid small color classes. Thus, the algorithm first computes the convex hull of every C_i with $|C_i| \leq 2^{\sqrt{\log n}}$ in time $O(|C_i| \log |C_i|)$ [23, Chapter 11]. Let K denote the remaining colors, and let $Q = \bigcup_{i \in K} C_i$, $n_1 = |Q|$ and $n_2 = n - n_1$. For Step 5a, we can use an algorithm to combine 3-polytopes separated by a plane [31, Chapter 9.3] to merge each $\operatorname{conv}(C_i \cap R_j)$ with $\operatorname{conv}(S)$. For $j \in \{1, \dots, \beta\}$, this takes time $O(1 + |C_i \cap R_j|)$, since all new edges are incident to constantly many points in S by properties 3 and 4 of Lemma 3.2.6 and since the conflict sets of the R_j do not interact. By Theorem 3.1.1, Step 5b takes expected time $O(|S| + |C_i \cap R|)$, and as Chazelle [45] showed, Step 5c needs time $O(|C_i|)$. Hence, the total expected time for Step 5 is $O(|K| \cdot |S| + \sum_{i \in K} |C_i|)$. Recall that $|C_i| > 2^{\sqrt{\log n}}$ for all $i \in K$. Hence, $|K| < n / 2^{\sqrt{\log n}}$ and $|K| \cdot |S| < n$. Therefore, the total running time of the algorithm is $O(n_2 \sqrt{\log n} + n)$, not counting the recursive calls. The first term represents the time for the convex hull computation in Step 1, and the second term counts the remaining steps.

We get the following recursion for the running time:

$$T(n) \leq T(|Q \setminus (S \cup R)|) + \sum_{j=1}^{\beta} T(|R_j|) + c(n_2\sqrt{\log n} + n),$$

for some constant $c > 0$. We know that $|R| \geq \alpha n_1$ and we also know that $\max_{1 \leq j \leq \beta} |R_j| \leq cn_1\sqrt{\log n}/2^{\sqrt{\log n}}$, where $\alpha \in (0, 1]$ and we reuse c (making it larger if necessary). A simple induction shows that $T(n) \ll n\sqrt{\log n}$.

Let us do the calculation. Recalling that $|Q| = n_1$ and plugging in the inductive hypothesis $T(m) \leq \gamma m\sqrt{\log m}$ for $m < n$ and some constant $\gamma > 0$, we get

$$\begin{aligned}
&T(n) \\
&\leq \gamma(n_1 - |R|)\sqrt{\log n} + \gamma|R|\sqrt{\log n + \log(c\sqrt{\log n}) - \sqrt{\log n}} + cn_2\sqrt{\log n} + cn \\
&\leq \gamma(n_1 - |R|)\sqrt{\log n} + \gamma|R|\sqrt{\log n - 0.5\sqrt{\log n}} + cn_2\sqrt{\log n} + cn,
\end{aligned}$$

for n large enough. Since $\sqrt{\log n - 0.5\sqrt{\log n}} \leq \sqrt{\log n} - 1/4$, it follows that

$$\begin{aligned}
T(n) &\leq \gamma(n_1 - |R|)\sqrt{\log n} + \gamma|R|\sqrt{\log n} - \gamma|R|/4 + cn_2\sqrt{\log n} + cn \\
&\leq \gamma n_1\sqrt{\log n} + 2cn_2\sqrt{\log n} + (c - \gamma\alpha/4)n_1,
\end{aligned}$$

which is bounded by $\gamma n\sqrt{\log n}$ for γ large enough.

3.3 Data structure version

One drawback of Theorem 3.1.1 is that it requires time linear in the size of the polytope $\mathcal{P}$. Therefore, if the blue set has cardinality smaller than $|\mathcal{P}|/\log|\mathcal{P}|$, we do not gain any advantage over traditional convex hull algorithms. To obtain a better running time, we change the model: given a point set U in gcp, we want to preprocess U such that the convex hull of any large enough subset $P \subseteq U$ can be found faster. Such a query is called a *convex hull query*. In the following, we will have $u = |U|$ and $n = |P|$. Our description starts with a simple structure building directly on the techniques of Section 3.2.2. It handles convex hull queries in time $O(n\sqrt{\log u}\log\log u)$. Then, we bootstrap this structure so that convex hull queries take $O(n(\log\log u)^2)$ time. We call our data structure the *scaffold tree*. Using scaffold trees, we can also improve Theorem 3.2.5 to $O(n(\log\log n)^2)$ running time. Later, in Theorem 5.2.6, we will prove that for the special case of Delaunay triangulations a query time of $O(n\log\log u)$ is possible.

3.3.1 The basic structure

The basic data structure is described in the following theorem.

Theorem 3.3.1. *Let $U \subseteq \mathbb{R}^3$ be a u-point set in gcp. In $O(u \log u)$ time, we can construct a data structure of size $O(u\sqrt{\log u})$ such that for any n-point set $P \subseteq U$ we can compute* $\operatorname{conv} P$ *in time $O(n\sqrt{\log u}\log\log u)$. If* $\operatorname{conv} U$ *is known, the preprocessing time is $O(u\sqrt{\log u})$.*

Proof. We describe the preprocessing phase. If necessary, we construct $\operatorname{conv} U$ in time $O(u \log u)$. The scaffold tree is computed through the recursive procedure `BuildTree`(U). Please consider Algorithm 3.6. By Lemma 3.2.6, the sizes of the

---

Algorithm 3.6 Building the basic scaffold tree.

```
BuildTree($U$)

 1. If $|U| = O(1)$, store $U$ and return, otherwise, let $U_1 = U$ and $i = 1$
 2. While $|U_i| > u/2^{\sqrt{\log u}}$.
    (a) Apply Lemma 3.2.6 to $U_i$ with $\alpha = 2^{\sqrt{\log |U_i|}}$ to obtain subsets $S_i, R_i \subseteq U_i$,
        as well as a partition $R_i^{(1)}, \dots, R_i^{(\beta)}$ of $R_i$, and the hulls $\operatorname{conv} S_i$, $\operatorname{conv} R_i^{(j)}$
        described in the lemma.
    (b) Call BuildTree$\left(R_i^{(j)}\right)$ for $j = 1, \dots, \beta$.
    (c) Let $U_{i+1} = U_i \setminus (S_i \cup R_i)$. Use SplitHull to compute $\operatorname{conv} U_{i+1}$, and
        increment $i$.
 3. Let $\ell = i$ and call BuildTree$(U_\ell)$.
```

---

sets U_i decrease geometrically, so $\ell = O(\sqrt{\log u})$ and

$$\sum_{i=1}^{\ell} |S_i| = O\left(2^{\sqrt{\log u}}\sqrt{\log u}\right). \tag{3.5}$$

By Lemma 3.2.6 and Theorem 3.1.1, the total time for Steps 2a and 2c is $O(u)$. Since the sets for the recursive calls in Steps 2b and 3 have size $O(u\sqrt{\log u}/2^{\sqrt{\log u}})$, the expected running time $T(u)$ of `BuildTree` obeys the recursion

$$T(u) = O(u) + \sum_i T(m_i),$$

where the m_i are such that

$$\sum_i m_i < u \text{ and } \max_i m_i = O(u\sqrt{\log u}/2^{\sqrt{\log u}}).$$

Thus, the work in each level of the recursion is $O(u)$, and the number of levels $L(u)$ has

$$L(u) \le 1 + L\big(u\sigma\sqrt{\log u}/2^{\sqrt{\log u}}\big),$$

for some constant σ. To see that $L(u) \ll \sqrt{\log u}$, we use induction to prove

$$\begin{aligned} L(u) &\le 1 + c\sqrt{\log u + \log\sigma + (1/2)\log\log u - \sqrt{\log u}} \\ &\le 1 + c\sqrt{\log u}\sqrt{1 - 1/(2\sqrt{\log u})} \le c\sqrt{\log u}, \end{aligned}$$

since for c large enough $c\sqrt{\log u}\sqrt{1 - 1/2(\sqrt{\log u})} \le c\sqrt{\log u} - 1$. Therefore, we get $T(u) \ll u\sqrt{\log u}$. Queries are answered by a recursive procedure called `Query`(P). Refer to Algorithm 3.7. Step 1 takes $O\left(n\sqrt{\log u}\right)$ time [23, 31, 122, 128]. With

Algorithm 3.7 Querying the simple scaffold tree.

```
Query(P)

1. If $n \le 2^{\sqrt{\log u}}\sqrt{\log u}$, use a traditional algorithm to find conv $P$ and return.
2. For $i = 1, \ldots, \ell - 1$
    (a) Let $P_i = P \cap R_i$ and determine the intersections $P_{ij}$ of $P_i$ with the sets $R_i^{(j)}$.
    (b) For all non-empty $P_{ij}$, call Query($P_{ij}$) to compute conv $P_{ij}$.
    (c) Merge conv $P_{ij}$ into conv $S_i$.
    (d) Use SplitHull to extract conv $P_i$ from the convex hull conv $(P_i \cup S_i)$.
3. Let $P_\ell = U_\ell \cap P$. If $P_\ell \ne \emptyset$, call Query($P_\ell$) for conv $P_\ell$.
4. Compute conv $P$ as the union of conv $P_1$, ..., conv $P_\ell$.
```

an appropriate pointer structure that provides links for the points in U to the corresponding subsets (as in the pointer-based implementation of van Emde Boas trees [114]), the total time for Step 2a is $O(n)$. The next claim handles Step 2c.

Claim 3.3.2. *Step 2c takes $O(|P_i|)$ time.*

Proof. Fix j with $P_{ij} \neq \emptyset$. We show how to insert P_{ij} into $\operatorname{conv} S_i$ in time $O(|P_{ij}|)$. This implies the claim, since by the definition of the $R_i^{(j)}$ there can be no edge between two points $p \in P_{ij}$ and $q \in P_{ij'}$, for $j \neq j'$. Furthermore, the only facets in $\operatorname{conv} S_i$ that are destroyed are facets in $\Delta = \bigcup_{p \in P_{ij}} D_p$ by definition of D_p. By Lemma 3.2.6(2,3), the size of Δ is constant, because all the D_p have constant size, form connected components in the dual of $\operatorname{conv} S_i$ (a 3-regular graph), and have one facet in common. Thus, all we need to do is insert the constantly many points in S_i incident to a facet in Δ into $\operatorname{conv} P_{ij}$, which takes time $O(|P_{ij}|)$, as claimed. □

By Theorem 3.1.1, Step 2d needs $O(|P_i|+|S_i|)$ time. Using an algorithm for merging convex hulls [45], Step 4 can be done in time $O(n \log \ell) = O(n \log\log u)$. Thus, the total time for Steps 2 to 4 is $O(n \log\log u + \sum_{i=1}^{\ell} |S_i|) = O(n \log\log u)$, by (3.5) and Step 1. Since there are $O(\sqrt{\log u})$ levels and since the computation in Step 1 is executed only once for each point in P, the result follows. □

3.3.2 Bootstrapping the tree construction

We now describe the bootstrapping step. The main idea is to increase the degree in each node of the scaffold tree and to use a more basic tree to combine the results from the recursive calls quickly. However, if we are not careful, we could lose a constant factor in each bootstrapping step, which would not give the desired running time. To avoid this, we need the following result, which uses the dependent sampling technique which we will encounter again in Chapter 5.

Theorem 3.3.3. *Let $U \subseteq \mathbb{R}^3$ be in gcp, and let $S = \bigcup_{i=1}^{k} S_i \subseteq U$ with $|S| = m$, such that $|S_i| \leq c$, for some constant c, and such that the subgraphs $\operatorname{conv} U|_{S_i}$ are connected and available. Furthermore, let $S' \subseteq S$ be such that S' contains exactly one point from each S_i, chosen uniformly at random, and suppose $\operatorname{conv} S'$ is available, and that we have a van Emde Boas structure for the neighbors of each vertex in $\operatorname{conv} U$. Then we can find $\operatorname{conv} S$ in expected time $O(m \log\log u)$.*

Proof. To obtain $\operatorname{conv} S$ from $\operatorname{conv} S'$, we walk along the edges of the subgraphs $\operatorname{conv} U|_{S_i}$ and insert the points one by one. More precisely, we proceed as follows: let Q be a queue which we initialize with the points from S'. While Q is not empty, let p be the next point in Q. Using the van Emde Boas structure for p, sort the neighbors of p in the $\operatorname{conv} U|_{S_i}$ according to clockwise order. Then insert the neighbors in that order, using the edges of the $\operatorname{conv} U|_{S_i}$ for conflict location. Add all the neighbors that have not been encountered before to Q. The time taken is $O(m \log\log u)$ for the sorting and the traversal of the queue, plus the number of facets of the convex hull that were created (and possibly destroyed) during the construction.

Let f be a facet with conflict set B_f. The facet f is created only if $S' \cap B_f = \emptyset$. The probability of this event is at most

$$\prod_{i=1}^{k}\left(1 - \frac{|S_i \cap B_f|}{|S_i|}\right) \leq \exp\left(-\sum_{i=1}^{k}\frac{|S_i \cap B_f|}{c}\right) = \exp(-|B_f|/c).$$

Now, since it is well known [59] that there are at most $O(ms^2)$ facets with conflict size s, the expected number of created facets is $O\left(m\sum_{s=0}^{\infty} s^2/e^{s/c}\right) = O(m)$, and the result follows. □

Theorem 3.3.4. *Let $k > 1$ and $U \subseteq \mathbb{R}^3$ be a u-point set in gcp, and let $\operatorname{conv} U$ be given. Set $l_k = (\log u)^{1/k}$. There is a constant β with the following property: if $D_k(U)$ is a data structure for convex hull queries with preprocessing time $P_k(u) \ll ukl_k$ and query time $Q_k(n,u) \leq \beta nkl_k \log\log u$, then there is a data structure $D_{k+1}(U)$ with preprocessing time $P_{k+1}(u) \ll u(k+1)l_{k+1}$ and query time $Q_{k+1}(n,u) \leq \beta n(k+1)l_{k+1}\log\log u$. The constants in the asymptotic bounds do not depend on k.*

Proof. Since the function $x \mapsto x\log^{1/x} u$ reaches its minimum for $x = \ln 2 \log\log u$, we may assume that $k < 0.7\log\log u$, because otherwise the theorem holds by assumption. The preprocessing is very similar to Algorithm 3.6 with a few changes: (i) we iterate the loop in Step 2 while $|U_i| > u/2^{l_{k+1}^k}$; (ii) we apply Lemma 3.2.6 with $\alpha = 2^{(\log|U_i|)^{k/(k+1)}}$; and (iii) for each sample S_i we compute a data structure $D_k(S_i)$ for convex hull queries, which exists by assumption, for details see Algorithm 3.8.

Since the sizes of the U_i decrease geometrically, we have $\ell = O\left(l_{k+1}^k\right)$ and the total time for Step 2a is $O(u)$. Since $k \in \{2, \dots, 0.7\log\log u - 1\}$, we have

$$2^{0.7} \leq l_{k+1} \leq l_k \leq \sqrt{\log u}, \tag{3.6}$$

and therefore the total time to construct the data structures $D_k(S_i)$ in Step 2b is proportional to

$$\begin{aligned} l_{k+1}^k P_k\big(2^{l_{k+1}^k}\big) \ll l_{k+1}^k k l_{k+1} 2^{l_{k+1}^k} &\ll 2^{\log u/l_{k+1}} \log u \log\log u \\ &\leq u^{1/2^{0.7}} \log u \log\log u = o(u). \end{aligned}$$

Similarly, the total time for the construction of the vEB trees in Step 2c can be bounded by $O\big(l_{k+1}^k 2^{l_{k+1}^k}\log\log u\big) = o(u)$. Since the sets $R_i^{(j)}$ all have size at most $O\big(ul_{k+1}^k/2^{l_{k+1}^k}\big)$, the number of levels $L_{k+1}(u)$ has

$$L_{k+1}(u) \leq 1 + L_{k+1}\big(u\sigma(\log u)/2^{l_{k+1}^k}\big),$$

for some constant σ. To prove that $L_{k+1}(u) \ll (k+1)l_{k+1}$, we use induction to get

$$L_{k+1}(u) \leq 1 + c(k+1)\left(\log u + \log\sigma + \log\log u - l_{k+1}^k\right)^{1/(k+1)}.$$

Algorithm 3.8 Bootstrapping the scaffold tree.

```
BuildTree_{k+1}(U)

  1. Let $U_1 = U$ and $i = 1$
  2. While $|U_i| > u/2^{l_{k+1}^k}$.
     (a) Apply Lemma 3.2.6 to $U_i$, where we set $\alpha = 2^{(\log |U_i|)^{k/(k+1)}}$. This yields
         subsets $S_i, R_i \subseteq U_i$, a partition $R_i^{(1)}, \dots, R_i^{(\beta)}$ of $R_i$, and the convex hulls
         $\operatorname{conv} S_i$, $\operatorname{conv} R_i^{(j)}$ with the properties of Lemma 3.2.6.
     (b) Execute BuildTree_{k+1}$\left(R_i^{(j)}\right)$ for $j = 1, \dots, \beta$ and compute a data structure
         $D_k(S_i)$ for $S_i$.
     (c) Let $U_{i+1} = U_i \setminus (S_i \cup R_i)$. Compute $\operatorname{conv} U_{i+1}$ using SplitHull, and
         for each vertex, create a van Emde Boas structure for its neighbors.
         Increment $i$.
  3. Let $\ell = i$ and call BuildTree_{k+1}$(U_\ell)$.
```

By (3.6), $l_{k+1}^k \geq \sqrt{\log u}$, so $\log \sigma + \log \log u - l_{k+1}^k < -0.9 l_{k+1}^k$, for u large enough, and

$$L_{k+1}(u) \leq 1 + c(k+1)l_{k+1}\left(1 - \frac{0.9}{l_{k+1}}\right)^{1/(k+1)}.$$

Now, since

$$\left(1 - \frac{0.9}{l_{k+1}}\right)^{1/(k+1)} \leq \exp\left(-\frac{0.9}{(k+1)l_{k+1}}\right) \leq 1 - \frac{1}{3(k+1)l_{k+1}}, \tag{3.7}$$

we get

$$L_{k+1}(u) \leq 1 + c(k+1)l_{k+1} - c/3 \leq c(k+1)l_{k+1},$$

for c large enough. The work at each level is $O(u)$, therefore the total preprocessing time is $P_{k+1}(u) \ll u(k+1)l_{k+1}$. Queries are answered by `Query`$_{k+1}$, please look at Algorithm 3.9.

In the following, let c denote a large enough constant. As before, Step 1a takes time

$$T_{1a} \leq cn, \tag{3.8}$$

using an appropriate pointer structure. In Step 1b, let n_1 denote the total number of points for which we compute the convex hull directly, and let $n_2 = n - n_1$. Then this step takes time

$$T_{1b} \leq cn_1 \log \log u + cn_1 \log \beta + Q_{k+1}\left(n_2, cul_{k+1}^k / 2^{l_{k+1}^k}\right), \tag{3.9}$$

Algorithm 3.9 Querying the bootstrapped scaffold tree.

```
Query$_{k+1}(P)$

  1. For $i = 1, \dots, \ell - 1$
     (a) Let $P_i = P \cap R_i$ and determine the intersections $P_{ij}$ of $P_i$ with the sets $R_i^{(j)}$.
     (b) For each non-empty $P_{ij}$, if $|P_{ij}| \le \beta k l_{k+1}$, compute $\operatorname{conv} P_{ij}$ directly, otherwise call Query$_{k+1}(P_{ij})$.
     (c) For each nonempty $P_{ij}$, determine the set $S_{ij}$ of points adjacent to a conflict facet of $P_{ij}$. Compute $\operatorname{conv}(P_{ij} \cup S_{ij})$.
     (d) Let $S'_i$ be a set that contains one random point from each $S_{ij}$. Use $D_k(S_i)$ to find $\operatorname{conv} S'_i$.
     (e) Use Theorem 3.3.3 to compute $\operatorname{conv}(P_i \cup \bigcup_j S_{ij})$.
     (f) Use SplitHull to extract $\operatorname{conv} P_i$ from $\operatorname{conv}(P_i \cup \bigcup_j S_{ij})$.
  2. Recursively compute $\operatorname{conv} P_\ell$, where $P_\ell = U_\ell \cap P$.
  3. Compute $\operatorname{conv} P$ as the union of $\operatorname{conv} P_1, \dots, \operatorname{conv} P_\ell$.
```

assuming that Q_{k+1} is linear in the first and monotonic in the second component (which holds by induction on the second component). Since the conflict size of each P_{ij} is constant, Step 1c takes time

$$T_{1c} \le cn, \tag{3.10}$$

if we just insert the points in each S_{ij} into $\operatorname{conv} P_{ij}$ one by one. Furthermore, since we select one point per conflict set, the total size of the sets S'_i in Step 1d is at most $n_1 + n_2/(\beta k l_{k+1})$, so computing the convex hulls $\operatorname{conv} S'_1, \dots, \operatorname{conv} S'_\ell$ takes time

$$\begin{aligned} T_{1d} \le Q_k\left(n_1 + \frac{n_2}{\beta k l_{k+1}}, 2^{l^k_{k+1}}\right) &\le \beta k \left(n_1 + \frac{n_2}{\beta k l_{k+1}}\right) l_{k+1} \log\log u \\ &= (\beta n_1 k l_{k+1} + n_2) \log\log u. \end{aligned} \tag{3.11}$$

In Step 1e, we use Theorem 3.3.3 to find $\operatorname{conv}\big(\bigcup_j S_{ij}\big)$, from which we can find $\operatorname{conv}\big(P_i \cup \bigcup_j S_{ij}\big)$ easily by the independence of the $R_i^{(j)}$. By Theorems 3.3.3 and 3.1.1, Steps 1e and 1f take time

$$T_{1e,1f} = O(n \log\log u + n + \sum_i |S'_i| \log\log u) \le cn \log\log u. \tag{3.12}$$

Step 2 is already accounted for by T_{1b}. Finally, Step 3 requires

$$T_3 \leq cn \log\log u \tag{3.13}$$

steps. By summing (3.8–3.13) we get the following recurrence for $Q_{k+1}(n, u)$:

$$Q_{k+1}(n,u) \leq 5cn\log\log u + cn_1 \log\beta + \\ \beta n_1 k l_{k+1} \log\log u + Q_{k+1}\left(n_2, cu l_{k+1}^k / 2^{l_{k+1}^k}\right). \tag{3.14}$$

By induction, we get

$$Q_{k+1}\left(n_2, cu\frac{l_{k+1}^k}{2^{l_{k+1}^k}}\right) \leq \beta n_2(k+1)\left(\log u + \frac{k\log(c\log u)}{k+1} - l_{k+1}^k\right)^{1/(k+1)} \log\log u.$$

By (3.6), we have $l_{k+1}^k \geq \sqrt{\log u}$, and hence $\frac{k}{k+1}\log(c\log u) - l_{k+1}^k \leq -0.9 l_{k+1}^k$, for u large enough. Using (3.7), it follows that

$$\begin{aligned} Q_{k+1}\left(n_2, cu\frac{l_{k+1}^k}{2^{l_{k+1}^k}}\right) &\leq \beta n_2(k+1) l_{k+1}\left(1 - \frac{0.9}{l_{k+1}}\right)^{1/(k+1)} \log\log u \\ &\leq \beta n_2(k+1)\left(l_{k+1} - \frac{1}{3(k+1)}\right)\log\log u \\ &= \beta n_2(k+1) l_{k+1} \log\log u - \frac{\beta}{3} n_2 \log\log u. \end{aligned}$$

Plugging this bound into (3.14), and choosing β large enough, we conclude that $Q_{k+1}(n,u) \leq \beta n(k+1) l_{k+1} \log\log u$, as claimed. □

Now we can finally prove the main result of this section.

Corollary 3.3.5. *Let $U \subseteq \mathbb{R}^3$ be a u-point set in gcp. In $O(u\log u)$ time, we can construct a data structure for convex hull queries with expected query time $O(n(\log\log u)^2)$. The space needed is $O(u(\log\log u)^2)$, and if* $\operatorname{conv} U$ *is available, the preprocessing time reduces to $O(u(\log\log u)^2)$.*

Proof. For $k = \frac{1}{2}\log\log u$, we have $(\log u)^{1/k} \ll 1$, and the result follows from Theorem 3.3.1 and a repeated application of Theorem 3.3.4. □

This immediately yields the improvement over Theorem 3.2.5 that we promised at the beginning.

Corollary 3.3.6. *Let $P \subseteq \mathbb{R}^3$ be an n-point set in gcp and $c\colon P \to \{1, \dots, \chi\}$ a coloring of P. Given* $\operatorname{conv} P$*, we can find* $\operatorname{conv} c^{-1}(1), \dots, \operatorname{conv} c^{-1}(\chi)$ *in expected time $O(n(\log\log n)^2)$.*

Proof. Given $\operatorname{conv} P$, build the structure from Corollary 3.3.5 in $O(n(\log\log n)^2)$ time. Then perform a query for each color class. Since the color classes are disjoint, the total time is $O(n(\log\log n)^2)$. □

3.4 Points in halfspaces

We continue the investigation of the data structure problem by considering a special case: colorings induced by halfspaces. Specifically, we would like to preprocess a point set in gcp to report the convex hull of all the points inside a query halfspace. In this case, the convex hull can be found in linear time, as long as the query set contains $\Omega(\log n)$ points. We base our approach on a data structure by Chan [36] that uses *filtering search* [43]: first, it obtains a superset of the result with comparable size (the *candidate set*), and then examines each point individually to find the result. By storing not only the candidate sets, but also their convex hulls, we obtain a data structure that reports the convex hull of the points in a query halfspace by using `SplitHull`. We also show how to improve the preprocessing time over the straightforward $O(n \log^2 n)$.

Theorem 3.4.1. *Let $P \subseteq \mathbb{R}^3$ be an n-point set in gcp. In $O(n \log n)$ time we can build a randomized data structure of $O(n \log n)$ size to answer queries of the following kind: given an oriented plane h, compute the convex hull of $P \cap h^+$, where h^+ denotes the left halfspace of h. The expected query time is $O(\log n + k)$, where $k = |P \cap h^+|$ denotes the output size.*

The main obstacle in improving the preprocessing time is this: given a sample $S \subseteq P$, compute the convex hulls of the conflict sets B_f for $f \in F[S]$. In Sections 3.2 and 3.3, we modified the conflict sets to obtain a simple algorithm for this problem. This is no longer possible, and we need a more sophisticated approach. Given a plane h, let $G(h)$ denote the induced subgraph of $\operatorname{conv} P$ with vertex set $P \cap h^+$ (ie, $G(h)$ has vertex set $P \cap h^+$ and contains all edges of $\operatorname{conv} P$ with both endpoints in h^+). Here are some simple facts about $G(h)$ (eg, [45]); see Figure 3.8a.

Lemma 3.4.2. *Let E be the set of edges in $G(h)$ incident to a facet of* $\operatorname{conv} P$ *that intersects h. There exists a closed walk*[10] *L along the edges in E such that L separates $G(h)$ from the rest of* $\operatorname{conv} P$*. Every edge $e \in E$ occurs in L once or twice, depending on whether e is incident to one or two such facets. It follows that $G(h)$ is connected. Given $G(h)$, L can be found in time $O(|V[G(h)]|)$.*

Proof. Consider the intersection $\mathcal{A}$ of h and $\operatorname{conv} P$ (interpreted as a subset of $\mathbb{R}^3$). The set $\mathcal{A}$ is a two-dimensional convex polygon whose edges correspond to the facets of $\operatorname{conv} P$ that intersect h. Let $F = f_1, f_2, \dots, f_k$ be those facets in counterclockwise order along $\mathcal{A}$, and let $F' = f'_1, f'_2, \dots, f'_k \subseteq F$ be the subsequence of facets that are incident to an edge in E. Since consecutive facets in F' share an incident vertex in $P \cap h^+$, the sequence F' induces a closed walk L along the edges in E. Every path

[10]In our terminology, a *walk* is an arbitrary sequence of adjacent vertices, whereas a *path* consists of distinct vertices (except possibly the first and the last).

from a point in $P \cap h^+$ to a point in $P \cap h^-$ has to cross a point incident to a facet in F'. Hence, L separates $G(h)$ from the rest of $\operatorname{conv} P$. Furthermore, every edge in E appears once in L for each incident facet in F'. Finally, since consecutive edges in L are consecutive in the cyclic order of edges around their common endpoint in $G(h)$, L can be computed in time linear in the size of $G(h)$. $\square$

The walk L is called the *lace* of $G(h)$; see Figure 3.8a. Knowing $G(h)$ is enough to compute $\operatorname{conv}(P \cap h^+)$ quickly.

Corollary 3.4.3. *Given* $\operatorname{conv} P$ *and* $G(h)$, *we can compute* $\operatorname{conv}(P \cap h^+)$ *in time* $O(|P \cap h^+|)$.

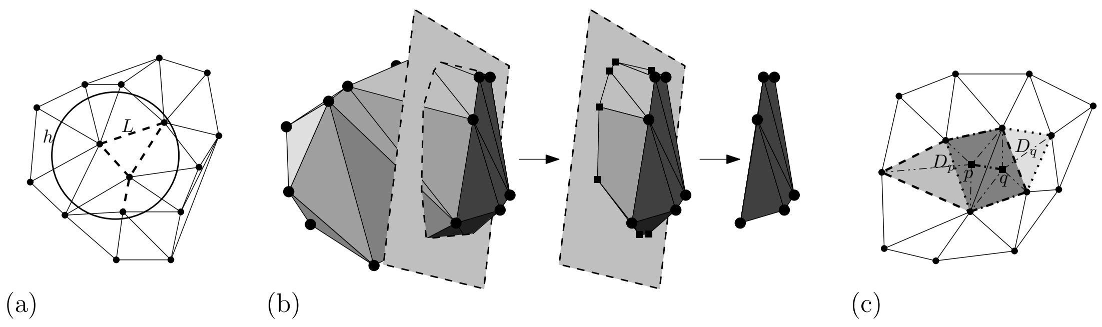


Figure 3.8: The halfspace range reporting algorithm: (a) A lace: h^+ corresponds to the inside of the circle. The lace L is shown as a dashed line. (b) The three stages of Corollary 3.4.3: Given $G(h)$, find an intermediate polytope that contains the result, and split it. (c) Finding the conflict facets for an edge. $D_{\{p,q\}}$ is darkest, while D_p, D_q are lighter. D_p is bounded by dashed line segments, D_q by dotted line segments.

Proof. The idea is to find an intermediate polytope $\mathcal{P}$ of complexity $O(|P \cap h^+|)$ whose vertices contain $P \cap h^+$. This is done by computing (part of) the intersection of $\operatorname{conv} P$ with h^+ and adding a few edges to ensure general position; see Figure 3.8b. Using `SplitHull`, we extract $\operatorname{conv}(P \cap h^+)$ from $\mathcal{P}$ in the desired time.

Let L be the lace of $G(h)$. By Lemma 3.4.2, L can be found in time $O(|P \cap h^+|)$ from $G(h)$. Let $F = f_1, \ldots, f_k$ be the sequence of facets in $F[P]$ that are incident to L and intersect h, where the ordering is according to L. The sequence F induces in the plane h a sequence E of line segments whose endpoints are in convex position. As the order of E corresponds to the convex hull order, we can compute the convex hull $\mathcal{C}$ of E in linear time. Let $V[\mathcal{C}]$ and $E[\mathcal{C}]$ denote the vertices and edges of $\mathcal{C}$.

We are now ready to construct the convex polytope $\mathcal{P}$. The set of $\mathcal{P}$'s facets consists of three disjoint parts, F_1,F_2, and F_3: (i) F_1 contains the facets of $G(h)$;

(ii) for each line segment $e \in E$, F_2 contains a quadrilateral facet spanned by e and its corresponding edge $\tilde{e}$ in L.[11] Furthermore, for each $e \in E[\mathcal{C}] \setminus E$, F_2 contains a triangular facet f_e spanned by e and the point in $P \cap h^+$ incident to the edges whose intersections with h determine e; (iii) let Z be the unbounded prism with base $\mathcal{C}$ that extends into h^-. Pick a point $q \in Z \cap \operatorname{conv} P$ infinitesimally close to h. F_3 contains all facets spanned by q and an edge in $E[\mathcal{C}]$. It is easily seen that the facets in $F_1 \cup F_2 \cup F_3$ are in convex position and bound a convex polytope $\mathcal{P}$ with $O(|P \cap h^+|)$ vertices. Since all the facets of $\mathcal{P}$ have bounded complexity, and since all vertices in $V[\mathcal{C}]$ have bounded degree, we can perform a local perturbation of $V[\mathcal{C}]$ to obtain a polytope $\mathcal{P}'$ in general position. Now we compute $\operatorname{conv}(P \cap h^+)$ in time $O(|V[\mathcal{P}']|) = O(|P \cap h^+|)$ using `SplitHull`. □

For Corollary 3.4.3, we need to compute all the graphs $G(h_f)$ for $f \in F[S]$ (recall that h_f denotes the plane supporting f in $\operatorname{conv} S$).

Lemma 3.4.4. *Let $S \subseteq P$ be a random subset. Then the graphs $G(h_f)$ for $f \in F[S]$ can be computed in $O(n)$ expected time.*

Proof. By Lemma 2.2.4 the total size of the sets $P \cap h_f^+$ and hence the total complexity of the graphs $G(h_f)$ is $O(n)$. Let $e = (p, q) \in E[P]$, and let $D_e = D_p \cap D_q$ be the facets in conflict with both p and q. Note that $e \in G(h_f)$ precisely if $f \in D_e$. We will compute the sets D_e for $e \in E[P]$ and then use them to construct the graphs $G(h_f)$. Let T_e denote the graph on vertex set D_e where two vertices f_1, f_2 are adjacent if f_1, f_2 share an edge in $\operatorname{conv} S$ that is destroyed in $\operatorname{conv}(S \cup \{p, q\})$. Since T_e is connected[12], it suffices to compute one facet $f_e \in D_e$ (if it exists). The remaining facets can be found by traversing T_e.

We extend `SubsetConflictWalk` to find conflict facets of edges by changing Step 2d as follows: when considering a neighbor $q \in \Gamma_P(p)$, we not only compute the conflict facet f_q, but also a conflict facet f_e for the edge $e = \{p, q\}$, if it exists. To do this, let Γ_p denote the simple polygon in $\operatorname{conv} S$ that bounds the conflict region of p. The facet $\tilde{f}_q \in F[S \cup p]$ is adjacent to an edge e_q on Γ_p, and q conflicts with at least one facet in $\operatorname{conv} S$ incident to e_q. Let $f_1, f_2 \in F[S]$ be the facets incident to e_q, where f_1 conflicts with p while f_2 does not. Now, if q conflicts with f_1, we set $f_q = f_e = f_1$, otherwise, we set $f_q = f_2$ and $f_e = \perp$.[13] This takes constant time, and therefore the running time of the algorithm remains linear, as in the proof of Lemma 3.2.3.

To prove correctness, we claim that if $D_e \neq \emptyset$, then $f_1 \in D_e$. Indeed, let T be the graph on vertex set $D_p \cup D_q$, where two vertices g_1, g_2 of T are adjacent if

[11] That is, $\tilde{e}$ is the edge incident to the facet whose intersections with h create e.

[12] We define the empty graph to be connected.

[13] As is often done in the study of programming languages, we use $\perp$ as a symbol for an undefined value.

g_1, g_2 share an edge in $E[S]$ that is destroyed in $\operatorname{conv}(S \cup \{p, q\})$. We have that T_e is a subgraph of T and that T is a tree (by convex position). Observe that e_q corresponds to the edge $e_q^* = \{f_1, f_2\}$ of T. Let T_1 be the connected component of $T \setminus e_q^*$, with $f_1 \in T_1$. Note that $D_p \subseteq V[T_1]$. Furthermore, at least one of f_1, f_2 is in conflict with q, hence $D_q \cap \{f_1, f_2\} \neq \emptyset$. Since the induced subgraph of T on vertex set D_q is connected, it follows that if $V[T_1] \cap D_q \neq \emptyset$, then D_q contains f_1, and hence $f_1 \in D_p \cap D_q = D_e$, as desired.

Using the sets D_e, we can now compute a DCEL representation of the graphs $G(h_f)$ in $O(n)$ time through careful pointer manipulation (Algorithm 3.10). □

Algorithm 3.10 Computing the subgraphs.

```
ComputeSubgraphs

1. For every $e \in E[P]$, if $f_e \neq \bot$, use $f_e$ to compute $D_e$. For each $f \in D_e$ create
   records for the two half edges corresponding to $e$ in $G(h_f)$.

2. For every point $p \in P$, use $f_p$ to find $D_p$. For each $f \in D_p$, create a record $p_f$
   corresponding to $p$ in $G(h_f)$. Every facet in $D_p$ has a pointer p which we set
   to $p_f$. For each incident edge $e$ of $p$ in cyclic order, iterate through all facets
   $f \in D_e$. Use the pointer p of $f$ to find the record $p_f$ corresponding to $p$ in
   $G(h_f)$ and add the appropriate half edge to the edge list of $p_f$.
```

Proof of Theorem 3.4.1. We rely on a variant of Chan's data structure [36] due to Ramos [130]. The candidate sets are the conflict sets of an appropriate gradation of P. By Corollary 3.4.3 and Lemma 3.4.4, we can find their convex hulls in time $O(n \log n)$. To process a query, we extend the original query algorithm to use `SplitHull` on the candidate set after coloring the points in h^+ blue.

The details are as follows: take a *gradation* $\emptyset = P_{-1} \subseteq P_0 \subseteq \cdots \subseteq P_{\log n} = P$, where P_{i-1} is derived from P_i by sampling every point with probability 1/2. We compute the convex hulls $\operatorname{conv} P_i$ in time $O(n \log n)$. Using Lemma 3.4.4 and Corollary 3.4.3, we then find the convex hulls $\operatorname{conv} B_f$ for all the conflict sets B_f, $f \in F[P_i]$, $i = 0, \dots, \log n$. Since this takes $O(n)$ time for each i, the total time is $O(n \log n)$. Now we switch into dual space. For this, we use duality with respect to the unit paraboloid which turns upper convex hulls into upper envelopes and lower convex hulls into lower envelopes [122, Chapter 2.4.1]. We compute two data structures, one for the upper envelope and one for the lower envelope, focusing the discussion on the lower envelope. For each $i = 0, \dots, \log n$, we find the set of planes H_i dual to P_i and a canonical triangulation T_i of the lower envelope of H_i (this takes linear time since we know $\operatorname{conv} P_i$). Then we construct a point location structure for the xy-projection of T_i. Every facet Δ of T_i is incident to at most three points of the lower envelope of $\mathcal{H}_i$, corresponding to at most three facets f_1, f_2, f_3 of $\operatorname{conv} P_i$. Let $B_\Delta = B_{f_1} \cup B_{f_2} \cup B_{f_3}$. We compute $\operatorname{conv} B_\Delta$ in linear time [45] and store it with Δ. By the properties of canonical triangulations and the arguments given by Chan [36], the preprocessing phase takes expected time $O(n \log n)$ and uses expected space $O(n \log n)$. Then we repeat the process to obtain two independent data structures D_1, D_2.

Now suppose that we are given a query plane h. We need to find all the planes in H below h^*, the point dual to h. Let ℓ be the vertical line through h^*. Perform the following procedure simultaneously on D_1 and D_2, until one of them yields the answer: For $i = \log(n/\log n), \log(n/\log n) - 1, \dots, 0$, locate the facet Δ_i of T_i intersected by ℓ in $O(\log n)$ time with the point location structure. Stop when the dual point h^* lies below the lower envelope of H_i. Now find the planes in H below h^* by inspecting the conflict set B_{Δ_i}, and use `SplitHull` to compute $\operatorname{conv}(P \cap h^+)$ in $O(|B_{\Delta_i}|)$ time. As was argued by Ramos [130, Section 2.2.1] such a query takes expected time $O(\log n + |P \cap h^+|)$, as claimed.

For completeness, we repeat the calculation here. Let $\mathcal{E}_i$ denote the event that i is the largest index for which h^* lies below the lower envelope of h_i in either D_1 or D_2. The expected running time is

$$\sum_{i=0}^{\log n - \log\log n} \left((\log n - \log\log n - i)\log n + \mathbf{E}\left[|B_{\Delta_i}| \mid \mathcal{E}_i\right]\right) \cdot \Pr[\mathcal{E}_i]. \tag{3.15}$$

Let $k = |P \cap h^+|$. Note that $\mathbf{E}\left[|B_{\Delta_i}| \mid \mathcal{E}_i\right] = O(k + n/2^i)$, since by Lemma 2.2.6, we have $\mathbf{E}\left[|B_{\Delta_i}|\right] = O(n/2^i)$, and the random choices for the points in $P \setminus (P \cap h^+)$ are independent of $\mathcal{E}_i$. Thus,

$$\begin{aligned}
(3.15) &\ll \sum_{i=0}^{\log(n/\log n)} \left(\left(\log\left(\frac{n}{\log n}\right) - i\right)\log n + k + \frac{n}{2^i}\right)\Pr[\mathcal{E}_i] \\
&\ll k + \sum_{i=\log(n/k)}^{\log(n/\log n)} \left(\left(\log\left(\frac{n}{\log n}\right) - i\right)\log n + \frac{n}{2^i}\right)\Pr[\mathcal{E}_i] \\
&\quad + \sum_{i=0}^{\log(n/k)-1} \left(\left(\log\left(\frac{n}{\log n}\right) - i\right)\log n + \frac{n}{2^i}\right)\Pr[\mathcal{E}_i].
\end{aligned}$$

If $k < \log n$, the first sum is zero. Otherwise, we get

$$\sum_{i=\log(n/k)}^{\log(n/\log n)} \left(\left(\log\left(\frac{n}{\log n}\right) - i\right)\log n + \frac{n}{2^i}\right)\Pr[\mathcal{E}_i] \\
\leq (\log n)\log\left(\frac{k}{\log n}\right) + \sum_{i \geq \log(n/k)} \frac{n}{2^i} \ll k.$$

(To see that $\log n \cdot \log(k/\log n) \leq k$, write $k = \alpha \log n$). To bound the second sum, we observe that $\Pr[\mathcal{E}_i] \leq (k2^{i+1}/n)^2$, because if $\mathcal{E}_i$ holds, then P_{i+1} in both data structures D_1 and D_2 must necessarily contain one of the at most k points in $P \cap h^+$. We get

$$\sum_{i=0}^{\log(n/k)-1} \left(\left(\log\left(\frac{n}{\log n}\right) - i\right)\log n + \frac{n}{2^i}\right)\Pr[\mathcal{E}_i]$$
$$\leq (\log n)\log\left(\frac{k}{\log n}\right) + \sum_{i=0}^{\log(n/k)-1}\left(\left(\log\left(\frac{n}{k}\right) - i\right)\log n + \frac{n}{2^i}\right)\left(\frac{k2^{i+1}}{n}\right)^2$$
$$\ll k + \sum_{j=1}^{\log(n/k)}\left(j\log n + 2^j k\right)2^{-2(j-1)} \ll \log n + k,$$

as desired. □

3.5 Few connected components

Finally, we consider colorings where the set of blue points has few connected components in conv P, and we will see that for this case we can get an algorithm which almost runs in linear time, without any preprocessing. For this, we look at the problem DisjUnion, where the task is the following: given point sets $P_1, \ldots, P_k \subseteq \mathbb{R}^3$ and their convex hulls conv P_1,..., conv P_k such that conv $P_i \cap$ conv $P_j = \emptyset$ for $i \neq j$ and such that $P = \bigcup_{i=1}^k P_i$ is in convex position, we would like to compute conv P. In general, we cannot do better than to repeatedly merge pairs of the hulls.

Theorem 3.5.1. *Any algorithm that solves* DisjUnion *requires* $\Omega(|P|\log k)$ *comparisons.*

Proof. We use an old lower bound [45, Section 4A] and combine it with Seidel's method of including the index as a coordinate [137]. We reduce from the *list merging* problem, in which k sorted lists of numbers need to be merged into one. We lift the lists onto the unit paraboloid $y = x^2$, using the z-coordinate to represent the index of the list. Clearly, the lifting and the individual convex hulls, which are pairwise disjoint, can be found in time $O(n)$. A simple geometric argument now shows that the merged list can be derived from the convex hull of the union in linear time; see Figure 3.9.

More precisely, consider the problem ListMerge: given k sorted integer sequences $L_1, \ldots, L_k$, compute the sorted list $L = \bigcup_{i=1}^k L_i$. A straightforward counting argument shows that any algorithm for ListMerge requires $\Omega(|L|\log k)$ comparisons. We describe a linear time reduction from ListMerge to DisjUnion:

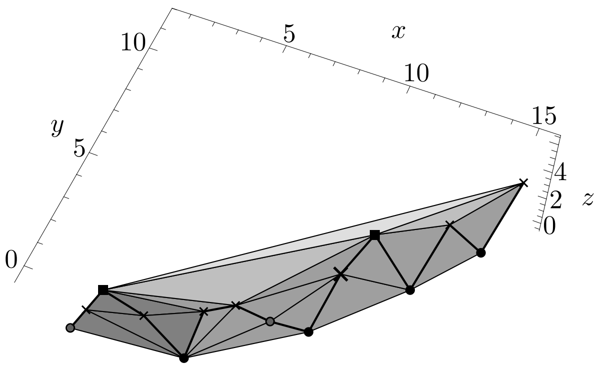


Figure 3.9: An illustration of the reduction from ListMerge to DisjUnion for the 3 lists $(5, 9, 12, 14)$, $(1, 8)$, $(2, 4, 6, 7, 10, 13)$, and $(3, 11)$. The path marked by the bold edges represents the merged list.

let $L_i = (r_1, \dots, r_j)$. We map L_i to a point set $P_i \subseteq \mathbb{R}^3$ by mapping each r_z to $p(r_z) = (r_z, r_z^2, i)$. All the points lie on the parabolic surface $y = x^2$, and hence $P = \bigcup_{i=1}^{k} P_i$ is in convex position. Furthermore, each P_i is contained in the plane $z = i$, and hence $\operatorname{conv} P_i \cap \operatorname{conv} P_j = \emptyset$ for $i \neq j$. The $\operatorname{conv} P_i$ can be computed in linear time, since the lists L_i are sorted.

If r, s are consecutive in the sorted list L, then $p(r)p(s)$ is an edge of $\operatorname{conv} P$. To see this, let $\hat{p}(r)$, $\hat{p}(s)$ denote the projections of $p(r), p(s)$ onto the xy-plane, and let h denote the plane orthogonal to the xy-plane that contains the line segment $\hat{p}(r)\hat{p}(s)$. By definition, h contains $p(r)$ and $p(s)$, and hence also the line segment $p(r)p(s)$. Furthermore, all other points of P are on the same side of h. For this, fix $i \in \{1, \dots, k\}$, and consider the parabola $Z_i = x \mapsto (x, x^2, i)$. Clearly, h intersects Z_i in the points (r, r^2, i) and (s, s^2, i), cutting off the part of Z_i between r and s. Since r and s are consecutive in L, this part contains no points in P_i. It follows that h supports the line segment $p(r)p(s)$, making it an edge of $\operatorname{conv} P$. Consequently, it takes $\Omega(|P| \log k)$ time to compute $\operatorname{conv} P$, since otherwise we could recover the sorted list L by examining the $O(|P|)$ edges of $\operatorname{conv} P$. $\square$

Intuitively, what makes our lower bound instance hard is the fact that when merging $\operatorname{conv} P_i$, we need to switch often between the individual hulls in an unpredictable way. We can avoid this by imposing additional constraints on the input, and thus obtain a better result.

Theorem 3.5.2. *Let $Q \subseteq \mathbb{R}^3$ be in gcp. Let $P = \bigcup_{i=1}^{k} P_i \subseteq Q$ with $|P| = n$ such that the P_i are pairwise disjoint and the subgraphs $\operatorname{conv} Q|_{P_i}$ are connected. Then, given spanning trees $T_1, \dots, T_k$ for $\operatorname{conv} Q|_{P_i}$, we can compute $\operatorname{conv} P$ in expected time $O(n \log^* n + k \log k)$.*

Proof. We use Seidel's tracing technique [138]: pick a subset $K \subseteq P$ that meets each T_i in exactly one point, and an appropriate gradation $S_0 \subseteq \cdots \subseteq S_\beta = P \setminus K$ with $\beta \ll \log^* n$. Then compute $\operatorname{conv}(S_0 \cup K)$ in time $O(n + k \log k)$ and successively each $\operatorname{conv}(S_i \cup K)$ in $O(n)$. Here, the bottleneck is to locate the conflict facets for S_{i+1} in $\operatorname{conv}(S_i \cup K)$. This is done using the spanning trees T_i and an appropriate variant of `SubsetConflictWalk`.

We may assume that $k < n/2$, since otherwise the theorem is easy. Let $K \subseteq P$ such that K contains exactly one point of each P_i, and let $m = n - k$. Let $z = \max\{k, m/\log m\}$ and choose $1 \le \alpha \le \log^* m$ such that $m/\log^{(\alpha-1)} m < z \le m/\log^{(\alpha)} m$, where $\log^{(i)} m$ denotes the i-th iterated logarithm[14] of m. Let $\beta = \log^* m - \alpha + 1$. Compute a gradation of subsets $S_0 \subseteq \cdots \subseteq S_\beta = P \setminus K$, such that S_i is a random subset of S_{i+1} with $|S_0| = z$ and $|S_{i+1}| = |S_i| \log^{(\alpha+i)} m / \log^{(\alpha+i+1)} m$ for $0 \le i < \beta$. By induction, it follows that $|S_i| \le m/\log^{(\alpha+i)}$. For $i = 0, \ldots, \beta$, let $\widetilde{S}_i = S_i \cup K$. We will show how to compute $\operatorname{conv} \widetilde{S}_{i+1}$ from $\operatorname{conv} \widetilde{S}_i$ in time $O(n)$ for each i. Furthermore, $\operatorname{conv} \widetilde{S}_0$ can be computed in time $O(n + k \log k)$ with a regular convex hull algorithm, as $|S_0 \cup K| = O(n/\log n + k)$. Hence, it takes $O(n \log^* n + k \log k)$ steps to compute $\operatorname{conv} Q = \operatorname{conv} \widetilde{S}_\beta$.

To derive $\operatorname{conv} \widetilde{S}_{i+1}$ from $\operatorname{conv} \widetilde{S}_i$ we proceed in two steps: first, we determine the conflict sets B_f for $f \in F[\widetilde{S}_i]$. Below, we will argue that this can be done in linear time. Then, we use the algorithm from Lemma 2.2.5 to compute $\operatorname{conv} \widetilde{S}_{i+1}$. This takes time proportional to

$$\left(|S_{i+1}| + k\frac{|S_{i+1}|}{|S_i|}\right) \log \frac{|S_{i+1}|}{|S_i|} \le 2|S_{i+1}| \log\left(\frac{\log^{(\alpha+i)} m}{\log^{(\alpha+i+1)} m}\right) \ll \frac{m}{\log^{(\alpha+i+1)} m} \log\left(\frac{\log^{(\alpha+i)} m}{\log^{(\alpha+i+1)} m}\right),$$

since $k \le |S_0| \le |S_i|$. The last term is $O(n)$, as claimed.

It remains to show how to find the conflict sets B_f in time $O(n)$. For each $j = 1, \ldots, k$, we determine conflict facets for P_j as follows: let $r_j = P_j \cap K$. We use a variant of `SubsetConflictWalk`: merge the neighbors of r_j in $\operatorname{conv} \widetilde{S}_i$ with the neighbors $\Gamma_{T_j}(r_j)$ of r_j in T_j in order to find a conflict facet f_p for each $p \in \Gamma_{T_j}(r_j)$. Then continue in a BFS-manner along T_j, inserting in turn each $p \in \Gamma_{T_j}(r_j)$ into $\operatorname{conv} \widetilde{S}_i$, and so on. As in Section 3.2, we see that the total time is proportional to

$$\sum_{p \in \widetilde{S}_i} \deg_{\widetilde{S}_i} p + \sum_{j=1}^{k} \sum_{p \in T_j} \deg_{T_j} p + \sum_{p \in P \setminus S_i} d_p \ll |\widetilde{S}_i| + |P| + \sum_{f \in F[\widetilde{S}_i]} b_f \ll |P| + n - k + k\frac{|S_{i+1}|}{|S_i|},$$

[14]Defined by $\log^{(0)} m = m$ and $\log^{(k)} m = \max\{1, \log(\log^{(k-1)} m)\}$ for $k \ge 1$.

by Lemma 2.2.4. Since $k \leq |S_i|$ and $|S_{i+i}| \leq n$, the last term is linear. This finishes the proof. $\square$

For our original question, this means that we can quickly compute the blue hull without considering the whole polytope, as long as the number of induced blue components is small.

Corollary 3.5.3. *Let $P \subseteq \mathbb{R}^3$ be a finite point set in gcp, and let B be a subgraph of $\operatorname{conv} P$ with n vertices. Then $\operatorname{conv} V[B]$ can be computed in time $O(n \log^* n + k \log k)$, where k denotes the number of connected components of B.*

Proof. This follows immediately from Theorem 3.5.2. Given B, we can find spanning trees for its components in $O(n)$ time, using, say, depth-first search [61]. $\square$

In particular, we get the following nice fact about Delaunay triangulations, which proved a Delaunay analogue to an old result by Bar-Yehuda and Chazelle [19].

Corollary 3.5.4. *Let $T = (V, E)$ be a Delaunay triangulation and let $S \subseteq T$ be a set of n vertices and edges of T with k connected components. Then the Delaunay triangulation of S can be computed in time $O(n \log^* n + k \log k)$.*

Proof. Use Corollary 3.5.3 and the connection between planar Delaunay triangulations and three-dimensional convex hulls. $\square$

Chapter 4

Interlude: Self-Improving Algorithms

Next, we shall explore how the methods developed in Chapter 3 can be used to simplify the analysis of a self-improving algorithm for Delaunay triangulations by Clarkson and Seshadhri [58]. Before we begin, let us explain what self-improving algorithms are and what is known about them.

Self-improving algorithms. The notion of self-improving algorithms was first introduced by Ailon et al. [4] in order to address shortcomings of the traditional average-case analysis of algorithms. Suppose we want to process a sequence of instances $I_1, I_2, \dots$ of a certain computational problem, and we know that the instances I_j are drawn independently according to some distribution $\mathcal{D}$. The goal is to design an algorithm that somehow takes advantage of the structure offered by $\mathcal{D}$ in order to achieve improved performance. If we knew $\mathcal{D}$, we could try to tailor a special algorithm to $\mathcal{D}$ and to prove better bounds for its running time. This is what usually happens in average case analysis. But what should we do if $\mathcal{D}$ is not known? The self-improving paradigm offers a solution: for the first few instances, the algorithm runs in its vanilla version and gives only the usual worst-case performance. But over time, it becomes acquainted with $\mathcal{D}$ and adapts itself accordingly, until eventually the algorithm becomes optimal for $\mathcal{D}$. Ailon et al. [4] designed such an algorithm for the sorting problem: suppose $\mathcal{D} = \prod_{i=1}^{n} \mathcal{D}_i$,[1] where each $\mathcal{D}_i$ is a distribution on $\mathbb{R}$, and let $\varepsilon \in (0, 1]$ be any constant. Then there is an algorithm for sorting instances drawn from $\mathcal{D}$ that needs $O(n \log n)$ steps for the first n^{ε} instances, and after that runs in optimal time $O(\varepsilon^{-1}(n + H(\mathcal{D})))$, where $H(\mathcal{D})$ is the entropy of the distribution that $\mathcal{D}$ induces on the set of permutations of $\{1, \dots n\}$. The space requirement is $O(n^{1+\varepsilon})$, and surprisingly this trade-off between space and

[1]$\mathcal{D}$ is a *product distribution*, which means the elements of each input are drawn independently of each other.

running time is best possible. Subsequently, Clarkson and Seshadhri [58] generalized this result to planar Delaunay triangulations. We shall give a brief description of their algorithm below and show a quick way to analyze it using the techniques we developed in Chapter 3.

4.1 Algorithm

Let $\mathcal{D} = \prod_{i=1}^{n} \mathcal{D}_i$, where each $\mathcal{D}_i$ is a (not necessarily discrete) distribution on $\mathbb{R}^2$. Consider a sequence $I_1, I_2, \dots$ of instances, where each I_j is drawn independently according to $\mathcal{D}$. That is, each I_j is a sequence of n points in the plane, and the i-th point in I_j is sampled independently according to $\mathcal{D}_i$. The goal is to compute the Delaunay triangulations $\mathrm{DT}(I_1), \mathrm{DT}(I_2), \dots$ in an optimal way. Namely, let $H(\mathcal{D})$ denote the entropy of the distribution on labeled graphs that is induced by $\mathrm{DT}(I), I \in_{\mathcal{D}} (\mathbb{R}^2)^n$. Fix a constant $\varepsilon \in (0, 1]$. Clarkson and Seshadhri [58] showed that there exists an algorithm that after n^{ε} rounds achieves an optimal expected running time of $O(\varepsilon^{-1}(n + H(\mathcal{D})))$, using space $O(n^{1+\varepsilon})$.

The algorithm operates in two phases: during the *learning phase*, $\mathrm{DT}(I)$ is computed in $O(n \log n)$ time, using a standard method [23]. At the same time, the algorithm records useful information and builds appropriate data structures to prepare for the *limiting phase*, in which the inputs are processed in a way that is optimally tailored to $\mathcal{D}$.

Learning Phase. After the first $\log n$ instances $I_1, \dots, I_{\log n}$, the algorithm determines $\hat{I} = \bigcup_{j=1}^{\log n} I_j$. Then it finds a $(1/\log n)$-*net* $V \subseteq \hat{I}$ for $\hat{I}$ with respect to open disks [60]. Specifically, V is a subset of $\hat{I}$ with the following properties: (i) $|V| = O(n)$; and (ii) for every planar open disk D with $|D \cap \hat{I}| \geq \log n$, we have $D \cap V \neq \emptyset$. Next, the algorithm computes $\mathrm{DT}(V)$ and stores it. Then it uses the following n^{ε} rounds to learn approximately the distributions $\mathcal{D}_1, \dots, \mathcal{D}_n$, and to build approximate entropy-optimal point location structures $T_1, \dots, T_n$ for $\mathrm{DT}(V)$, where T_j is optimal with respect to $\mathcal{D}_j$. Each of these structures needs $O(n^{\varepsilon})$ space and ensures that a point $x_i \in_{\mathcal{D}_i} \mathbb{R}^2$ can be located in $\mathrm{DT}(V)$ in expected time $O(\varepsilon^{-1} H^V(\mathcal{D}_i))$, where $H^V(\mathcal{D}_i)$ denotes the entropy of the distribution on the triangles of $\mathrm{DT}(V)$ induced by $\mathcal{D}_i$ [15, 16]. By distributing the work over the n^{ε} rounds, we can ensure that each of them needs $O(n \log n)$ steps, as desired.

Limiting Phase. Let $I = (x_1, \dots, x_n)$ be a problem instance in the limiting phase. We handle I as follows: for each x_i, we use T_i to find the triangle of $\mathrm{DT}(V)$ containing x_i. Then we walk along T_i to find the set of triangles D_i that have x_i in their (open) circumcircle. Having done this for all points, we determine for each triangle f in $\mathrm{DT}(V)$ the set $B_f \subseteq I$ of points inside its circumcircle, its *conflict set*. Using the connection between planar DTs and convex hulls in $\mathbb{R}^3$, and Lemma 2.2.5, we can find $\mathrm{DT}(V \cup S)$ in time $O\big(\sum_{f \in F[V]} b_f \log b_f\big)$, where $F[V]$ denotes the triangles of $\mathrm{DT}(V)$ and $b_f = |B_f|$.[2] Finally, we use Theorem 3.1.1 (or the previous algorithm by Chazelle et al. [47]) to obtain $\mathrm{DT}(I)$ in $O(n)$ time. All in all, this takes expected time $O\big(\varepsilon^{-1} \sum_{i=1}^{n} H^V(\mathcal{D}_i) + \sum_{f \in F[V]} b_f \log b_f + n\big)$, where the first term bounds the expected point-location time. In the next section we shall show that this is indeed optimal.

4.2 Analysis

To begin, we need to show that $\mathrm{DT}(V)$ in fact represents a "typical" problem instance. For this, we prove that with high probability over V, the expected conflict size for each triangle in $\mathrm{DT}(V)$ is constant [58, Claim 2.1].

Claim 4.2.1. *With probability at least* $1 - 1/n^3$ *over* V, *we have* $\mathbf{E}[b_f] \ll 1$ *and* $\mathbf{E}[b_f^2] \ll 1$ *for all triangles* f *in* $\mathrm{DT}(V)$.

Proof. Write $\hat{I} = s_1, \ldots, s_{n \log n}$, the concatenation of $I_1, \ldots, I_{\log n}$. Let f be the triangle given by s_1, s_2, s_3, and let C_f be f's circumcircle. For $j \in \{4, \ldots, n \log n\}$, denote by $Y_j^{(f)}$ the indicator random variable for the event $s_j \in C_f$. Note that all the $Y_j^{(f)}$ are independent of each other. Let $Y^{(f)} = \sum_{j=4}^{n} Y_j^{(f)}$, the conflict size of f. By independence and the Chernoff bound [119, Theorem 4.2], for any $\beta \in [0, 1]$,

$$\Pr\left[Y^{(f)} < (1-\beta)\mathbf{E}\left[Y^{(f)}\right]\right] \le \exp\left(-\beta^2 \mathbf{E}\left[Y^{(f)}\right]/2\right).$$

Setting $\beta = 13/14$, we see that if $\mathbf{E}[Y^{(f)}] > 14 \log n$, then $Y^{(f)} > \log n$ with probability at least $1 - n^{-6}$. By applying the above argument to any triangle $f_{\mathbf{u}}$ generated by some triple $\mathbf{u}$ of distinct points in $\hat{I}^3$, and taking a union bound, it follows that with probability at least $1 - n^{-3}$, for any triangle f generated by a triple of distinct points in $\hat{I}$, we have that

$$\text{if } Y^{(f)} \le \log n, \text{ then } \mathbf{E}[Y^{(f)}] \le 14 \log n. \tag{4.1}$$

From now on, we assume that this event happens.

Now let f be a triangle in $\mathrm{DT}(V)$, and C_f be its circumcircle. Since V is a $(1/n)$-net for $\hat{I}$ with respect to open disks, C_f contains less than $\log n$ points of $\hat{I}$, ie, $Y^{(f)} < \log n$. Thus, (4.1) implies $\mathbf{E}[Y^{(f)}] \ll \log n$. Now, since $\mathbf{E}[Y^{(f)}] \ge (\log n - 3)\mathbf{E}[b_f]$, we get $\mathbf{E}[b_f] \ll 1$, as claimed. To get the bound on $\mathbf{E}[b_f^2]$, note that b_f is a sum of independent nonnegative random variables, and apply the following claim:

[2]Specifically, the parameters for Lemma 2.2.5 are as follows: $P = V \cup I$, $p = n/(|V| + n) \gg 1$, $S = V$, and $K = \emptyset$.

Claim 4.2.2. *Let $Z = \sum_i Z_i$ be a sum of independent nonnegative random variables with $Z_i \ll 1$ for all i and $\mathbf{E}[Z] \ll 1$. Then $\mathbf{E}[Z^2] \ll 1$.*

Proof. By linearity of expectation,

$$\mathbf{E}[Z^2] = \mathbf{E}\big[\big(\sum_i Z_i\big)^2\big] = \sum_i \mathbf{E}[Z_i^2] + 2\sum_{i<j} \mathbf{E}[Z_i]\mathbf{E}[Z_j]$$
$$\ll \sum_i \mathbf{E}[Z_i] + \big(\sum_i \mathbf{E}[Z_i]\big)^2 \ll 1,$$

as desired. □

This finishes the proof of Claim 4.2.1. □

Using Claim 4.2.1, we can immediately bound the term $\sum_{f\in F[V]} b_f \log b_f$.

Lemma 4.2.3. *We have $\mathbf{E}\left[\sum_{f\in F[V]} b_f \log b_f\right] \ll n$.*

Proof. Since $b_f \log b_f \ll b_f^2$,

$$\mathbf{E}\left[\sum_{f\in F[V]} b_f \log b_f\right] \ll \sum_{f\in F[V]} \mathbf{E}\left[b_f^2\right] \ll |F[V]| \ll n,$$

by Claim 4.2.1 and because $|V| \ll n$. □

It remains to bound the sum $\sum_{i=1}^n H^V(\mathcal{D}_i)$. More specifically, we would like to show that this entropy is upperbounded by $H(\mathcal{D})$, which is the entropy of the distribution that $\mathrm{DT}(I), I \in_{\mathcal{D}} (\mathbb{R}^2)^n$, induces on the set of labeled graphs with n vertices. For this, we need to find a way to relate these two entropies. First, let $F = (f_1, \dots, f_n)$ be the random variable that assigns to each input $x_i \in I$ the facet f_i that contains it. The next claim, which is a standard fact about the joint entropy of independent random variables, shows that $H(F) = \sum_{i=1}^n H^V(\mathcal{D}_i)$.

Claim 4.2.4. *Let $H(Z_1, \dots, Z_n)$ be the joint entropy of independent random variables $Z_1, \dots, Z_n$. Then $H(Z_1, \dots, Z_n) = \sum_i H(Z_i)$.*

Proof. This follows by an inductive application of the binary chain rule $H(X, Y) = H(X \mid Y) + H(Y)$, and the fact that $H(X \mid Y) = H(X)$ for independent random variables X and Y. Here, $H(X \mid Y)$ is the *conditional entropy* defined as $H(X \mid Y) = \sum_y \Pr[Y = y] H(X \mid Y = y)$. □

Hence, we need to bound $H(F)$ in terms of $H(\mathcal{D})$. The next claim, which is implicit in Clarkson and Seshadhri [58] and was stated formally in a follow-up version of their paper [3], demonstrates that we can do this by designing an appropriate algorithm.

Claim 4.2.5. *Let $\mathcal{D}$ be a distribution on a universe $\mathcal{U}$, and let $X : \mathcal{U} \to \mathcal{X}$ and $Y : \mathcal{U} \to \mathcal{Y}$ be two random variables. Suppose that the function $f : \mathcal{U} \times X(\mathcal{U}) \to \mathcal{Y}$ defined by $f : (I, X(I)) \mapsto Y(I)$ can be computed with $O(n)$ expected comparisons (where the expectation is over $\mathcal{D}$). Then $H(Y) = O(n + H(X))$, where all the entropies are with respect to $\mathcal{D}$.*

Proof. By a classic result from information theory (eg, [62, Theorem 5.4.1]), any unique encoding $s : X(\mathcal{U}) \to \{0,1\}^*$ of $X(\mathcal{U})$ has an expected code length of $E_{\mathcal{D}}[|s(X(I))|] \geq H(X)$, and there exists an encoding s^* that has expected code length $O(H(X))$. Using f, this can be converted into an encoding t of $Y(\mathcal{U})$. Indeed, for every I, $Y(I)$ can be uniquely identified using $s^*(X(I))$ and additional bits that represent the outcomes of the comparisons for the computation of $f(I, X(I))$. By taking a shortest such string for each element of $Y(\mathcal{U})$, we obtain a unique encoding t for $Y(\mathcal{U})$ with excepted code length $E_{\mathcal{D}}[|t(Y(I))|] = O(n + E_{\mathcal{D}}[|s(X(I))|]) = O(n + H(X))$. Since any encoding of $Y(\mathcal{U})$ has expected code length at least $H(Y)$, the claim follows. □

Hence, in order to achieve the desired entropy bound, we only need to design an algorithm that given $\mathrm{DT}(I)$ computes $F = (f_1, \ldots, f_n)$ by determining for each $x_i \in I$ the triangle f_i of $\mathrm{DT}(V)$ that contains it. Clarkson and Seshadhri do this by giving a delicate multistage algorithm with a subtle analysis, but using `SubsetConflictWalk` from Chapter 3, the result becomes almost immediate.

Claim 4.2.6. *There exists an algorithm that given $\mathrm{DT}(V)$ and $\mathrm{DT}(I)$ finds for each $x_i \in I$ the triangle of $\mathrm{DT}(V)$ that contains it, with a total expected number of $O(n)$ comparisons.*

Proof. First, find $\mathrm{DT}(V \cup I)$ in $O(n)$ time, using an algorithm by Chazelle [45]. Then use the connection between planar DTs and convex hulls in $\mathbb{R}^3$ and the algorithm `SubsetConflictWalk` (Algorithm 3.3) to find for each $x_i \in I$ the triangle containing it. By the proof of Lemma 3.2.3, `SubsetConflictWalk` takes time proportional to $\sum_{f \in F[V]} b_f \ll n$, by Claim 4.2.1. This finishes the proof. □

Now we have all the ingredients for the desired entropy bound.

Lemma 4.2.7. *We have $\sum_i H^V(\mathcal{D}_i) \ll n + H(\mathcal{D})$.*

Proof. As defined above, let $F = (f_1, \ldots, f_n)$ be the random variable that assigns to $x_i \in I$ the triangle $f_i \in F[V]$ that contains it. By Clam 4.2.4, $\sum_i H^V(\mathcal{D}_i) = H(F)$. Next, consider the function $f : (I, \mathrm{DT}(I)) \mapsto F(I)$. By Claim 4.2.6, f can be computed with $O(n)$ expected comparisons. So, by Claim 4.2.5, we have $H(F) \ll n + H(\mathcal{D})$, as desired. □

Hence, we have shown that the algorithm from Section 4.1 is indeed an optimal self-improving algorithm for Delaunay triangulations.

Theorem 4.2.8. *After a learning phase of n^{ε} rounds and using space $n^{1+\varepsilon}$, the algorithm from Section 4.1 achieves the following guarantee: with probability at least $1 - n^{-3}$ over the learning phase, the expected running time per instance $I \in_{\mathcal{D}} (\mathbb{R}^2)^n$ is $O(\varepsilon^{-1}(n + H(\mathcal{D})))$.*

Proof. The bounds on the length of the learning phase and on the space usage where already justified in Section 4.1. There, we also saw that the expected running time in the limiting phase is $O\big(\varepsilon^{-1} \sum_{i=1}^{n} H^V(\mathcal{D}_i) + \sum_{f \in F[V]} b_f \log b_f + n\big)$, which by Lemmas 4.2.3 and 4.2.7 is $O(\varepsilon^{-1}(n + H(\mathcal{D})))$ with probability at least $1 - n^{-3}$ over the learning phase. □

Chapter 5

Transdichotomous Delaunay Triangulations

We now turn to the structure offered by *transdichotomous models*, a term coined by Fredman and Willard [80,81]. Sorting n numbers takes $\Omega(n \log n)$ time—and yet, as we already discussed in Chapter 1, this lower bound can often be broken by going beyond the comparison based model. In that model, each step can distinguish only between two different outcomes: either a predicate holds, or it does not hold; the model is therefore *dichotomous*.[1] By using methods such as table lookup or bit-level operations, however, we can implement decisions with more than two possible outcomes; therefore, models that use these operations are *transdichotomous*.[2] Under the right assumptions, radix sort and bucket sort run in linear time [61]. Using van Emde Boas (vEB) trees [71, 72], we can sort n elements from a universe U in $O(n \log\log |U|)$ time on a pointer machine. In a transdichotomous model, we can surpass the sorting lower bound with *fusion trees*, achieving $O(n\sqrt{\log n})$ time. Fusion trees were introduced in 1990 by Fredman and Willard [80] and triggered off a development (see, for example, [11, 87, 88, 129, 144]) that culminated in the $O(n\sqrt{\log\log n})$ time integer sorting algorithm by Han and Thorup [88]. For small and large word sizes (that is, for word size $w \ll \log n$ or $w \gg \log^{2+\varepsilon} n$), we can even sort in linear time (via radix sort [61], resp. signature sort [11]).

In computational geometry, there have been many results that use vEB trees or similar structures to overcome traditional lower bounds (eg, [10, 25, 50, 92, 96, 97, 126]). However, these results assume that the input is rectilinear or can be efficiently approximated by a rectilinear structure, such as, for example, a quadtree. In this sense, the above results are all *orthogonal*. Similarly, Willard [149] applied fusion

[1]The Oxford English Dictionary (OED) defines *dichotomy* as the “division of a whole into two parts”.

[2]According to the OED, the prefix *trans–* can mean “across, through, over, to or on the other side of, beyond, outside of”.

trees to achieve better bounds for orthogonal range searching, axis-parallel rectangle intersection, and others. Again, his results are all orthogonal, and he asked whether improved bounds can be attained for Voronoi diagrams. The breakthrough came in 2006, when Chan and Pătraşcu [40] discovered transdichotomous algorithms for point location in non-orthogonal planar subdivisions. This led to better bounds for many classic computational geometry problems. In a follow-up paper [41], they considered off-line planar point location and thereby improved the running time for Delaunay triangulations, three-dimensional convex hulls, and other problems. The running time is a rather unusual $n2^{O(\sqrt{\log\log n})}$, which raises the question whether the result is optimal. More generally, Chan and Pătraşcu asked if the approach via point location is inherent, or if there are more direct algorithms for convex hulls or Delaunay triangulations.

One of the main results of this chapter will be that for Delaunay triangulations (DTs), we can indeed do better. For this, we will describe a randomized reduction from DTs to nearest-neighbor graphs (NNGs).[3] Our method uses a new variant of the classic randomized incremental construction (RIC) paradigm [9, 59, 122] that relies on *dependent* sampling for faster conflict location. If NNGs can be computed in linear time, the running time of our reduction is proportional to the structural change of a standard RIC, which is always linear for planar point sets and also in many other cases, eg, point sets suitably sampled from a $(d-1)$-dimensional polyhedron in $\mathbb{R}^d$ [8, 17].[4] The algorithm is relatively simple and works in any dimension, but the analysis turns out to be rather subtle. It is a well-known fact that given a quadtree for a point set, its nearest-neighbor graph can be computed in linear time [35, 39, 54]. This leads to the main take-away message from this chapter: *Given a quadtree for a point set $P \subseteq \mathbb{R}^d$, we can compute the Delaunay triangulation of P, $\mathrm{DT}(P)$, in time proportional to the expected structural change of a RIC.* This constitutes an improvement over classic RICs whenever the expected structural change is low (ie, $o(n \log n)$), because in that case a classic RIC would incur an $\Omega(n \log n)$ overhead to maintain the conflict lists, which we now avoid. This connection between quadtrees and DTs may be surprising, since even though DTs appear to be inherently non-orthogonal, we actually need only the information encoded in quadtrees, a highly orthogonal structure. We use this idea to obtain several results.

- **DTs on a word RAM.** We answer Willard's seventeen-year-old open question by showing that planar DTs, and hence planar Voronoi diagrams and

[3]Given a point set $P \subseteq \mathbb{R}^d$, the nearest-neighbor graph of P, $\mathrm{NN}(P)$, it the undirected graph with vertex set P that contains an edge between every point $p \in P$ and its nearest neighbor in $P \setminus p$, ie, the point $q \in P \setminus p$ that minimizes $\|p-q\|_2$ (by general position, we may assume that q is unique).

[4]The bound in the references is only proved for the complexity of the final DT, but we believe that it can be extended to the structural change of a RIC.

related structures like Euclidean minimum spanning trees, can be computed in time $O(\texttt{sort}(n))$ on a word RAM.[5] As given, our algorithm requires one non-standard, but AC^0, operation, the *shuffle*. However, in Section 5.3 we show how to remove this assumption by giving a slightly weaker $O(n \log\log n)$ time algorithm for transdichotomous planar DTs. This is done by adapting the (comparatively simple) $O(n \log\log n)$ sorting algorithm by Andersson et al. [11].

- **DTs from a fixed universe.** We can preprocess a point set $U \subseteq \mathbb{R}^d$ such that for any subset $P \subseteq U$, it takes $O(|P| \log\log |U| + C(P))$ time to find $\mathrm{DT}(P)$. Here, $C(P)$ denotes the expected structural change of a RIC on P.

- **DTs for presorted point sets.** Since a planar quadtree can be computed by an algebraic computation tree (ACT) [14] of linear depth once the points are sorted according to the x- and y-direction, we find that after presorting in *two* orthogonal directions, a planar DT can be computed by an ACT of expected linear depth. This should be compared with the fact that there is an $\Omega(n \log n)$ lower bound when the points are sorted in *one* direction [67], and also for convex hulls in $\mathbb{R}^3$ when the points are sorted in any constant number of directions [137]. This problem has appeared in the literature for at least twenty years [1, 50, 67]. Our result seems to mark the first non-trivial progress on this question, and it shows that unlike for convex hulls and point sets sorted in one direction, a Ben-Or style lower bound in the algebraic decision tree model [22] does not exist. However, we do not know if a quadtree for presorted points can indeed be constructed in linear time, since the algorithms we know still need an $\Omega(n \log n)$ overhead for data handling. It would be interesting to see if there is a connection to the notorious SORTING $X + Y$ problem [79], which seems to exhibit a similar behavior.

In the next chapter, we will see another application of your reduction, namely for *restricted* point sets.

The shuffle operation. In the following sections, we will consider several different computational models, and we refer the reader to Section 2.3 for more background. In particular, for our transdichotomous result, we will extend the standard word RAM from Section 2.3 by one nonstandard operation: given a point $p \in \mathbb{R}^d$ with w-bit coordinates $p_{1w} \dots p_{12}p_{11}$, $p_{2w} \dots p_{21}$, $\dots$, $p_{dw} \dots p_{d1}$, the result of $\texttt{shuffle}(p)$ is the dw-bit word $p_{1w}p_{2w} \dots p_{dw} \dots p_{12} \dots p_{d2}p_{11} \dots p_{d1}$, see Figure 5.1. Clearly, `shuffle` is in AC^0, and we assume that it takes constant time on our RAM. In Section 5.3, we shall explain how this assumption can be dropped.

[5] $\texttt{sort}(n)$ is the time needed to sort n words. The currently best known bound depends on the word size w of the word RAM, but is never more than $O(n\sqrt{\log\log n})$ [88].

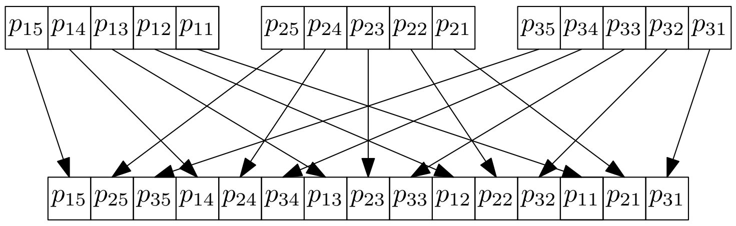

p_{15} p_{14} p_{13} p_{12} p_{11}
p_{25} p_{24} p_{23} p_{22} p_{21}
p_{35} p_{34} p_{33} p_{32} p_{31}
p_{15} p_{25} p_{35} p_{14} p_{24} p_{34} p_{13} p_{23} p_{33} p_{12} p_{22} p_{32} p_{11} p_{21} p_{31}

Figure 5.1: The shuffle-operation for a 3-dimensional point with 5-bit coordinates.

5.1 From nearest-neighbor graphs to Delaunay triangulations

We now describe our reduction from nearest-neighbor graphs (NNGs) to Delaunay triangulations (DTs). This is done by a randomized algorithm which we call `BrioDC`, which stands for **B**iased **R**andom **I**nsertion **O**rder with **D**ependent **C**hoices. Please consider Algorithm 5.1.

Algorithm 5.1 Reducing Delaunay triangulations to nearest-neighbor graphs.

```
BrioDC($P$)

  1. If $|P| = O(1)$, compute DT($P$) directly and return.
  2. Compute NN($P$), the nearest-neighbor graph for $P$.
  3. Let $S \subseteq P$ be a random sample such that (i) $S$ meets every connected com-
     ponent of NN($P$) and (ii) $\Pr[p \in S] = 1/2$, for all $p \in P$.
  4. Call BrioDC($S$) to compute DT($S$).
  5. Compute DT($P$) by inserting the points in $P \setminus S$ into DT($S$), using NN($P$)
     as a guide.
```

The algorithm is similar to the one described in Theorem 3.3.3 from Chapter 3. To find S in Step 3, we define a partial matching $\mathcal{M}(P)$ on P by pairing up two arbitrary points in each component of NN(P), the nearest-neighbor graph of P. Then, S is obtained by picking one random point from each pair in $\mathcal{M}(P)$ and sampling the points in $P \setminus \mathcal{M}(P)$ independently with probability 1/2 (although they could also be paired up). In Step 5, we successively insert the points from $P \setminus S$ as follows: pick a point $p \in P \setminus S$ that has not been inserted yet and is adjacent in NN(P) to a point q in the current DT. Such a point always exists by the definition of S. Walk along the edge qp to locate p in the current DT, and insert it. Repeat until all of P has been processed. The algorithm is illustrated in Figure 5.2.

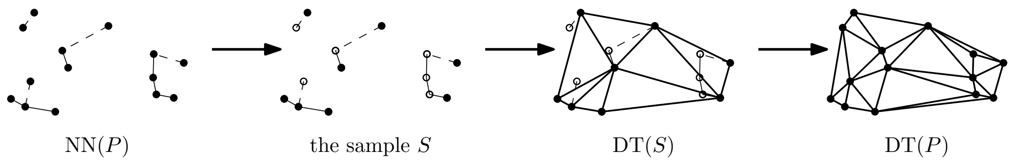


Figure 5.2: The algorithm `BrioDC`: the edges of $\mathcal{M}(P)$ are shown dashed, the remaining edges of NN(P) are solid.

Theorem 5.1.1. *Suppose the nearest-neighbor graph of an m-point set can be found in $f(m)$ time, where $f(m)/m$ is increasing. Let $P \subseteq \mathbb{R}^d$ be an n-point set. The expected running time of `BrioDC` is $O(C(P) + f(n))$, where $C(P)$ is the expected structural change of a RIC on P. The constant in the O-notation depends exponentially on d.*

Let $P = S_0 \supseteq \cdots \supseteq S_\ell$ be the sequence of samples taken by `BrioDC`. Fix a set $\mathbf{u}$ of $d+1$ distinct points in P. Let Δ be the simplex spanned by $\mathbf{u}$, and let $B_\mathbf{u} \subseteq P$ denote the points inside Δ's circumsphere. We call $\mathbf{u}$ the *trigger* set and $B_\mathbf{u}$ the *conflict* set for Δ. The elements of $B_\mathbf{u}$ are called *stoppers*. Consider the event A_α that Δ occurs during the construction of DT(S_α) from DT($S_{\alpha+1}$), for some α. Clearly, A_α can only happen if $\mathbf{u} \subseteq S_\alpha$ and $B_\mathbf{u} \cap S_{\alpha+1} = \emptyset$. To prove Theorem 5.1.1, we bound $\Pr[A_\alpha]$.

Lemma 5.1.2. *We have*

$$\Pr[A_\alpha] \leq e^{2d+2}\, 2^{-(d+1)\alpha} \left(1 - 2^{-\alpha-1}\right)^{|B_u|}.$$

We visualize the sampling process as follows [119, Chapter 1.4]: imagine a particle that moves at discrete time steps on the nonnegative x-axis and always occupies integer points. Please refer to Figure 5.3. The particle starts at position $|B_\mathbf{u}|$, and after β steps, it is at position $|S_\beta \cap B_\mathbf{u}|$, the number of stoppers in the current sample. The goal is to upperbound the probability of reaching 0 in $\alpha+1$ steps while retaining all triggers. However, the random choices in a step not only depend on the current position, but also on the matching $\mathcal{M}(S)$. Even worse, the probability distribution in the current position may depend on the previous positions of the particle. We avoid these issues through appropriate conditioning and show that the random walk essentially behaves like a Markov process that in each round eliminates $d+1$ stoppers and samples the remaining stoppers independently. The elimination is due to trigger-stopper pairs in $\mathcal{M}(S)$, since we want all triggers to survive. The remaining stoppers are not necessarily independent, but dependencies can only help, because in each stopper-stopper pair one stopper is guaranteed to survive. Eliminating $d+1$ stoppers in the ith step has a similar effect as starting with about $(d+1)2^i$ fewer stoppers: though a given trigger can be matched with only one stopper per round, these pairings can vary for different instances of the walk, and since a given stopper survives a round with probability roughly $1/2$, the "amount" of stoppers eliminated by one trigger in all instances roughly doubles per round.

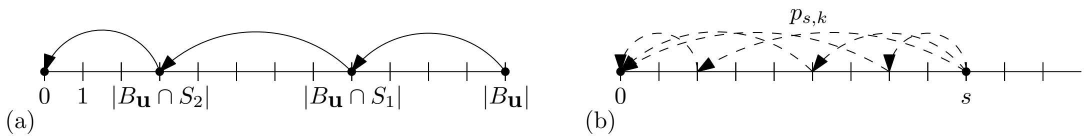


Figure 5.3: (a) We visualize the sampling process as a particle moving on the positive x-axis from $|L_{\mathbf{u}}|$ towards 0; (b) $p_{s,k}$ roughly corresponds to the probability of reaching 0 from s in k steps, while retaining all the triggers. We use appropriate conditioning in order to deal with the dependencies between the different events.

Proof of Lemma 5.1.2. For $S \subseteq P$, let the *matching profile* for S be the triple $(a,b,c) \in \mathbb{N}_0^3$ that counts the number of trigger-stopper, stopper-stopper, and trigger-trigger pairs in $\mathcal{M}(S)$. We consider

$$p_{s,k} = \max_{\mathcal{P}_k} \Pr[A_\alpha \mid X_{s,k}, \mathcal{P}_k], \tag{5.1}$$

where $X_{s,k} = \{\mathbf{u} \subseteq S_{\alpha-k}\} \cap \{|B_{\mathbf{u}} \cap S_{\alpha-k}| = s\}$ is the event that the sample $S_{\alpha-k}$ contains all triggers and exactly s stoppers. The maximum in (5.1) is taken over all possible sequences $\mathcal{P}_k = \mathbf{m}_0, \ldots, \mathbf{m}_{\alpha-k-1}, Y_0, \ldots, Y_{\alpha-k-1}$ of matching profiles $\mathbf{m}_i$ for S_i and events Y_i of the form $X_{t_i,\alpha-i}$ for some t_i. Since $\Pr[A_\alpha] = p_{|B_{\mathbf{u}}|,\alpha}$, it suffices to upperbound $p_{s,k}$. We describe a recursion for $p_{s,k}$. For that, let $T_k = \{\mathbf{u} \subseteq S_{\alpha-k}\}$ be the event that $S_{\alpha-k}$ contains all the triggers, and let $U_{k,i} = \{|B_{\mathbf{u}} \cap S_{\alpha-k}| = i\}$ denote the event that $S_{\alpha-k}$ contains exactly i stoppers.

Proposition 5.1.3. *We have*

$$p_{s,k} \leq \max_{\boldsymbol{m}} \Pr[T_{k-1} \mid X_{s,k}, \boldsymbol{m}] \cdot \sum_{i=0}^{s} p_{i,k-1} \Pr[U_{k-1,i} \mid T_{k-1}, X_{s,k}, \boldsymbol{m}],$$

where the maximum is over all possible matching profiles $\boldsymbol{m} = (a,b,c)$ for $S_{\alpha-k}$.

Proof. Fix a sequence $\mathcal{P}_k$ as in (5.1). Then, by distinguishing how many stoppers are present in $S_{\alpha-k+1}$,

$$\Pr[A_\alpha \mid X_{s,k}, \mathcal{P}_k] = \sum_{i=0}^{s} \Pr[X_{i,k-1} \mid X_{s,k}, \mathcal{P}_k] \Pr[A_\alpha \mid X_{i,k-1}, X_{s,k}, \mathcal{P}_k].$$

Now if we condition on a matching profile $\mathbf{m}$ for $S_{\alpha-k}$, we get

$$\Pr[X_{i,k-1} \mid X_{s,k}, \mathcal{P}_k, \mathbf{m}] = \Pr[T_{k-1} \mid X_{s,k}, \mathbf{m}] \Pr[U_{k-1,i} \mid T_{k-1}, X_{s,k}, \mathbf{m}],$$

since the distribution of triggers and stoppers in $S_{\alpha-k+1}$ becomes independent of $\mathcal{P}_k$ once we know the matching profile and the number of triggers and stoppers in $S_{\alpha-k}$. Furthermore,

$$\Pr[A_\alpha \mid X_{i,k-1}, \mathbf{m}, X_{s,k}, \mathcal{P}_k] \leq \max_{\mathcal{P}_{k+1}} \Pr[A_\alpha \mid X_{i,k-1}, \mathcal{P}_{k+1}] = p_{i,k-1}.$$

The claim follows by taking the maximum over $\mathbf{m}$. $\square$

We use Proposition 5.1.3 to bound $p_{s,k}$: if $\mathbf{m} = (a, b, c)$ pairs up two triggers, we get $\mathbf{u} \not\subseteq S_{\alpha-k+1}$ and $\Pr[T_{k-1} \mid X_{s,k}, \mathbf{m}] = 0$. Hence we can assume $c = 0$ and therefore $\Pr[T_{k-1} \mid X_{s,k}, \mathbf{m}] = 1/2^{d+1}$, since all triggers are sampled independently. Furthermore, none of the a stoppers paired with a trigger and half of the $2b$ stoppers paired with a stopper end up in $S_{\alpha-k+1}$, while the remaining $t_\mathbf{m} = s-a-2b$ stoppers are sampled independently. Thus, Proposition 5.1.3 gives

$$p_{s,k} \leq \max_{\substack{\mathbf{m} \\ c=0}} \sum_{i=b}^{s-a-b} \frac{p_{i,k-1}}{2^{d+1}} \Pr\left[B_{1/2}^{t_\mathbf{m}} = i - b\right], \tag{5.2}$$

where $B_{1/2}^{t_\mathbf{m}}$ denotes a binomial distribution with $t_\mathbf{m}$ trials and success probability $1/2$.

Proposition 5.1.4. *We have*

$$p_{s,k} \leq 2^{-(d+1)k} \left(1 - 2^{-k-1}\right)^s \prod_{j=1}^{k} \left(1 - 2^{-j}\right)^{-d-1}.$$

Proof. The proof is by induction on k. For $k = 0$, we have $p_{s,0} \leq (1 - 1/2)^s$, since we require that none of the s stoppers in S_α be present in $S_{\alpha+1}$, and this can only happen if they are sampled independently of each other. Furthermore, by (5.2),

$$p_{s,k+1} \leq \max_{\substack{\mathbf{m} \\ c=0}} \sum_{i=b}^{s-a-b} \frac{p_{i,k}}{2^{d+1}} \Pr\left[B_{1/2}^{t_\mathbf{m}} = i - b\right] = \max_{\substack{\mathbf{m} \\ c=0}} \frac{1}{2^{d+1+t_\mathbf{m}}} \sum_{i=0}^{t_\mathbf{m}} \binom{t_\mathbf{m}}{i} p_{i+b,k}. \tag{5.3}$$

Using the inductive hypothesis and the binomial theorem, we bound the sum as

$$\begin{aligned}\sum_{i=0}^{t_{\mathbf{m}}}\binom{t_{\mathbf{m}}}{i}p_{i+b,k} &\le \frac{\sum_{i=0}^{t_{\mathbf{m}}}\binom{t_{\mathbf{m}}}{i}\left(1-2^{-k-1}\right)^{i+b}}{2^{(d+1)k}}\prod_{j=1}^{k}\left(1-2^{-j}\right)^{-d-1}\\ &= \frac{\left(2-2^{-k-1}\right)^{t_{\mathbf{m}}}}{2^{(d+1)k}}\left(1-2^{-k-1}\right)^{b}\prod_{j=1}^{k}\left(1-2^{-j}\right)^{-d-1}\\ &= \frac{\left(1-2^{-k-2}\right)^{t_{\mathbf{m}}}}{2^{(d+1)k-t_{\mathbf{m}}}}\left(1-2^{-k-1}\right)^{b}\prod_{j=1}^{k}\left(1-2^{-j}\right)^{-d-1}.\end{aligned}$$

Now, since $t_{\mathbf{m}} = s-a-2b \ge s-d-1-2b$ and since

$$\frac{\left(1-2^{-k-1}\right)^{b}}{\left(1-2^{-k-2}\right)^{2b}} = \left(\frac{1-2^{-k-1}}{1-2^{-k-1}+2^{-2k-4}}\right)^{b} \le 1,$$

it follows that

$$\sum_{i=0}^{t_{\mathbf{m}}}\binom{t_{\mathbf{m}}}{i}p_{i+b,k} \le \frac{\left(1-2^{-k-2}\right)^{s}}{2^{(d+1)k-t_{\mathbf{m}}}}\prod_{j=1}^{k+1}\left(1-2^{-j}\right)^{-d-1},$$

and hence (5.3) gives

$$p_{s,k+1} \le \frac{\left(1-2^{-k-2}\right)^{s}}{2^{(d+1)(k+1)}}\prod_{j=1}^{k+1}\left(1-2^{-j}\right)^{-d-1},$$

which finishes the induction. □

Now, since $1-x \ge \exp(x/(x-1))$ for $x<1$ we have

$$\prod_{j=1}^{k}(1-2^{-j})^{-d-1} \le \exp\left(2(d+1)\sum_{j=1}^{\infty}2^{-j}\right) = e^{2(d+1)},$$

so we get

$$\Pr[A_\alpha] \le p_{|B_{\mathbf{u}}|,\alpha} \le e^{2d+2}2^{-(d+1)\alpha}(1-2^{-\alpha-1})^{|B_{\mathbf{u}}|},$$

which proves Lemma 5.1.2. □

Proof of Theorem 5.1.1. Since $f(m)/m$ increases, the expected cost to compute the NNGs for all the samples is $O(n)$. Furthermore, the cost of tracing an edge pq of NN(P), where p is in the current DT and q will be inserted next consists of (a) the cost of finding the starting simplex at p and (b) the cost of walking through the DT. Part (a) can be bounded by the degree of p in the current DT. In total, any simplex appears at most as often as the total degree of its vertices in NN(P), which is constant [116, Corollary 3.2.3]. Hence, (a) is proportional to the structural change. The same holds for (b), since every traversed simplex will be destroyed when the next point is inserted.

It is now sufficient to show that the probability that the simplex spanned by $\mathbf{u} \subseteq P$ occurs in `BrioDC` is asymptotically upperbounded by the corresponding probability in a RIC. In the case of `BrioDC`, this probability is bounded by $\sum_{\alpha=0}^{\infty} \Pr[A_\alpha]$. Performing an analysis similar to Buchin [34, Theorem 3.8], and writing $b_{\mathbf{u}} = |B_{\mathbf{u}}|$, we have by Lemma 5.1.2,

$$\begin{aligned}\sum_{\alpha=0}^{\infty} \Pr[A_\alpha] &\ll \sum_{\alpha=0}^{\infty} 2^{-(d+1)\alpha}\left(1 - 2^{-\alpha-1}\right)^{b_{\mathbf{u}}} \\ &\ll \sum_{\alpha=0}^{\infty} 2^{-(d+1)\alpha-(\log e)b_{\mathbf{u}}/2^{\alpha+1}}.\end{aligned}$$

Using elementary calculus, we see that the function $\alpha \mapsto -(d+1)\alpha - (\log e)b_{\mathbf{u}}/2^{\alpha+1}$ reaches its maximum for $\alpha^* = \log(b_{\mathbf{u}}/(d+1)) - 1$. Therefore,

$$\sum_{\alpha=0}^{\alpha} \Pr[A_\alpha] \ll \sum_{\alpha=0}^{\alpha^*} 2^{-(d+1)(\alpha^*-\alpha)-(\log e)b_{\mathbf{u}}/2^{\alpha*+1-\alpha}} + \sum_{\alpha=1}^{\infty} 2^{-(d+1)(\alpha^*+\alpha)-(\log e)b_{\mathbf{u}}/2^{\alpha*+\alpha+1}}.$$

Now,

$$\begin{aligned}\sum_{\alpha=0}^{\alpha^*} 2^{-(d+1)(\alpha^*-\alpha)-(\log e)b_{\mathbf{u}}/2^{\alpha*+1-\alpha}} &= \sum_{\alpha=0}^{\alpha^*} 2^{-(d+1)\log(b_{\mathbf{u}}/(d+1))+(d+1)(1+\alpha)-(\log e)(d+1)2^\alpha} \\ &\le 2^{d+1}\left(\frac{d+1}{b_{\mathbf{u}}}\right)^{d+1} \cdot \sum_{\alpha=0}^{\infty} \left(2^{d+1}\right)^{\alpha-(\log e)2^\alpha} \\ &\ll b_{\mathbf{u}}^{-(d+1)},\end{aligned}$$

and

$$\begin{aligned}\sum_{\alpha=1}^{\infty} 2^{-(d+1)(\alpha^*+\alpha)-(\log e)b_{\mathbf{u}}/2^{\alpha*+\alpha}} &\le \sum_{\alpha=1}^{\infty} 2^{-(d+1)(\log(b_{\mathbf{u}}/(d+1)))-(d+1)(\alpha-1)} \\ &= \left(\frac{d+1}{b_{\mathbf{u}}}\right)^{d+1} \sum_{\alpha=0}^{\infty} 2^{-(d+1)\alpha} \\ &\ll b_{\mathbf{u}}^{-(d+1)}.\end{aligned}$$

It follows that the probability that the simplex spanned by $\mathbf{u}$ appears in `BrioDC` is $O(1/b_{\mathbf{u}}^{d+1})$. In a standard RIC, the analogous probability is the probability for the event that in a random permutation of $\mathbf{u} \cup B_{\mathbf{u}}$ all the triggers appear before all the stoppers. This is bounded by

$$\frac{(d+1)!b_{\mathbf{u}}!}{(d+1+b_{\mathbf{u}})!} = \prod_{k=1}^{d+1} \frac{k}{b_{\mathbf{u}}+k} \geq \left(\frac{1}{b_{\mathbf{u}}+1}\right)^{d+1} \gg 1/b_{\mathbf{u}}^{d+1}.$$

Hence, it follows that the probability that the simplex spanned by $\mathbf{u}$ appears during `BrioDC` is asymptotically upperbounded by the analogous probability in a standard RIC, as needed. □

Remark. The reduction also shows that it takes $\Omega(n \log n)$ time to compute NNGs and well-separated pair decompositions, even if the input is sorted along one direction [67].

Remark. The dependent sampling has more advantages than just allowing for fast point location. For instance, if P samples a region, eg, a surface [17], in the sense that for any point in the region there is a point in P at distance at most ε, then similar guarantees with increasing ε still hold for $S = S_1, S_2, \ldots$. Furthermore, Lemma 5.1.2 directly extends to the more general setting of *configuration spaces* [122] by replacing $d+1$ by the degree bound, ie, the maximum number of triggers.[6] Thus, our dependent sampling scheme can be used in the incremental construction of a wide range of structures, and may be useful in further applications.

Remark. Finally, we note that if P is planar, the proof of Theorem 5.1.1 can be simplified considerably:

A simple proof of Theorem 5.1.1 for $d = 2$. As we argued at the beginning of the proof of Theorem 5.1.1, it suffices to bound the structural change. For this, we use induction to show that there exists a constant c such that the expected structural change is at most cn. This clearly holds for $n \ll 1$. Now we suppose that DT(S) can be found with expected structural change at most $c|S|$, and we examine what happens in Step 5 of `BrioDC`. Let p_s be the probability that a given triangle f with conflict size s appears in Step 5. Clearly, we have $p_s \leq 1/2^s$: if the stoppers of t are sampled independently of each other, we directly get this bound, and otherwise S includes a stopper and f cannot be created at all. By the well known Clarkson-Shor bound [59], the number of triangles with conflict size at most s is $O(ns^2)$, so the expected number of triangles created in Step 5 is $O\left(\sum_{s=0}^{\infty} ns^2/2^s\right) \leq dn$, for some constant d. Since the expected size of S is $n/2$, the total expected structural change is therefore $cn/2 + dn \leq cn$, for c large enough, which completes the induction. Note that this argument does not help in higher dimensions, because there the Clarkson-Shor bound gives $O(n^{\lfloor (d+1)/2 \rfloor} s^{\lceil (d+1)/2 \rceil})$ simplices with conflict size at most s, which might yield a bound that is much larger than the expected structural change of a RIC. □

[6] See Section 7.1.3 for more information about configuration spaces.

5.2 Delaunay triangulations

Let $P \subseteq \mathbb{R}^d$ be an n-point set whose coordinates are w-bit words. The *shuffle-order* of P is obtained by taking `shuffle`(p) for every $p \in P$, as described above, and sorting the resulting numbers in the usual order. The shuffle order is also known as the Morton-order [118] or the Z-order, and it is intimately related to quadtrees [27, 39], see Figure 5.4.

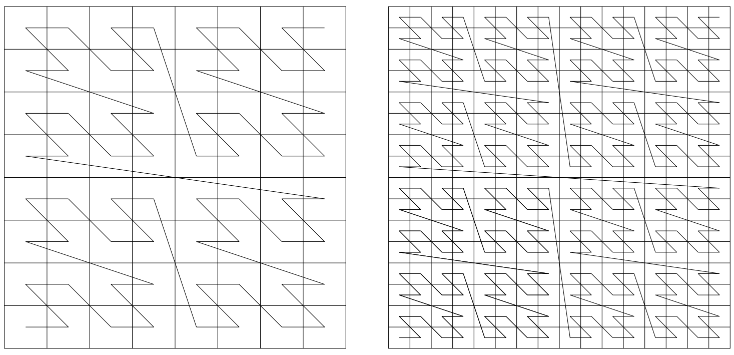


Figure 5.4: The shuffle-order corresponds to the inorder traversal of a quadtree.

Lemma 5.2.1. *Suppose our computational model is a word RAM. Suppose that $P \subseteq \{0, \dots, 2^w - 1\}^d$ is given in shuffle-order. Then, a compressed quadtree for P can be computed in $O(|P|)$ time.*

Proof. Our argument mostly follows Chan's presentation [39, Step 2]. We define a hierarchy H of quadtree-boxes, by taking the hypercube $\{0, \dots, 2^w - 1\}^d$ as the root box and by letting the children of a box b be the hypercubes that divide b into 2^d equal axis-parallel parts. For two points p, q, let `box`(p, q) be the smallest quadtree box that contains p and q, and let $|$`box`$(p, q)|$ be the side-length of this box. Both can be found by examining the most significant bits in which the coordinates of p and q differ. A *compressed quadtree* for a point set P is the subtree of H induced by the leaves in H that correspond to P. The crucial observation that connects compressed quadtrees with the shuffle order is that if the children of each node in H are ordered lexicographically, then the leaves of H are sorted according to the shuffle order. The quadtree is constructed by `BuildQuadTree`. Please see Algorithm 5.2. The algorithm is similar to the construction of *Cartesian trees* [83]. Assuming our model supports the `msb` (most significant bit) operation[7] in constant time, the algorithm runs in linear time: the total number of iterations in Step 2a is

[7]The operation `msb` returns the index of the first nonzero bit in a given word, and -1 if the word is 0.

Algorithm 5.2 Building a compressed quadtree.

```
BuildQuadTree

  1. $q_0$.box $= \{0, \dots, 2^w - 1\}^d$, $q_0$.children $= (p_1)$, $k = 0$

  2. for $i = 2, \dots, n$

     (a) while $|\texttt{box}(p_{i-1}, p_i)| > |q_k.\text{box}|$ do $k = k - 1$

     (b) if $|q_k.\text{box}| = |\texttt{box}(p_{i-1}, p_i)|$, let $p_i$ be the next child of $q_k$; otherwise, create
         $q_{k+1}$ with $q_{k+1}.\text{box} = \texttt{box}(p_{i-1}, p_i)$, and move the last child of $q_k$ to the
         first child of $q_{k+1}$, make $p_i$ the second child of $q_{k+1}$, and $q_{k+1}$ the last
         child of $q_k$. Set $k = k + 1$.
```

bounded by the number of times k is decremented, which is clearly at most n, and the box sizes can be computed in constant time.

With some more effort, we can avoid the `msb`-operation. First, as observed by Chan [37], note that box-sizes can be compared without the need for `msb`, because for two binary numbers x, y, we have

$$\texttt{msb}(x) < \texttt{msb}(y) \iff (x \leq y) \wedge (x < x \oplus y), \tag{5.4}$$

where $x \oplus y$ denotes the bitwise `xor`-operation. Thus, `BuildQuadTree` can find the combinatorial structure of the compressed quadtree T for P in linear time. Using a postorder traversal of T, we can find for each node b in T a minimum bounding box for the points under b, again in linear time. This information suffices to apply Lemma 5.2.2, as we can see by inspecting the proof of Callahan and Kosaraju [35]. $\square$

Via well-separated pair decompositions, we can go from quadtrees to NNGs in linear time as was shown by Callahan and Kosaraju [35] and by Clarkson [54] (see also Chan [39]). Their result is stated as follows:

Lemma 5.2.2. *Let $P \subseteq \mathbb{R}^d$. Given a compressed quadtree for P, we can find $\mathrm{NN}(P)$ in $O(|P|)$ time in a traditional model (and also on a word RAM).* $\square$

Combining Theorem 5.1.1 with Lemma 5.2.2, we establish a connection between quadtrees and Delaunay triangulations.

Theorem 5.2.3. *Let $P \subseteq \mathbb{R}^d$. Given a quadtree for P, we can find $\mathrm{DT}(P)$ in expected time $O(n + C(P))$, where $C(P)$ denotes the structural change of a RIC on P.*

Proof. `BrioDC` from Theorem 5.1.1 generates a sequence of samples

$$P = S_0 \supseteq S_1 \supseteq \cdots \supseteq S_\ell,$$

and we need to find the nearest-neighbor graphs $\mathrm{NN}(S_0), \dots, \mathrm{NN}(S_\ell)$ in total $O(n)$ time. By Lemma 5.2.2, and the fact that $\mathbf{E}\left[\sum_i |S_i|\right] \ll n$, it suffices to have quadtrees for $S_0, \dots, S_\ell$. This can be achieved in total linear time as follows: start with the quadtree for $P = S_0$ and in each round i remove the points from $S_{i-1} \setminus S_i$ and prune the tree. □

Theorem 5.2.3 immediately gives us an improved transdichotomous algorithm for Delaunay triangulations.

Theorem 5.2.4. *Suppose our computational model is a word RAM with a constant-time shuffle operation. Let $P \subseteq \{0, \dots, 2^w - 1\}^d$ be an n-point set. Then $\mathrm{DT}(P)$ can be computed in expected time $O(\mathtt{sort}(n) + C(P))$, where $C(P)$ denotes the expected structural change of a RIC on P and $\mathtt{sort}(n)$ denotes the time needed for sorting n numbers.*

Proof. First use Lemma 5.2.1 to build a compressed quadtree for P in $O(\mathtt{sort}(n))$ time, and then apply Theorem 5.2.3, see Figure 5.5. □

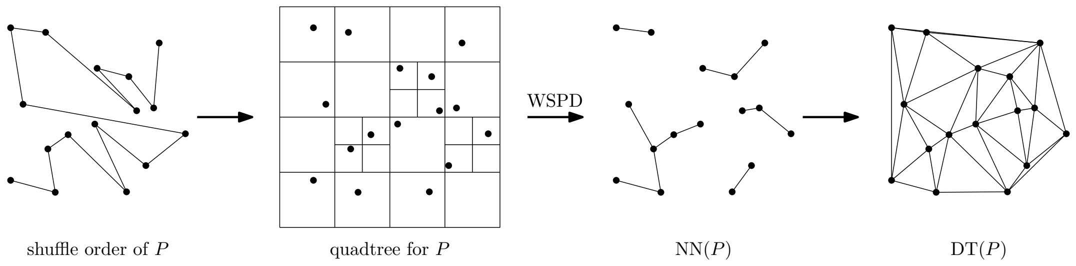


Figure 5.5: An illustration of Theorem 5.2.4: from shuffle sorting to quadtree to well-separated pair decomposition to nearest-neighbor graph to Delaunay triangulation.

Remark. For planar point sets, $C(P)$ is always linear, and this also often holds in higher dimensions. Furthermore, in the plane there is another approach to Theorem 5.2.4, which we sketch here: we sort P in shuffle order and compute a quadtree for P using Lemma 5.2.1. Then we use the techniques of Bern et al. [26, 27] to find a point set $P' \supseteq P$ and $\mathrm{DT}(P')$ in $O(n)$ time, where $|P'| \ll n$. Finally, we extract $\mathrm{DT}(P)$ using Theorem 3.1.1.

Our reduction has a curious consequence about presorted point sets, since we can find quadtrees for such point sets by an algebraic computation tree [14, Chapter 14] of linear depth.

Theorem 5.2.5. *Let $P \subseteq \mathbb{R}^d$ be an n-point set, such that the order of P along each coordinate axis is known. Then $\mathrm{DT}(P)$ can be computed by an algebraic computation tree with expected depth $O(n + C(P))$, where $C(P)$ denotes the expected structural change of a RIC on P.*

Proof. Build the quadtree in the standard way (as described, eg, in [26,35], and also in Section 6.3) but use simultaneous exponential searches from both sides when partitioning the points in each box. In each step of the search, we compare a coordinate of an input point with the average of the corresponding coordinates of two other input points. Then, we see that the number of such comparisons obeys a recursion of the type $T(n) = O(\log(\min(n_1, n_2)) + T(n_1) + T(n_2)$, with $n_1, n_2 \leq n-1$ and $n_1 + n_2 = n$, which solves to $T(n) \ll n$. This recursion holds only for nodes in which we are making progress in splitting the point set, but in all other nodes we perform only constantly many comparisons and there are linearly many of them. Nonetheless, the algorithm still needs $\Omega(n \log n)$ time, since we must split both the x- and the y-lists while building the tree. $\square$

As mentioned in Chapter 1, van Emde Boas trees [71,72] give us a way to preprocess a large universe so that its subsets can be sorted more quickly, and in Section 3.3, we saw that an analogous result for convex hulls is possible. We will now derive an improved result for the special case of planar Delaunay triangulations.

Theorem 5.2.6. *Let $U \subseteq \mathbb{R}^d$ be a u-point set. In $O(u \log u)$ time we can preprocess U into a data structure for the following kind of queries: given $P \subseteq U$ with n points, compute $\mathrm{DT}(P)$. The time to answer a query is $O(n \log\log u + C(P))$. As in Section 3.3, the algorithm runs on a traditional pointer machine.*

Proof. Compute a compressed quadtree T for U in time $O(u \log u)$, as described in Section 6.3. We use T in order to find NNGs quickly. Let $S \subseteq U$ be a subset of size m. The *induced subtree* for S, T_S, is the union of all paths from the root of T to a leaf in S. It can be found in time $O(m \log\log u)$.

Claim 5.2.7. *We can preprocess T into a data structure of size $O(u \log\log u)$ such that for any subset $S \subseteq U$ of m points we can compute the induced subtree T_S in time $O(m \log\log u)$.*

Proof. Build a vEB tree [72, 114] A for U, and preprocess T into a pointer-based data structure for least-common-ancestor (lca) queries[8] [109]. Furthermore, for each node of T compute its depth, and build a vEB priority queue B for the depths in T. These data structures need $O(u \log\log u)$ space. Given S, use the vEB tree A to sort S according to the order of T. Then use the lca-structure to compute T_S as follows:

[8]Given a rooted tree T and two nodes $u, v \in T$, the *least common ancestor* of u and v is the last node that appears both on the path from the root to u and on the path from the root to v.

initialize a linked list Q with S in sorted order. Insert the elements in S into vEB tree B, using the depth of the corresponding leaves as the key. Remove from B an element v with maximum depth, use Q to find v's two neighbors, and perform two lca-queries, one on v and its left neighbor, one on v and its right neighbor. Replace v in Q and B by the lower of the two ancestors (or delete v, if the ancestor is already present). Since there are $O(m)$ queries to the vEB trees, and $O(m)$ queries to the lca-structure, the whole process takes time $O(m \log \log u)$, as claimed. □

In order to find $\mathrm{NN}(S)$, use Claim 5.2.7 to compute T_S and then use Theorem 5.2.3. This takes $O(m \log \log u)$ time, and now the claim follows from Theorem 5.1.1. □

Remark. As is well known [23, 31, 122, 128], once we have computed the DT, we can find many other important geometric structures in $O(n)$ time, for example the Voronoi diagram, the Euclidean minimum spanning tree, or the Gabriel graph. Given the Voronoi diagram of a planar point set P, we can also find the largest empty circle inside $\operatorname{conv} P$ in linear time [128].

5.3 Shuffle-sorting on a word RAM

Finally, we will consider how to sort a point set $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$ according to the shuffle order in expected time $O(n \log \log n)$ on a standard word RAM, without using the shuffle operation. For this, we adapt a classic sorting algorithm by Andersson et al. [11]. This algorithm consists of two parts: (i) *range reduction*, which reduces the number of bits needed to represent the coordinates; and (ii) *packed sorting*, which packs many points into one word to speed up sorting. We describe how to adapt each of these steps and how to obtain the sorting algorithm from them. Let $w \geq \log n$ be the word size and b the number of bits required to represent the coordinates of the p_i (ie, the coordinates of the p_i are in the range $\{0, \dots, 2^b - 1\}$). We assume that w and b are known.

5.3.1 Packed sorting for large words

The following theorem is a simple extension of a result by Albers and Hagerup [5] and shows that we can sort in linear time if many points fit into one word.

Theorem 5.3.1. *Suppose that $b \leq \lceil w/(d \log n \log \log n) \rceil - 1$. Then P can be sorted according to the shuffle order in $O(n)$ time.*

Proof. By assumption, we can store $2k = w/(db+1) \geq \log n \log \log n$ *fields* in one word, where each field consists of a point preceded by a *testbit*. Given a word a, we let $a[1], \dots, a[2k]$ denote the $2k$ fields in a, and $a[i].t, a[i].d$ represent the test bit and data stored in field i. The main ingredient is a procedure for merging two sorted words in logarithmic time [5, Section 3].

Claim 5.3.2. *Given two words w_1, w_2 containing two sorted sequences $(p_i), (q_i)$ of k points each, we can compute a word w' containing the merged sequence (z_i) in time $O(\log k)$.*

Proof. The proof relies on a parallel implementation of a bitonic sorting network by Batcher [21]. We need to solve the following problem: given two words a and b, each containing a sequence of k points, compute in constant time a word z such that[9] $z[i].t = [a[i].d <_\sigma b[i].d]$ for $i = 1, \ldots, k$, ie, the testbit $z[i].t$ indicates whether the point in $a[i]$ precedes the point in $b[i]$ in the shuffle order.[10] We solve this problem with an algorithm `BatchCompare` that is based on a technique by Chan [37]. Refer to Algorithm 5.3. For each $i = 1, \ldots, k$, `BatchCompare` identifies the smallest coordinate $h[i]$ that has maximum $\mathtt{msb}(c_{h[i]}[i])$. Then, it determines whether $p_i <_\sigma q_i$ by comparing their coordinate $h[i]$. Correctness follows immediately from the definition of the shuffle order. Again, for an efficient implementation we use (5.4) to compare the positions of the most significant bits. `BatchCompare` runs in constant time, assuming that the constants $(1, \ldots, 1)$, ..., $(d, \ldots, d)$ have been precomputed, which can be done in $O(\log k)$ time. In particular, note that Step 6 takes constant time since there are only constantly many possible indices $h[i]$. □

Given Claim 5.3.2, the theorem now follows by an application of merge sort, see Albers and Hagerup [5] for details. □

5.3.2 Range reduction

In order to pack several points into a word, we need to adapt a range reduction technique due to Kirkpatrick and Reisch [98].

Theorem 5.3.3. *With expected $O(n)$ time overhead, the problem of shuffle-sorting n points with b-bit coordinates can be reduced to the problem of sorting n points with $b/2$-bit coordinates. The space needed for the reduction is $O(n)$.*

Proof. The proof follows the paper by Kirkpatrick and Reisch [98, Section 4]: we bucket the points according to the upper $b/2$ bits of their coordinates. Using universal hashing, this can be done in $O(n)$ expected time with $O(n)$ space. From each nonempty bucket b, we select the maximum element it contains, m_b. We truncate the coordinates of each m_b to the upper $b/2$ bits, and store a flag `isMaximum` in the satellite data for m_b. The coordinates of the remaining points are truncated to the lower $b/2$ bits, and the number of their corresponding bucket is stored in the

[9]We use Iverson's notation: $[X] = 1$ if X is true and $[X] = 0$, otherwise.

[10]We denote the shuffle order by $<_\sigma$.

Algorithm 5.3 Comparing many points simultaneously.

```
BatchCompare

 1. Create $d$ copies $a_1, \dots, a_d$ and $b_1, \dots, b_d$ of $a$ and $b$.
 2. Shift and mask the words $a_j, b_j$ so that $a_j[i].d = p_{ij}$ and $b_j[i].d = q_{ij}$ for
    $i = 1, \dots, k$, where $p_{ij}$ and $q_{ij}$ denote the $j$th coordinates of $p_i$ and $q_i$.
 3. Create $d$ words $c_1, \dots, c_d$ with $c_j[i].d = a_j[i].d \oplus b_j[i].d$ for $i = 1, \dots, k$, where
    $\oplus$ denotes bitwise xor.
 4. Create words $g, h$ such that $g = c_1$ and $h = (1, \dots, 1)$, ie, the word with a 1
    in the data item in each of its $k$ fields.
 5. For $j = 2, \dots, d$:
    (a) Set the testbits in $c_j$ such that $c_j[i].t = [g[i].d \leq c_j[i].d]$ and compute a
        mask $M$ for the fields $i$ with $c_j[i].t = 1$.
    (b) Let $c_j = c_j \oplus (g \texttt{ and } M)$.
    (c) Set the testbits in $c_j$ so that $c_j[i].t = [g[i].d < c_j[i].d]$ and compute a
        mask $M'$ for the fields $i$ with $c_j[i].t = 1$. Let $M = M \texttt{ and } M'$.
    (d) Set
            $g = (g \texttt{ and } \overline{M}) \texttt{ or } ((a_j \oplus b_j) \texttt{ and } M)$,
        where $\overline{M}$ is the bitwise negation of $M$. Furthermore, set
            $h = (h \texttt{ and } \overline{M}) \texttt{ or } ((j, \dots, j) \texttt{ and } M)$.
 6. Create a word $z$ with $z[i].t = [a_{h[i]}[i] < b_{h[i]}[i]]$.
```

satellite data. After the resulting point set has been sorted, we use the satellite data to establish (i) the ordering of the buckets, by using the sorted maxima, and (ii) the ordering within each bucket. The crucial fact needed for correctness is that for any two points p, q, we have $p <_\sigma q$ precisely if $(p' <_\sigma q') \vee (p' = q' \wedge p'' <_\sigma q'')$, where (p', p''), (q', q'') are derived from p, q by splitting each of their coordinates into two blocks of $b/2$ bits. □

5.3.3 Putting it together

Following Andersson et al. [11], we combine the algorithms in Sections 5.3.1 and 5.3.2 to obtain a simple randomized $O(n \log \log n)$ sorting algorithm.

Theorem 5.3.4. *Given a set P of n points with b-bit integer coordinates, we can sort P in expected time $O(n \log \log n)$ with $O(n)$ space.*

Proof. Iterate Theorem 5.3.3 $O(\log \log n)$ times until $O(\log n \log \log n)$ points fit into one word. Then apply Theorem 5.3.1. The total space for the range reduction is $O(n)$ because in each step the number of bits for the satellite data is halved. $\square$

Chapter 6

Restricted Inputs

The third and last kind of structural information we consider are *restricted inputs.* Suppose we are given a set $\mathcal{R}$ of planar regions, and we want to preprocess $\mathcal{R}$ such that we can answer queries of the following kind: given a point set P with exactly one point from each region in $\mathcal{R}$, find $\mathrm{DT}(P)$. We call such queries *Delaunay queries.* Löffler and Snoeyink [110] showed that if $\mathcal{R}$ consists of disjoint unit disks, then there is a linear-size data structure that can answer Delaunay queries in time $O(n)$.

The algorithm by Löffler and Snoeyink is deterministic and optimal, but it is based on linear-time polygon triangulation [44] which would make an actual implementation quite challenging. Furthermore, it does not adapt well to more general input regions $\mathcal{R}$, and scales badly when the disks in $\mathcal{R}$ can overlap or are not of the same size. We will show that these shortcomings can be fixed as long as we are satisfied with randomized algorithms: (i) the algorithm by Löffler and Snoeyink [110] can be simplified considerably using randomization; (ii) the same result can be obtained for disjoint disks of different sizes; and (iii) we get a better dependence on the realism parameters. In particular, let k be the depth of the arrangement of $\mathcal{R}$. We can preprocess $\mathcal{R}$ in $O(n \log n)$ time into a data structure of $O(n)$ size that handles imprecise Delaunay queries in $O(n \log k)$ time. For comparison, the previous bound is $O(nk)$ [110].

Our approach is this: given $\mathcal{R}$, we find a quadtree T such that every cell of T meets a bounded number of regions in $\mathcal{R}$. To answer a query P, we locate the points of P in the cells of T, turn T into a quadtree T' for P, and use T' to compute $\mathrm{DT}(P)$ quickly. For that, we need the connection between quadtrees and Delaunay triangulations established in Theorem 5.2.3 in Chapter 5. In fact, owing to technical reasons we will need a slight generalization of Theorem 5.2.3, and we will say more about it in Section 6.4.

Previous work and relation to data imprecision. The original motivation for the restricted inputs comes from the investigation of *imprecise input models.* What are these all about? Typically, the input to a computational geometry problem is a point set P in the plane, or more generally $\mathbb{R}^d$. Traditionally, one assumes that P is known exactly, and indeed, in the 1980s and 1990s this was often justified, as much of the input data was hand-constructed for computer graphics or simulations. Nowadays, however, the input is often sensed from the real world, and thus inherently imprecise.

An early model for imprecise geometric data, motivated by finite precision of coordinates, is *ε-geometry* [85]. Here, the input is a traditional point set P and a parameter ε. The true point set is unknown, but each point is guaranteed to lie in a disk of radius ε. Even though this model has proven fruitful and remains popular due to its simplicity [18, 86], it may often be too restrictive: imprecision regions could be more complicated than disks, and their shapes and sizes may even differ from point to point, eg, to model imprecision from different sources, independent imprecision in different input dimensions, etc. In these settings, the extra freedom in modeling leads to more involved algorithms, but still many results are known [105, 107, 124, 125].

The above results assume that the imprecise input is given once and simply has to be dealt with. While this holds in many applications, it is also often possible to get (more) precise estimates of the points, but they will only become available later when there is less time, or they come at a higher cost. For example, in the *update complexity* model [33, 78], each data point is given imprecisely at the beginning but can always be found precisely at a certain price.

This leads us to the restricted input model: the input regions $\mathcal{R}$ represent an imprecise point set which we want to preprocess so that some structure can be computed faster when the exact points become available later.

As mentioned above, this model was considered by Löffler and Snoeyink [110] who obtained a data structure that supports linear-time Delaunay queries for disjoint unit disks. A weaker result is due to Held and Mitchell [89] who could preprocess a set of disjoint unit disks so that it takes linear time to find a (not necessarily Delaunay) triangulation of a point set with one point from each disk. This was extended by van Kreveld, Löffler and Mitchell [106], who supposed that $\mathcal{R}$ consists of n disjoint polygons with a total of m vertices and then obtained an $O(m)$-space data structure with $O(m \log m)$ preprocessing time and $O(m)$ time for finding a (not necessarily Delaunay) triangulation of the query. There is no restriction on the shapes and sizes of the individual regions (they do not even strictly have to be polygonal), only on the overlap. All the above algorithms are deterministic, and, with the exception of Löffler and Snoeyink's result, relatively simple.

6.1 Disks of varying sizes: quadtree-approach

Firstly, we will extend the theorem by Löffler and Snoeyink to disks of varying sizes by using quadtrees. The main idea is to find a quadtree T such that each cell of T meets a bounded number of regions of $\mathcal{R}$. To answer a Delaunay query P we locate the points of P in T, compute a quadtree T' for P, and then apply Theorem 5.2.3. With the additional structure of the quadtree we can handle disks of varying sizes.

Let us first define a *free quadtree* T. It is an ordered rooted tree that corresponds to a hierarchical decomposition of the plane into axis-aligned square *boxes*. Each node v of T has a square box B_v associated to it, according to the following rules:

1. If w is a descendant of v in T, then $B_w \subseteq B_v$.

2. If v and w are not related, then $B_v \cap B_w = \emptyset$.

The boxes B_v constitute a *laminar family* of squares in $\mathbb{R}^2$. We say the *size* of a node is the side length of its box. With each node v, we associate the *cell* C_v that consists of the part of B_v that is not covered by the children of v.[1] Any two distinct cells C_v and C_w are disjoint, and the union of the cells of all nodes of T covers the root square.

A standard quadtree [75] is a special case of a free quadtree, and in particular has only two types of nodes: *internal nodes* v with exactly four children half the size of v, and *leaf nodes* without any children. In this section we define a (compressed) quadtree as a standard quadtree, but with the addition of a third type of node, which we call *cluster node*: a node v with just one child w, whose size is smaller than its parent by at least a large constant factor 2^c. We require that the horizontal and vertical distances between the boundary B_w and the boundary of B_v are either zero or at least the size of B_w. Figure 6.1a shows an example of a quadtree of this type. Cluster nodes ensure that the complexity of T can be kept linear [26, Section 3.2]. Given a planar point set P, we say that T is a quadtree *for* P if the following properties hold:

1. For each leaf $v \in T$, we have $|P \cap C_v| \leq 1$, ie, C_v has at most one point of P inside it.

2. For all non-leaves $v \in T$, we have $P \cap C_v = \emptyset$.

3. The root box of T contains all of P.

4. The total number of nodes in T is $O(|P|)$.

[1]Note that C_v could be disconnected or empty.

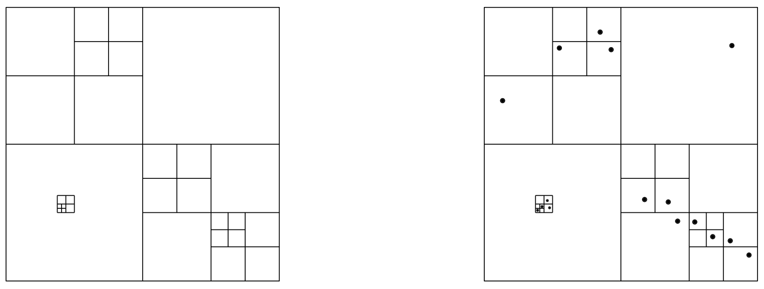

(a) (b)

Figure 6.1: (a) A quadtree. The lower left box contains a cluster node. (b) The quadtree is a valid quadtree for this set of points.

Figure 6.1b shows a point set P and a valid quadtree for it. To apply Theorem 5.2.3 we need to preprocess $\mathcal{R}$ into a quadtree T such that any cell C_v in T intersects only constantly many disks. While we could consider the disks directly, we will instead use a quadtree T for a point set Q representing the disks. For each disk we include in Q its center and top-, bottom-, left- and rightmost point. Then, T can be constructed in $O(n \log n)$ time, see Section 6.3.

Lemma 6.1.1. *Every cell C_v of T is intersected by $O(1)$ disks in $\mathcal{R}$.*

Proof. There are three types of nodes to consider. First, if v is an internal node with four children, then C_v is empty so the condition holds trivially. Next, suppose that v is a leaf node, so $C_v = B_v$. If a disk D intersects B_v and does not contain a corner of B_v, then B_v must either contain D's center or one of its four extreme points [24]. Thus, B_v intersects at most 5 disks, one for each corner and one for the point of Q it contains. See Figure 6.2a for an example.

Now suppose v is a cluster node, with child w. Then $C_v = B_v \setminus B_w$. We know there are no points of Q in C_v, and there are at most four disks that have all their representative points outside B_v. So it remains to count the disks that intersect B_v, do not cover a corner of B_v, and have an extreme point or their center in B_w. For this, consider the at most four orthogonal neighbors of B_w in B_v (i.e., copies of B_w directly to the left, to the right, above and below B_w) that lie inside B_v. Using the same argument as above, each of these neighbors meets at most four disks, and every disk D with an extreme point or center in B_w that intersects C_v also meets one of the orthogonal neighbors (if D has no extreme point or center in an orthogonal neighbor and does not cover any of its corners, it has to cover its center), which implies the claim, because by our assumption about cluster nodes all the orthogonal neighbors are completely contained in B_v. Figure 6.2b shows an example involving a cluster node. $\square$

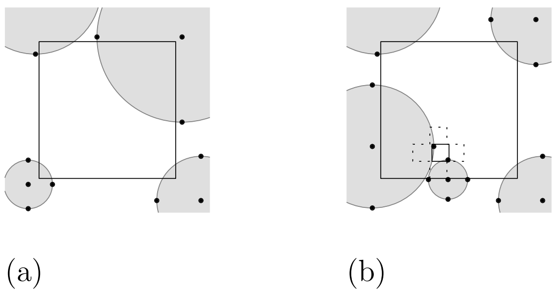


Figure 6.2: (a) At most 4 disjoint disks can intersect any given box B belonging to a leaf of the quadtree, without one of their points being inside B. (b) For a box B belonging to a parent of a cluster node with box C, slightly more disks can intersect the interior of $B \setminus C$, but not more than 4 can cover any of the four neighboring boxes, so a crude upper bound is 20.

Theorem 6.1.2. *Let $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$ be a sequence of disjoint planar disks (of possibly different size). In $O(n \log n)$ time and using $O(n)$ space we can preprocess $\mathcal{R}$ into a data structure that can answer imprecise Delaunay queries in $O(n)$ expected time.*

Proof. We construct Q and the quadtree T for Q as described above. For each R_i we store a list with the leaves in T that intersect it. By Lemma 6.1.1, the total size of these lists, and hence the complexity of the data structure, is linear. Now we describe how queries are handled: let $P = \langle p_1, \ldots, p_n \rangle$ be the input sequence. For each p_i, we find the node v of T such that $p_i \in C_v$ by traversing the list for R_i. This takes linear time. Since each leaf of T contains constantly many input points, we can turn T into a quadtree for P in linear time. We now compute DT(P) via Theorem 5.2.3.[2] $\square$

6.2 Overlapping disks: deflated quadtrees

We extend the approach to disks with limited overlap. Now $\mathcal{R}$ contains n planar disks such that no point is covered by more than k disks. The parameter k is called the *depth* of $\mathcal{R}$, and Aronov and Har-Peled [13] showed that k can be approximated up to a constant factor in $O(n \log n)$ time. It is easily seen that imprecise Delaunay queries take $\Omega(n \log k)$ time in the worst case, and we show that this bound can be achieved.

[2]We are ignoring some technical details here, because when turning T into a quadtree for P we need to be careful about handling cluster nodes that contain an input point. We will explain this in the next section, where we consider a more general setting.

Proposition 6.2.1. *For $k \in \{1, \dots, n\}$, there exists a set $\mathcal{R}$ of n planar unit disks with depth k such that Delaunay queries for $\mathcal{R}$ need $\Omega(n \log k)$ time in the algebraic decision tree model.*

Proof. Consider the problem k-$(1/k)$-Closeness: we are given n/k sequences $\mathbf{x}_1, \dots, \mathbf{x}_{n/k}$, each with k real numbers in $(-1, 1)$, and we would like to decide whether any $\mathbf{x}_i$ contains two numbers with difference at most $1/k$. To see that any algebraic decision tree for k-$(1/k)$-Closeness has depth $\Omega(n \log k)$, let $W' \subseteq \mathbb{R}^n$ be defined as

$$W' = \left\{(\mathbf{x}_1, \dots, \mathbf{x}_{n/k}) \mid |x_{ij} - x_{il}| > 1/k \text{ for } 1 \le i \le n/k; 1 \le j < l \le k\right\},$$

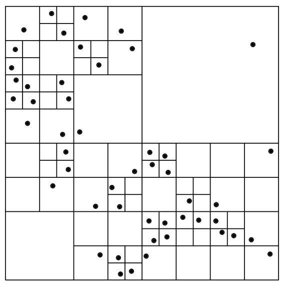
where x_{ij} is the jth coordinate of $\mathbf{x}_i$, and let $W = W' \cap (-1, 1)^n$. Since W has at least $(k!)^{n/k}$ connected components, Ben-Or's lower bound [22, Theorem 5] implies the claim.

Now let $\mathcal{R}$ consist of the unit disks with centers $(3i, 9i^2)$, for $i = 1, \dots, n/k$, each copied k times.[3] Let $\mathbf{x}_1, \dots, \mathbf{x}_{n/k}$ be an instance of k-$(1/k)$-Closeness. Create a point set P by mapping the input x_{ij} to $(3i + x_{ij}/12i, (3i + (x_{ij}/12i))^2)$. Note that P can be found in $O(n)$ time, and that it contains exactly one point from each circle in $\mathcal{R}$. Now, if we know $\mathrm{DT}(P)$, we can in linear time find the sorted order of each $\mathbf{x}_i$, and hence solve k-$(1/k)$-Closeness. Thus, we have a reduction from k-$(1/k)$-Closeness to Delaunay queries with regions of depth k, and the lower bound follows. □

The general strategy for the upper bound is the same as in Section 6.1. Let Q be the $5n$ representative points for $\mathcal{R}$, and let T be a quadtree for Q. As before, T can be found in time $O(n \log n)$ and has complexity $O(n)$. However, the cells of T can now be intersected by $O(k)$ regions, rather than $O(1)$. Since our data structure stores all the cell-disk incidences, this means that the space requirement would become $O(nk)$, which is too large. However, we can avoid this by reducing the complexity of T until it only has $O(n/k)$ cells, while maintaining the intersection property. Then, we share the more detailed structures between the regions in $\mathcal{R}$, thus saving space. For this we introduce the notion of λ-*deflated* quadtrees.

For a positive integer $\lambda \in \mathbb{N}$, a λ-deflated quadtree T' for a point set Q has the same general structure as the quadtrees from the previous section, but it has lower complexity: each node of T' can contain up to λ points of Q in its cell and there are only $O(n/\lambda)$ nodes. We distinguish four different types of nodes: (i) *leaves* are nodes v without children, with up to λ points in C_v; (ii) *internal nodes* v have four children of half their size covering their parent, and $C_v = \emptyset$; (iii) *cluster nodes* are, as before, nodes v with a single—much smaller—child, and with no points in C_v; (iv) finally, a *deflated node* v has only one child w—possibly much smaller than its

[3]We can perturb the centers of the copies slightly if we don't like $\mathcal{R}$ to be a multiset.


(a)

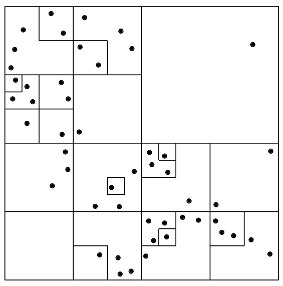

(b)

Figure 6.3: (a) A set of points and a quadtree for it. (b) A 3-deflated version of the quadtree.

Algorithm 6.1 Turning a quadtree into a λ-deflated quadtree.

```
Algorithm DeflateTree($v$)

1. If $n_v \leq \lambda$, return the tree consisting of $v$.
2. Let $T_v$ be the subtree rooted in $v$, and let $z$ be a node in $T_v$ with the smallest
   value $n_z$ such that $n_z > n_v - \lambda$. Note that $z$ could be $v$.
3. For all children $w$ of $z$, let $T'_w$ = DeflateTree($w$).
4. Build a tree $T'_v$ by picking $v$ as the root, $z$ as the only child of $v$, and linking
   the trees $T'_w$ to $z$. If $v \neq z$, then $v$ is a deflated node. Return $T'_v$ as the result.
```

parent—and additionally C_v may contain up to λ points. Cluster nodes and deflated nodes are very similar, but they play slightly different roles in the Delaunay query algorithm. An example of a quadtree and a 3-deflated version of it is shown in Figure 6.3.

Given a quadtree T for Q, a λ-deflated quadtree T' can be found in linear time: for every node v in T, compute $n_v = |B_v \cap Q|$. This takes $O(n)$ time with a postorder traversal. Then, T' is obtained by applying `DeflateTree` to the root of T. Please see Algorithm 6.1. Since `DeflateTree` performs a simple top-down traversal of T, it takes $O(n)$ time.

Lemma 6.2.2. *A λ-deflated quadtree T' produced by Algorithm 6.1 has $O(n/\lambda)$ nodes.*

Proof. Let T'' be the subtree of T' that contains all nodes v with $n_v > \lambda$, and suppose that every cluster node in T'' has been contracted with its child. We will show that T'' has $O(n/\lambda)$ nodes, which implies the claim, since no two cluster nodes are adjacent, and because all the non-cluster nodes in T' which are not in T'' must be leaves. We count the nodes in T'' as follows: (i) since the leaves of T'' correspond to disjoint subsets of Q of size at least λ, there are at most n/λ of them; (ii) the bound on the leaves also implies that T'' contains at most n/λ nodes with at least 2 children; (iii) the number of nodes in T'' with a single child that has at least 2 children is likewise bounded; (iv) when an internal node v has a single child w that also has only a single child, then by construction v and w together must contain at least λ points in their cells, otherwise they would not have been two separate nodes. Thus, we can charge $\lambda/2$ points from Q to v, and the total number of such nodes is $2n/\lambda$. □

Lemma 6.2.3. *Let T' be a k-deflated quadtree for Q. Every cell C_v of T' is intersected by $O(k)$ disks of $\mathcal{R}$.*

Proof. Treat deflated nodes like cluster nodes and note that the center and corners of every box of T' can be covered by at most k disks. Now the lemma follows from the same arguments as we used in the proof of Lemma 6.1.1. □

Theorem 6.2.4. *Let $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$ be a sequence of planar disks such that no point is covered by more than k disks. In $O(n \log n)$ time and using $O(n)$ space we can preprocess $\mathcal{R}$ into a data structure that can answer imprecise Delaunay queries in $O(n \log k)$ expected time.*

Proof. It remains to show how to preprocess T' to handle the imprecise Delaunay queries in time $O(n \log k)$. By Lemmas 6.2.2 and 6.2.3, the total number of disk-cell incidences in T' is $O(n)$. Thus, in $O(n)$ total time we can find for each $R \in \mathcal{R}$ the list of nodes in T' whose cells it intersects. Next, we determine for each node v in T' the portion X_v of the original quadtree T inside the cell C_v and build a point location data structure for X_v. Since X_v is a partial quadtree for at most k points, it has complexity $O(k)$, and since the X_v are disjoint, the total space requirement and construction time are linear. This finishes the preprocessing.

To handle an imprecise Delaunay query, we first locate the input points P in the cells of T' just as in Theorem 6.1.2. This takes $O(n)$ time. Then we use the point location structures for the X_v to locate P in T in total time $O(n \log k)$. Now we turn T into a quadtree for P in time $O(n \log k)$, and find the Delaunay triangulation in time $O(n)$, as before.

We now explain how T is turned into a quadtree for P: for cells corresponding to leaf nodes, we can just use a standard algorithm for computing quadtrees, which takes $O(\alpha \log k)$ time for a cell that contains α points, since $\alpha = O(k)$ by Lemma 6.2.3. For cells that correspond to cluster nodes, we must work harder in

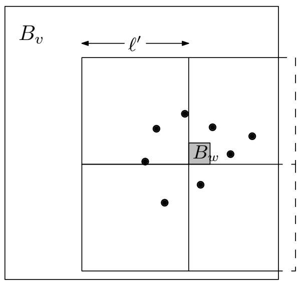


Figure 6.4: We align four boxes of side length ℓ' with B_w to obtain a bounding box for P' and B_w. Note that this box may intersect the boundary of B_v.

order to avoid the need for the floor function[4]: suppose v is a cluster node with child w, such that C_v contains α points P' from P. We sort P' according to x- and y-coordinates, and then determine a bounding box for B_w and P'. Let ℓ be the side length of this bounding box. Now we find in $O(\log k)$ time an integer $0 \leq \gamma \leq \alpha$, such that $|B_w| \leq 2^{-c\gamma}\ell$ and either $\gamma = \alpha$ or $|B_w| \geq 2^{-c\gamma-1}\ell$, where $|B_w|$ denotes the size of B_w. Then, we set $\ell' = 2^{c\gamma+1}|B_w|$ if $\gamma \neq \alpha$, and $\ell' = \ell$ otherwise. Align four boxes of side length ℓ' with B_w (see Figure 6.4) and use the result as the bounding box for the quadtree T'' that contains B_w and P'. This ensures that no edge of T'' intersects B_w, because all non-cluster nodes of T'' have size at least $2^{-c\alpha}\ell'$. If during the construction of T'' the cell C_w is again contained in a cluster node, we repeat the procedure (without the sorting step), which takes another $O(\log k)$ time. However, this cluster node will contain strictly less than α points from P' (because it is significantly smaller than the bounding box of T''), so the total cost for the bounding box computations cannot exceed $O(\alpha \log k)$. There is one subtlety: the bounding box of T'' might intersect the boundary of B_v. If this happens, we clip T'' to B_v. The resulting quadtree is *skewed*, which means that its cells can be shifted relative to their parent, and some of them may even be clipped. In Section 6.4 we argue that Theorem 5.2.3 still holds in this case. □

6.3 Computing compressed quadtrees in $O(n \log n)$ time

It is well known that given a planar n-point set P, we can compute a compressed quadtree T for P in $O(n \log n)$ time. However, the details of the construction are a bit involved [73], and for the reader's convenience we describe here a possible implementation that achieves the claimed running time on a pointer machine. The

[4]In Section 2.3 we explain why it is worthwhile to spend so much effort to avoid the floor function.

basic algorithm is as follows: sort the points in x- and y-direction, find a bounding box for P, and repeatedly split the boxes in the obvious way. If after c splits of the current box the size of the point set P' inside it has not decreased, we find a bounding box for P', create a cluster node and recursively compute a quadtree for P'.

In order to perform the splitting efficiently, we need to maintain the x- and y-orderings of the points contained in the current box. To do this, we do the splitting in two steps: first in x-direction, then in y-direction. To split in x-direction, we traverse the x-list simultaneously from both ends to find the split sets in time proportional to the size of the smaller set. Let P_1 and P_2 be the two resulting sets, and let n_1 and n_2 be their sizes, with $n_1 \leq n_2$. Using appropriate pointers, we find the points of P_1 in the y-list and remove them. Now we need to create the sorted y-list for P_1: if $n_1 \leq n_2^{1/3}$, we just sort P_1 according to y-coordinate in $O(n_1 \log n_1)$ time. Otherwise, we use a pointer-based radix sort for P_1 that takes $O(n_1)$ time. For this, we need to maintain an appropriate data structure with each of the y-lists.[5] This data structure is created when a y-list is split off and updated every time the size of a y-list halves, which leads to a linear time overhead. The total time needed for the splitting in x-direction obeys the recursion

$$T(n) = \begin{cases} T(n_1) + T(n_2) + O(n_1 \log n_1), & \text{if } n_1 \leq n_2^{1/3}, \\ T(n_1) + T(n_2) + O(n_1), & \text{if } n_2^{1/3} \leq n_1 \leq n_2. \end{cases}$$

This solves to $T(n) \ll n \log n$. The splitting in y-direction is done in the same way, and the total time needed for the construction of the quadtree is $O(n \log n)$, as claimed.

6.4 From quadtrees to Delaunay triangulations

For technical reasons described in Section 6.2, we need a slightly more general version of Theorem 5.2.3. Recall that the proof of Theorem 5.2.3 is based on a chain of reductions from Delaunay triangulations to nearest-neighbor graphs to well-separated pair decompositions to quadtrees: in Chapter 5, we saw that to compute $\mathrm{DT}(P)$, it suffices to find the nearest neighbor graphs $\mathrm{NN}(P_1), \ldots, \mathrm{NN}(P_t)$ for each level of an appropriate gradation $\emptyset = P_0 \subseteq P_1 \subseteq \cdots \subseteq P_t = P$ with $\sum_{i=0}^{t} |P_i| = O(n)$. By a classic reduction [35], $\mathrm{NN}(P_1), \ldots, \mathrm{NN}(P_t)$ can be found in linear time, once we know ε-well-separated pair decompositions (ε-WSPDs) for $P_1, \ldots, P_t$ of linear size and for appropriate ε. These ε-WSPDs can in turn be derived from compressed quadtrees for these sets. By assumption, we have a quadtree T

[5]We need a pointer structure that allows us to associate three-digit integers with the points according to their position in the y-list.

for $P = P_t$, and by successively pruning T, we can get quadtrees for $P_1, \dots, P_{t-1}$ in linear time.

Now we come to the adaptation of the proof. In Section 6.2, we defined a *skewed* quadtree as a standard compressed quadtree, but one where clusters can be shifted relative to their parents and parts of the cluster cells might be clipped. We need to prove that Theorem 5.2.3 still holds in the case where we are given a skewed quadtree for P, rather than a regular quadtree. Note that all steps outlined above can go through unchanged, except that now we need to go from skewed quadtrees to ε-WSPDs. For this, we take the algorithm that can compute an ε-WSPD based on a standard compressed quadtree [35,39] and check that it still works if the quadtree is skewed. We make a key observation about skewed quadtrees first.

Observation 6.4.1. *Let v be a node of a skewed quadtree T, and let d be the size its box would have without clipping. Then there is a cell B_w adjacent to B_v (possibly diagonally) such that the volume of B_w is at least cd^2 for some constant c.* $\square$

Also, recall the definition of an ε-WSPD for a point set P. It is a collection of pairs $\{(P_1, Q_1), \dots, (P_m, Q_m)\}$ with $P_i, Q_i \subseteq P$ such that each pair (P_i, Q_i) is *ε-well-separated*, which means that the diameters of P_i and Q_i are both at most ε times the minimum distance between P_i and Q_i. Furthermore, we require that every pair $(p, q) \in P \times P$ of distinct points is in $P_i \times Q_i$ or $Q_i \times P_i$ for exactly one i. We call m the *size* of the ε-WSPD.

Now we are ready to follow the argument. Let T be a skewed quadtree for P, and let us see how to find a well-separated pair decomposition for P of linear size in time $O(|P|)$, given T. Our presentation follows Chan [39], with some small changes to adapt the argument to skewed quadtrees. The WSPD is computed by performing the function `wspd`(v) on the root v of T. Refer to Algorithm 6.2.

Here, P_v denotes the points contained in B_v, the box corresponding to v, and $|B_v|$ is the size of B_v. Clearly, `wspd` computes an ε-WSPD for P. We need to argue that it has linear size and that the computation takes linear time. For this, it suffices to count the number of calls to `wspd`(v_1, v_2) such that B_{v_1} and B_{v_2} are not ε-well-separated. By induction, we see that whenever `wspd`(v_1, v_2) is called, the size of the box corresponding to the parent of v_1 is at least $|B_{v_2}|$ and the size of the parent box for v_2 is at least $|B_{v_1}|$. Without loss of generality, we assume $|B_{v_1}| \leq |B_{v_2}|$, and let r be the parent of v_1. As we noted above, we have $|B_r| \geq |B_{q_2}|$, and by successively subdividing B_r in a quadtree fashion (and shifting if necessary), we obtain a box B such that $B_{v_1} \subseteq B \subseteq B_r$, and such that $|B_{v_2}|/2 \leq |B| \leq |B_{v_2}|$. Since B and B_{v_2} are not ε-well-separated, B must be within distance $O(|B|/\varepsilon)$ of B_{v_2}. To see that there are only constantly many such boxes, we can use a volume argument, but we need to be careful about boxes which are clipped because they are contained in a cluster that is shifted relative to its parent. However, by Observation 6.4.1, every clipped cell has an adjacent cell which is only cut by a constant fraction c (if at all).

Algorithm 6.2 Finding a well-separated pair decomposition.

```
wspd($v$)
  1. If $v$ is a leaf, return $\emptyset$.
  2. Return the union of wspd($r$) and wspd($r_1, r_2$) for all children $r$ and pairs of
     distinct children $r_1, r_2$ of $v$.

wspd($v_1, v_2$)
  1. If $B_{v_1}$ and $B_{v_2}$ are $\varepsilon$-well-separated, return $(P_{v_1}, P_{v_2})$.
  2. Otherwise, if $|B_{v_1}| \leq |B_{v_2}|$, return the union of wspd($v_1, r$) for all children $r$
     of $v_2$.
  3. Otherwise, return the union of wspd($r, v_2$) for all children $r$ of $v_1$.
```

Therefore, the number of cells that are clipped too much can be at most 8 times the number of cells that are clipped by at most c, so the total number of nearby cells is still constant. Hence, the total size of the WSPD is linear.

Part II

The Role of Randomness

Chapter 7

Markov Incremental Constructions

So far, we have considered the effects of low entropy in the *inputs.* However, there is also another kind of entropy that plays a role in algorithm design, namely, the entropy of the randomness the algorithm uses for itself. In the previous chapters, we have already encountered several times the randomized incremental construction (RIC) paradigm, and we have seen the key features that make it so popular with computational geometers: (i) it is versatile; (ii) it is simple; and (iii) it often yields optimal randomized algorithms [23, 28–31, 48, 49, 56, 59, 64, 66, 84, 113, 115, 120–122, 135, 138–140, 148]. But what if the algorithm has no access to perfect randomness? In the worst case, the running time typically increases a factor of n. However, Mulmuley [123] proved that $O(1)$-wise independence is in fact sufficient. Furthermore, Amenta et al. [9], in their work on RICs with biased random insertion order which we already encountered in Chapter 5, showed that the entropy may slowly decay during the RIC without penalty; in other words, the insertion sequence can afford to be less and less random as the construction progresses. Devillers and Guigue [65] introduced the *shuffling buffer* which randomly permutes contiguous subsequences of the input sequence of a certain length k, and they provided trade-offs between the length k and the running time of the RIC. What these results demonstrate is that standard RIC analysis still works as long as there is sufficient *local* randomness *early* enough. Unfortunately, these two features are precisely what is lacking in *Markov sources.*

What are those? A Markov source is a probabilistic model of input data that serializes the production of data over time by means of a random walk in a graph. It is widely used in queuing theory, speech recognition, gesture modeling, protein homology, computer graphics, robotics, web searching, etc. It captures the statistical correlations created by *time coherence.* In speech, for example, the randomness of the next utterance is heavily dependent on the previous ones; hence the use of hidden Markov models. In geometric applications, Markov sources have been used in ray tracing [93,146], computer games [111], robotics [77], terrain generation [147], etc. In computer science, one of the main motivations has been *locality of reference*; in particular, there exists a vast body of research in online algorithms for Markov sources [42, 91, 94, 95, 104, 108, 127, 134, 142]. The work of Amenta et al. [9] on RICs is also motivated by the desire for local access. Therefore, it is natural to ask what happens to a general algorithmic paradigm (RIC) when one assumes a Markov source.[1]

Here, our model consists of an *event graph* $G = (V, E)$ which is connected and undirected. This means that G defines a Markov chain that is irreducible and reversible but not necessarily ergodic. Each node v is associated with an item x_v in a universe $\mathcal{U}$. Requests are specified by following a random walk, beginning at a random start node of G and hopping from node to node, each time choosing an adjacent node v uniformly at random. Upon reaching v, item x_v is inserted into the current structure. The structure in question depends, of course, on the application. Below, we will consider convex hulls, trapezoidal maps, and segment intersections. The structure is the corresponding *conflict graph*. Actually, we can use a data structure called *influence graph* [28–31, 66, 70] or *history graph* [122], which has the advantage of supporting queries and allowing for online (semi)dynamic algorithms. This means that we do not even need to know the graph G ahead of time. Our analysis, in fact, supports all known variants of RICs.

To obtain our result, we will need to extend Mulmuley's theory of Θ-*series* [122] to Markov chains[2]; and we will see a generalization of the classic Clarkson-Shor counting technique for Markov sampling as well as a new bound on the expected first passage time in a Markov chain with bounded spectral gap.

More concretely, we can bound the expected complexity of RIC for convex hulls in d dimensions by $O(\gamma^{-d} n^{\lfloor d/2 \rfloor} (\log n)^{\lceil d/2 \rceil})$ for $d > 3$, and $O(n(\gamma^{-1} \log n)^d)$ for $d \leq 3$, where γ is the spectral gap of the random walk, ie, the difference between the first and second largest eigenvalues of the transition matrix—note that γ is a positive constant in the case of a random graph or an expander. For trapezoidal maps of nonintersecting segments and segment intersections, the complexity is respectively $O(n(\gamma^{-1} \log n)^4)$ and $O((n+m)(\gamma^{-1} \log n)^6)$, where m is the number of intersections.

7.1 Background

Before we begin, let us review some background on Markov chains [117,119], spectral techniques [90] and configuration spaces [23, 122].

[1]See Section 7.1.1 for technical background on Markov Chains and our terminology.

[2]See Section 7.1.3 for more on configuration spaces and Θ-series

7.1.1 Markov chains

A Markov chain M over a finite *state space* Q is an infinite sequence of random variables $X_0, X_1, \dots$ with the following properties: (i) $X_t \in Q$ for $t \geq 0$; (ii) X_0 is drawn from a given initial distribution π_0 over Q; and (iii) there are $p_{qr} \in [0,1]$, $q, r \in Q$, such that $\Pr[X_{t+1} = q_{t+1} | X_0 = q_0, X_1 = q_1, \dots, X_t = q_t] = p_{q_t q_{t+1}}$ for $t \geq 0$, ie, the distribution of X_{t+1} depends only on X_t. The variable X_t is called the *state* at time t. The $|Q| \times |Q|$ matrix P formed by the p_{qr} is called the *transition matrix* of M. The distribution of X_t can be computed as $\pi_0^T P^t$. We say that M is *irreducible* if for any two states $q, r \in Q$ there exists $t \geq 0$ such that $\Pr[X_t = r \mid X_0 = q] > 0$, ie, every state can reach all other states with positive probability after a finite number of steps. A state $q \in Q$ is *periodic* if there exists an integer $\Delta > 0$ such that $\Pr[X_t = q \mid X_0 = q] = 0$, unless t is a multiple of Δ. Furthermore, q is *non-null persistent* if

$$\Pr[\exists t > 0 : X_t = q \mid X_0 = q] = 1$$

and

$$\sum_{t>0} t \Pr[X_t = q \mid X_0 = q, X_1, \dots, X_{t-1} \neq q] < \infty,$$

ie, if every state is revisited with probability 1 after a finite number of steps. The chain is *aperiodic* if none of its states is periodic. It is *ergodic* if it is aperiodic and if all its states are non-null persistent. Any finite, irreducible, aperiodic Markov chain is ergodic. This implies that it has a unique *stationary distribution* π, ie, there exists a unique distribution π with $\pi^T = \pi^T P$. Finally, M is *reversible* if there is a distribution π such that for any $q, r \in Q$ we have $\pi_q p_{qr} = \pi_r p_{rq}$.

Given an undirected graph $G = (V, E)$ and an initial distribution π_0 on V, a *random walk* on G is a sequence of vertices $v_0, v_1, \dots$, where v_0 is chosen according to π_0 and v_{t+1} is found by following a random edge out of v_t. A random walk induces a Markov chain with state space V. This chain is always reversible. It is irreducible if and only if G is connected, and aperiodic if and only if G has no bipartite components. Thus, any connected, non-bipartite graph induces an ergodic Markov chain. In this case, any random walk converges to the stationary distribution given by $\pi_v = \deg(v)/2|E|$ for $v \in V$. In particular, π is uniform if all vertices have the same degree.

7.1.2 Facts about matrices

We recall some basic facts from matrix theory [90]. Let $A \in \mathbb{R}^{n \times n}$. We say that A is *symmetric* if $A^T = A$. A symmetric matrix is *positive semidefinite* if $\vec{v}^T A \vec{v} \geq 0$ for every $\vec{v} \in \mathbb{R}^n$. We call $\vec{v} \in \mathbb{R}^n \setminus \vec{0}$ an *eigenvector* of A if there exists an *eigenvalue* $\lambda \in \mathbb{R}$ with $A\vec{v} = \lambda \vec{v}$. Every symmetric matrix has n real eigenvalues $\lambda_1, \dots, \lambda_n$

and a corresponding orthonormal basis $\vec{v}_1, \dots, \vec{v}_n$ of eigenvectors. They can be characterized as follows [90, Theorem 4.2.11]:

Theorem 7.1.1 (Courant-Fischer). *Let $A \in \mathbb{R}^{n \times n}$ be symmetric with eigenvalues $\lambda_1 \geq \dots \geq \lambda_n$ and corresponding eigenvectors $\vec{v}_1, \dots, \vec{v}_n$. Then, for $k = 1, \dots, n$,*

$$\lambda_k = \max_{\substack{\vec{v} \in \mathbb{R}^n \setminus \vec{0} \\ \vec{v} \perp \vec{v}_1, \dots, \vec{v}_{k-1}}} \frac{\vec{v}^T A \vec{v}}{\vec{v}^T \vec{v}}.$$

The Courant-Fischer theorem allows us to relate the eigenvalues of any principal submatrix of A to those of A [90, Theorem 4.3.15].

Theorem 7.1.2 (Interlacing Theorem). *Let $A \in \mathbb{R}^{n \times n}$ be symmetric with eigenvalues $\lambda_1 \geq \dots \geq \lambda_n$, and let A_r be obtained from A by deleting $n - r$ rows and the corresponding columns from A. Let $\mu_1 \geq \dots \geq \mu_r$ be A_r's eigenvalues. Then, for $1 \leq k \leq r$,*

$$\lambda_k \geq \mu_k \geq \lambda_{n-r+k}.$$

For $A \in \mathbb{R}^{n \times n}$, let G_A be the directed graph on $\{1, \dots, n\}$ which contains an edge from i to j precisely if $A_{ij} \neq 0$. We call A *irreducible* if G_A is strongly connected. If all entries of A are nonnegative, we can say more about its principal eigenvalue and eigenvector [90, Theorem 8.4.4].

Theorem 7.1.3 (Perron-Frobenius). *Let $A \in \mathbb{R}^{n \times n}$ be nonnegative, irreducible, and symmetric with eigenvalues $\lambda_1 \geq \dots \geq \lambda_n$. Then $\lambda_1 > \lambda_2$, and λ_1 has an eigenvector with all positive entries.*

7.1.3 Configuration spaces

A *configuration space* of degree d over a universe $\mathcal{U}$ is a set C of *configurations*. A configuration $\sigma \in C$ is a pair (D_σ, S_σ), where $D_\sigma, S_\sigma \subseteq \mathcal{U}$ are disjoint with $|D_\sigma| \leq d$. D_σ are the *triggers* and S_σ the *stoppers* of σ. Given a subset $U \subseteq \mathcal{U}$, we say σ is *active* in U if $D_\sigma \subseteq U$ and $S_\sigma \cap U = \emptyset$. The configuration space framework is powerful enough to capture many geometric construction problems, as we will explain below.

The generic construction problem can be phrased as follows: given $U \subseteq \mathcal{U}$, find all active configurations in U. The randomized incremental construction (RIC) paradigm solves this problem by picking a random permutation of U and inserting the elements one by one, creating and destroying configurations according to which trigger and stopper sets contain the newly inserted element. In order to locate the conflicting configurations for the new element quickly, the RIC maintains a *conflict graph* $\mathcal{C}$, ie, a bipartite graph representing the conflicts between the currently active

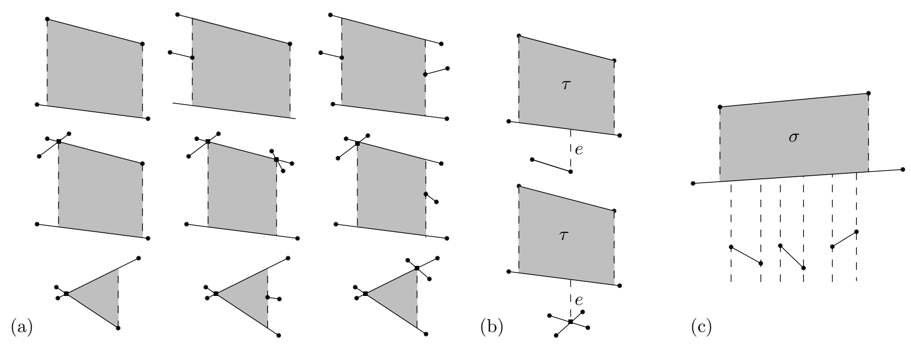


Figure 7.1: (a) A trapezoid is defined by 2, 3 or 4 line segments. (b) The handle e of a racquet (τ, e) is defined by 1 or 2 segments. (c) In an opaque representation, the trapezoid σ is incident to 4 vertices, in a planar graph representation, it is incident to 10 vertices.

configurations and the elements in U that still need to be processed. The graph $\mathcal{C}$ is updated after each insertion, and in our examples this takes time linear in the number of edges in $\mathcal{C}$ that are modified. Algorithms that rely on $\mathcal{C}$ are *static*, since all objects in U need to be known in advance. The influence or history graph [31,122] keeps track of all the configurations that have been active in the construction so far and stores information about their adjacencies that makes it possible to postpone the conflict updates for an element until it is inserted, with the same asymptotic cost. Thus, algorithms that use this structure are *online*, ie, they do not need to know the input beforehand. Using any of the above data structures, the expected running time of RIC for our examples is proportional to

$$\Theta = \sum_{\sigma \in C} |S_\sigma| \Pr[\sigma \text{ becomes active during the construction}].$$

We call this sum the Θ-series of the RIC. In this chapter, we will use the following configuration spaces:

- **Convex hulls in $\mathbb{R}^d$ and Voronoi diagrams in** $\mathbb{R}^{d-1}$ [122, Example 3.4.2]. Let $P \subseteq \mathbb{R}^d$ be in general position. The set C consists of all open half-spaces σ whose bounding hyperplane is spanned by a d-tuple D_σ of distinct points in P. The stopper set S_σ contains all points in $P \cap \sigma$. Clearly, the active configurations in a subset $U \subseteq P$ correspond to the facets of the convex hull of U. Note that a d-tuple D_σ defines two half-spaces, but this can be disambiguated using the ordering of D_σ. By a standard reduction this configuration space also handles Voronoi diagrams in $\mathbb{R}^{d-1}$.

- **Trapezoidal maps** [122, Example 3.4.1]. Let L be a set of nonintersecting planar line segments in general position. To avoid unbounded trapezoids, we assume a large bounding box that contains L. The set C consists of all trapezoids σ that can be defined by a set D_σ of line segments in L (and parts of the bounding box). It is easily seen that $|D_\sigma| \in \{2, 3, 4\}$ (see Figure 7.1a). The stopper set S_σ contains the line segments in L that cross the interior of σ. Again, a tuple D_σ may define more than one trapezoid, which we disambiguate with the ordering information of D_σ. Note that this configuration space gives an *opaque representation* of the map: each trapezoid is incident to at most 6 vertices, even though its bounding segments may be subdivided by trapezoids on the other side (see Figure 7.1c). Since L is nonintersecting, this is sufficient to capture the running time of the RIC, because newly inserted segments cross only vertical boundaries of trapezoids that are destroyed.

- **Segment Intersections** [122, Example 3.4.4]. Let L be a set of planar line segments in general position. Again, we assume a large bounding box for L. Since now a newly inserted segment can cross other line segments, we need a planar graph representation of the trapezoidal map. This is achieved using *racquets*, ie, pairs $\sigma = (\tau, e)$, where τ is a trapezoid and e a vertical attachment. The endpoint of e is defined by 1 or 2 segments, while τ is defined by $2, 3, 4$ segments (see Figure 7.1ab). Thus, $|D_\sigma| \in \{3, 4, 5, 6\}$. The stopper set S_σ of a racquet $\sigma = (\tau, e)$ contains all segments in L that intersect τ or e.

7.2 A simple example: treaps

We begin with a toy example that avoids some of the complications of the general case: suppose that each node v of G is labeled with an element x_v from a totally ordered universe and that all the labels are distinct. The structure to be maintained is a binary search tree T. Start with an empty tree and perform a random walk on G. When the walk reaches v for the first time, insert x_v into T. Our goal is to bound the expected time for the construction of T. For convenience, we assume that G is connected and r-regular[3], for some constant r. The complexity of the algorithm is tightly coupled to the spectral gap γ, which is the difference between the first and second largest eigenvalues of the (stochastic) transition matrix. We will prove the following result.

Theorem 7.2.1. *The expected time to construct the binary search tree is* $O\left(\frac{n \log n}{\gamma}\right)$.

For example, if G is the complete graph with self-loops, $\gamma = 1$ and we get the original theorem about treap construction [119, 141]. More interestingly, if G is a random r-regular graph or an expander, we have $\gamma = \Theta(1)$, and the running time

[3]A graph is *r-regular* if all its nodes have degree r.

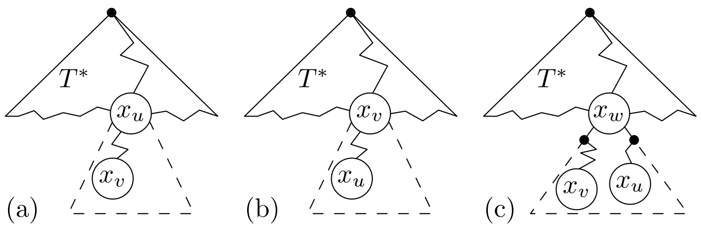


Figure 7.2: All elements x_z, $z \in \overline{S}$, are stored in a subtree rooted at x_w. (a) If $w = u$, then $I_{uv} = 0$; (b) if $w = v$, then $I_{uv} = 1$; and (c) if $w \in S_{uv}$, we have $I_{uv} = 0$.

is still optimal. A cycle, on the other hand, has $\gamma = \Theta(1/n^2)$, and Theorem 7.2.1 predicts a running time of $O(n^3 \log n)$.

For distinct $u, v \in V$, let S_{uv} denote the set of nodes $z \in V$ such that x_z lies in the open interval bounded by x_u and x_v. Furthermore, let I_{uv} be the indicator random variable for the event that x_u is compared to x_v when x_u is inserted into T, ie, the event that x_v is an ancestor of x_u in the binary search tree. Clearly, the time to insert x_u into T is proportional to $\sum_{v \in V \setminus u} I_{uv}$. We claim that $I_{uv} = 1$ precisely if the random walk encounters v before any other node in $\overline{S} = S_{uv} \cup \{u, v\}$ [141, Lemma 4.3]: let w be the first node in $\overline{S}$ that is encountered during the random walk, and let T^* be the tree just before the insertion of x_w. Since the labels of the nodes in $\overline{S}$ constitute an interval, for every element x in T^* the comparison of x with any x_z, $z \in \overline{S}$, yields the same result, irrespective of which $z \in \overline{S}$ is chosen. Therefore, the search paths in T^* for all x_z, $z \in \overline{S}$, are identical. It follows that all those elements will be stored in a subtree rooted at x_w. We can now cover all the cases (see Figure 7.2): if $w = u$, then $I_{uv} = 0$, and if $w = v$, then $I_{uv} = 1$. Finally, if $w \in S_{uv}$, then x_u and x_v will be stored in different subtrees of x_w, and hence they will never be compared to each other, ie, $I_{uv} = 0$. Thus, the expected time to build the binary search tree is proportional to

$$\Theta = \sum_{\substack{u,v \in V \\ u \neq v}} \mathbf{E}\left[I_{uv}\right] =$$

$$\sum_{\substack{u,v \in V \\ u \neq v}} \Pr[\text{the random walk meets } v \text{ before any node in } S_{uv} \cup \{u\}]. \quad (7.1)$$

To get a handle on this sum, we need some random walk theory. Recall that the *transition matrix* of a Markov process with n states is the $n \times n$ matrix P in which entry P_{ij} is the probability of a transition from state i to state j. The transition matrix of a random walk on a graph G is its adjacency matrix, normalized so that each row sums to one. Furthermore, for any initial probability distribution $\pi_0^T \in \mathbb{R}^n$, the distribution after t steps equals $\pi_0^T P^t$.

For technical reasons, we assume a lazy walk with $P = \frac{1}{2}(I + M/r)$, where M is the adjacency matrix of G. This is only for analytical convenience, and an actual implementation could assume a random walk in the original graph G. For the cost of a constant-factor slowdown, the lazy walk brings with it well-known analytical benefits. For example, P is positive semidefinite and the walk is ergodic. Fix a node $u_0 \in V$. Given any nonempty set $S \subset V$ and $u \in V \setminus S$, let $\Pr[u_0 \overset{u}{\to} S]$ be the probability that an infinite walk from u_0 reaches u before any node in S, and let $t_0 = \lfloor c(1-\lambda)^{-1} \log n \rfloor$ be an upper bound on the mixing time, where λ is the second largest eigenvalue of P and c is a large enough constant [53]. Note that $\lambda = 1 - \gamma/2$ (the factor 1/2 comes from the lazy walk) and that

$$\lambda^{t_0} \leq 1/n, \tag{7.2}$$

for appropriate c, since $\lambda = 1 - (1 - \lambda) \leq \exp(\lambda - 1)$. We begin with a technical result of independent interest.[4]

Lemma 7.2.2. *For any given $u_0 \in V$, nonempty $S \subset V$, $u \in V \setminus S$,*

$$\Pr[u_0 \overset{u}{\to} S] \ll \sum_{0 \leq t < 3t_0} (P^t)_{u_0 u} + \frac{1}{(1-\lambda)|S|}.$$

Proof. We may assume that $u_0 \neq u$, since otherwise the sum on the right-hand side is at least 1 and the lemma holds trivially. Similarly, we assume $u_0 \notin S$, since otherwise $\Pr[u_0 \overset{u}{\to} S] = 0$. Let Q be the matrix derived from P by zeroing out any entry P_{vw} with either v or w (or both) in $S \cup \{u\}$. (We index matrix elements and vector coordinates by their corresponding nodes in G.) Being positive semidefinite, Q has a (real) spectral decomposition $\sum_i \mu_i \vec{z}_i \vec{z}_i^T$ such that $\mu_1 \geq \cdots \geq \mu_n = 0$ and the $\vec{z}_i$ constitute an orthonormal basis of eigenvectors. By the Perron-Frobenius theorem (Theorem 7.1.3 in Section 7.1.2), $\lambda < 1$. We also have $\mu_1 < 1$. To see why, note that the components of $G \setminus (S \cup \{u\})$ induce a decomposition of Q into block matrices. If $\mu_1 = 1$, one of these block matrices Q' would have principal eigenvalue 1 and Perron-Frobenius would yield a corresponding eigenvector with all positive entries. But this is impossible, since Q' has a row whose entries sum to less than 1 (and all other rows sum to at most 1).[5] By the eigenvalue interlacing lemma

[4]Recall that we use the Vinogradov notation $\ll$ and $\gg$ for $O(\cdot)$ and $\Omega(\cdot)$, respectively.

[5]Let $\vec{v}$ be this eigenvector, and suppose that the ith row of Q' sums to less than 1. Since $\mu_1 = 1$, there must be an index j such that $Q'_{ij} > 0$ and $v_j > v_i$, where v_i, v_j denote the corresponding components of $\vec{v}$. The jth row sums to at most 1, and by symmetry $Q'_{ji} > 0$. Hence there must be a j' with $Q'_{jj'} > 0$ and $v_{j'} > v_j$. Repeating this argument yields an arbitrarily long strictly increasing sequence of components of $\vec{v}$, which is impossible since $\vec{v}$ has finite dimension.

(Theorem 7.1.2), $\mu_2 \le \lambda$, so for any $v, w \in V \setminus (S \cup \{u\})$

$$\begin{aligned}
(Q^t)_{vw} &= z_{1v}z_{1w}\,\mu_1^t + \sum_{i>1} z_{iv}z_{iw}\,\mu_i^t && \text{(spectral decomposition)} \\
&\le z_{1v}z_{1w}\,\mu_1^t + \mu_2^t\sqrt{\sum_{i>1} z_{iv}^2 \sum_{i>1} z_{iw}^2} && \text{(Cauchy-Schwarz and } \mu_i \le \mu_2\text{)} \qquad (7.3) \\
&\le z_{1v}z_{1w}\,\mu_1^t + \lambda^t. && \text{(orthonormality of the } \vec{z}_i \text{ and } \mu_2 \le \lambda\text{)}
\end{aligned}$$

Since $\vec{1}/\sqrt{n}$ is the principal unit eigenvector of P for the eigenvalue 1, an analogous calculation for P yields for any $v, w \in V$:

$$(P^t)_{vw} \le \frac{1}{n} + \lambda^t. \qquad (7.4)$$

To bound $\Pr[u_0 \overset{u}{\to} S]$, we proceed as follows: first, we distinguish between short paths (with less than $3t_0$ steps) and long paths (with at least $3t_0$ steps). The contribution of the short paths constitutes the first summand in the bound of Lemma 7.2.2. To analyze the contribution of the long paths, we break down every long path from u_0 to u into a pre-mixing part, a mixed portion, and the premixed part of the reverse path. We then assess the contribution of each piece. Let N_u denote the set of nodes in $V \setminus S$ adjacent to u via a nonloop edge. Since G is r-regular, $|N_u| \le r$. Note that $(Q^t)_{u_0 v}$ is the probability that a t-step random walk from u_0 ends in v while avoiding $S \cup \{u\}$. Therefore,

$$\Pr[u_0 \overset{u}{\to} S] = \frac{1}{r}\sum_{t=0}^{\infty}\sum_{v\in N_u}(Q^t)_{u_0 v} \le \sum_{t<3t_0}(P^t)_{u_0 u} + \frac{1}{r}\sum_{t\ge 3t_0}\sum_{v\in N_u}(Q^t)_{u_0 v}\,. \qquad (7.5)$$

We now break down the long paths. The last summand in (7.5) is bounded by

$$\begin{aligned}
&\frac{1}{r}\sum_{v\in N_u}\sum_{t\ge t_0}\sum_{a,b\in V}(P^{t_0})_{u_0 a}(Q^t)_{ab}(P^{t_0})_{bv} && \text{(break-up and } (Q^{t_0})_{uv} \le (P^{t_0})_{uv}\text{)} \\
\le&\left(\frac{1}{n}+\lambda^{t_0}\right)^2\sum_{t\ge t_0}\sum_{a,b\in V}(Q^t)_{ab} && \text{(by (7.4) and } |N_u| \le r\text{)} \qquad (7.6) \\
\le&\frac{4}{n^2}\sum_{t\ge t_0}\sum_{a,b\in V}(Q^t)_{ab}\,. && \text{(by (7.2))}
\end{aligned}$$

Since $\|\vec{z}_1\|_2 = 1$ and since at least $|S|+1$ of its coordinates are zero (an easy consequence of being an eigenvector for Q), Cauchy-Schwarz yields $\|\vec{z}_1\|_1^2 \le n - |S| - 1$. By applying Perron-Frobenius to the parts of the block decomposition of Q induced by the components of $G \setminus (S \cup \{u\})$, we can assume that $\vec{z}_1$ is nonnegative, and so

$$\sum_{a,b\in V} z_{1a}z_{1b} = \|\vec{z}_1\|_1^2 \le n - |S| - 1. \qquad (7.7)$$

We have

$$\begin{aligned}\Pr[u_0 \xrightarrow{u} S] - \sum_{t<3t_0} (P^t)_{u_0u} &\leq \frac{1}{r} \sum_{t\geq 3t_0} \sum_{v\in N_u} (Q^t)_{u_0v} && \text{(by (7.5))} \\ &\leq \frac{4}{n^2} \sum_{a,b\in V} \sum_{t\geq t_0} (z_{1a}z_{1b}\,\mu_1^t + \lambda^t) && \text{(by (7.3,7.6))} \qquad (7.8) \\ &\leq \frac{4}{n(1-\mu_1)} + \frac{4\lambda^{t_0}}{1-\lambda}\,. && \text{(geom. sum and (7.7))}\end{aligned}$$

We already noted $\mu_1 < 1$. However, to bound (7.8), we need a better estimate on $1 - \mu_1$. This can be done using an argument similar to one given by Broder and Karlin [32]: since $\vec{z}_1$ is nonnegative, $n\vec{z}_1 - \|\vec{z}_1\|_1\vec{1}$ is normal to the principal eigenvector $\vec{1}$ of P, and since P is symmetric, by Courant-Fischer (Theorem 7.1.1),

$$\begin{aligned}\lambda \geq\ & \frac{(n\vec{z}_1 - \|\vec{z}_1\|_1\vec{1})^T P (n\vec{z}_1 - \|\vec{z}_1\|_1\vec{1})}{\|n\vec{z}_1 - \|\vec{z}_1\|_1\vec{1}\|_2^2} \\ &= \frac{n^2\vec{z}_1^T P\vec{z}_1 - n\|\vec{z}_1\|_1(\vec{z}_1^T P\vec{1} + \vec{1}^T P\vec{z}_1) + \|\vec{z}_1\|_1^2\vec{1}^T P\vec{1}}{n^2\|\vec{z}_1\|_2^2 - n\|\vec{z}_1\|_1(\vec{z}_1^T\vec{1} + \vec{1}^T\vec{z}_1) + \|\vec{z}_1\|_1^2\vec{1}^T\vec{1}}.\end{aligned}$$

Now, since $\vec{1}$ is a left and right eigenvector of P, $\vec{1}^T P = \vec{1}^T$ and $P\vec{1} = \vec{1}$. Furthermore, $\vec{1}^T\vec{1} = n$, $\vec{z}_1^T\vec{1} = \vec{1}^T\vec{z}_1 = \|\vec{z}_1\|_1$, and $\|\vec{z}_1\|_2 = 1$. Hence,

$$\lambda \geq \frac{n\vec{z}_1^T P\vec{z}_1 - \|\vec{z}_1\|_1^2}{n - \|\vec{z}_1\|_1^2} \geq \frac{n\mu_1 - \|\vec{z}_1\|_1^2}{n - \|\vec{z}_1\|_1^2}\,,$$

because $\vec{z}_1^T P\vec{z}_1 \geq \vec{z}_1^T Q\vec{z}_1 = \mu_1$. It follows that $n\mu_1 \leq n\lambda + (1-\lambda)\|\vec{z}_1\|_1^2$, and using (7.7) we get $\mu_1 \leq 1 - (1-\lambda)(|S|+1)/n$. Plugging this bound into (7.8) completes the proof, as $\lambda^{t_o} \leq 1/n \leq 1/|S|$ by (7.2). □

Proof of Theorem 7.2.1. By (7.1), the expected running time is

$$\Theta = \sum_{\substack{u,v\in V\\ u\neq v}} \frac{1}{n} \sum_{u_0\in V} \Pr[u_0 \xrightarrow{v} S_{uv}\cup\{u\}],$$

where u_0 is the random start node of the walk. By Lemma 7.2.2,

$$\begin{aligned}\Theta \ll \sum_{\substack{u,v\in V\\ u\neq v}} \frac{1}{n} \sum_{u_0\in V} &\left(\sum_{t=0}^{3t_0-1} (P^t)_{u_0v} + \frac{1}{(1-\lambda)(|S_{uv}|+1)} \right) \\ &= \sum_{\substack{u,v\in V\\ u\neq v}} \left(\frac{3t_0}{n} + \frac{1}{(1-\lambda)(|S_{uv}|+1)} \right),\end{aligned}$$

since $\sum_{u_0 \in V} (P^t)_{u_0 v} = 1$, as $(P^t)_{u_0 v}$ is the probability that a t-step random walk ending in v started out at u_0. Hence,

$$\Theta \ll n t_0 + (1-\lambda)^{-1} \sum_{\substack{i,j=1 \\ i \neq j}}^{n} \frac{1}{|i-j|+1} \ll \gamma^{-1} n \log n.$$

□

7.3 Θ-series for Markov sources

We use the classic notion of *configuration spaces* (see [122] or Section 7.1.3 for a primer) and adapt it to the Markov model. This is done as follows: fix a natural number d, the *degree* of the configuration space. Each node v of G is assigned an object x_v chosen from a geometric universe (eg, points, hyperplanes, segments), and to each d-tuple $\mathbf{u} = (u_1, \ldots, u_d)$ of distinct $u_i \in V$ we assign a (possibly empty) $S_\mathbf{u} \subseteq V$ disjoint from $\mathbf{u}$. We denote by f_k the number of $\mathbf{u}$'s such that $|S_\mathbf{u}| = k$ and by $f_{\leq k}$ the prefix sum $f_0 + \cdots + f_k$. We write $f_k(n)$ and $f_{\leq k}(n)$ to refer to the maximum such values over all subsets of the universe of size n. The coordinates of a d-tuple $\mathbf{u}$ play the role of the *triggers* and the sets $S_\mathbf{u}$ that of the *stoppers.* Naturally, f_k counts the k-sets of the underlying range space.

The apparent simplifications of our model do not, in fact, restrict the generality of the results in any way. Indeed, our framework can just as easily handle cases where $\mathbf{u}$ is not a sequence but a multiset, where it maps to several stopper sets, or where the degree d is not unique. Given a random ordered $\mathbf{u} = (u_1, \ldots, u_d)$ with distinct elements, perform an infinite random walk from a random node in G. If the walk first reaches $u_1, \ldots, u_d$ in that order before hitting any node in $S_\mathbf{u}$, then set $\Phi = n^d |S_\mathbf{u}|$; else set $\Phi = 0$. Standard Θ-series theory shows that the expectation of Φ determines the expected amortized complexity of RIC [122]. As before, we assume that the graph is connected and r-regular, and we let γ denote the spectral gap. We postpone the proof of this result:

Theorem 7.3.1 (Master Theorem)**.** *If there is a constant $\alpha > 0$ with $f_0(n) = O(n^\alpha)$, then $\mathbf{E}[\Phi] \ll \gamma^{-d} n^\alpha (\log n)^{d-\alpha}$ for $\alpha > 1$ and $\mathbf{E}[\Phi] \ll \gamma^{-d} n (\log n)^d$ for $\alpha \leq 1$.*

We apply the theorem to three problems: convex hulls (and hence Voronoi diagrams); trapezoidal maps of disjoint segments; and line segment intersections. For simplicity, we assume that the input is in general position. The algorithms themselves operate in standard incremental fashion by inserting objects online with the help of the history graph. The algorithms do not require knowledge of the Markov chain (which is why we do not use conflict graphs).

- **Convex hulls in $\mathbb{R}^d$.** The convex hull of n points in $\mathbb{R}^d$ has $O\left(n^{\lfloor d/2 \rfloor}\right)$ faces, which implies that $\alpha = \lfloor d/2 \rfloor$. The algorithm runs in $O\left(\gamma^{-d} n^{\lfloor d/2 \rfloor} (\log n)^{\lceil d/2 \rceil}\right)$ time for $d > 3$, and $O\left(n(\gamma^{-1} \log n)^d\right)$ for $d \leq 3$.

- **Trapezoidal maps.** At each node, the trapezoidal map formed by a set of (nonintersecting) segments is maintained. The relevant configuration space is made of three subconfiguration spaces of respective degrees 2, 3, and 4. Hence, the time required by the algorithm is $O(n(\gamma^{-1} \log n)^4)$.

- **Segment intersections.** The m intersections among n segments are computed in $O\left((n+m)(\gamma^{-1} \log n)^6\right)$ steps. The proof depends on an extension of the Master Theorem discussed in Section 7.4.

To bound the expectation of Φ, we need to understand a certain stochastic process, which we proceed to describe. A random *thread* refers either to a single node w_1 chosen uniformly at random (thread size of 1) or to a sequence $w_1, \ldots, w_l$ (thread size of $l > 1$), where w_1 is random and, for each $i > 0$, w_{i+1} is the end node of a random walk from w_i of length $t_i > 0$. The time sequence $\theta = (t_1, \ldots, t_{l-1})$ parametrizes the thread. Given $1 \leq \mu \leq d$, a random *μ-thread* is a sequence of μ threads whose sizes add up to d: each thread is drawn independently and has its own size and time sequence. Its time sequence θ refers now to the collection of its constituent threads' time sequences. A μ-thread forms a d-tuple $\mathbf{u}$ and is therefore associated with a stopper set[6] $S_{\mathbf{u}}$. Let $g_k^{(\mu)}$ be the probability that a random μ-thread (with a given time sequence) produces $\mathbf{u}$ such that $|S_{\mathbf{u}}| = k$.

$$g_k^{(\mu)} = \Pr\left[\, \mu\text{-thread} \hookrightarrow \mathbf{u} \,:\, |S_{\mathbf{u}}| = k \,\right], \tag{7.9}$$

and let $g_{\leq k}^{(\mu)} = \sum_{0 \leq i \leq k} g_i^{(\mu)}$.

Lemma 7.3.2. *Let f_0 be monotonically increasing. For any μ-thread and any corresponding time sequence $\theta_1, \ldots, \theta_\mu$, we have $g_{\leq k}^{(\mu)} \ll (k/n)^\mu f_0(n/k)$, for $k > 0$.*

Proof. We use a Clarkson-Shor type counting argument [59] tailored for Markov chains.[7] As usual, the idea is to use sampling in order to bound $g_{\leq k}^{(\mu)}$ in terms of f_0. More precisely, we sample a set $R^v \subseteq V$ of size about n/k. Then, for a configuration $\mathbf{u} \subseteq R^v$ with $|S_{\mathbf{u}}| \leq k$, we argue that with constant probability $\mathbf{u}$ is active in R^v, ie, $S_{\mathbf{u}} \cap R^v = \emptyset$. Together with a bound on the probability that a given configuration $\mathbf{u}$ appears in R^v, this yields the desired result. We may assume that $k \leq n/2d$, since for larger k the bound becomes constant and $g_{\leq k}^{(\mu)} \leq g_{\leq n}^{(\mu)} \leq 1$.

[6]This is not true if $\mathbf{u}$ contains less than d distinct nodes. Since Lemma 7.3.2 deals only with finite stopper sets, we can invalidate this case by setting $S_{\mathbf{u}} = \mathbb{R}$, or any other infinite set.

[7]See the proof of Claim 2.2.7 for a basic application of the Clarkson-Shor method.

All μ-threads in this proof share the given time sequence $\theta_1, \dots, \theta_\mu$. Let $s \leq n$ be an integer to be determined later. For each $i = 1, \dots, \mu$, pick s random threads of type θ_i, and define R as the set of $\mathbf{u}$'s formed by taking all possible s^μ combinations of the resulting threads, one of each type. Given a fixed (nonrandom) $\mathbf{u} \in V^d$, let $p_\mathbf{u}$ denote the probability that $\mathbf{u}$ is chosen by a random μ-thread. Since each starting node is chosen independently, $p_\mathbf{u}$ is of the form $\prod_{1 \leq i \leq \mu} \frac{p_{\mathbf{u},i}}{n}$, where $p_{\mathbf{u},i}$ is the probability that the i-th thread visits the relevant nodes of $\mathbf{u}$ in the correct order, given that the first node of the i-th thread equals the corresponding node in $\mathbf{u}$. Therefore, $\mathbf{u}$ ends up in R with probability at least $\prod_{1 \leq i \leq \mu}(1 - (1 - p_{\mathbf{u},i}/n)^s)$. Now, since $p_{\mathbf{u},i} s/n \leq 1$, we have

$$\left(1 - \frac{p_{\mathbf{u},i}}{n}\right)^s \leq 1 - \binom{s}{1}\frac{p_{\mathbf{u},i}}{n} + \binom{s}{2}\left(\frac{p_{\mathbf{u},i}}{n}\right)^2 \leq 1 - \frac{p_{\mathbf{u},i}s}{n} + \frac{1}{2}\left(\frac{p_{\mathbf{u},i}s}{n}\right)^2 \leq 1 - \frac{p_{\mathbf{u},i}s}{2n}\,; \tag{7.10}$$

hence,

$$\Pr[\,\mathbf{u} \in R\,] \geq \prod_{i=1}^{\mu} \frac{p_{\mathbf{u},i}s}{2n} \gg p_\mathbf{u}\, s^\mu. \tag{7.11}$$

Let R^v be the collection of nodes appearing among the d-tuples of R. Given a fixed $\mathbf{u}$ with $|S_\mathbf{u}| \leq n/2d$, conditioned upon $\mathbf{u} \in R$, what is the probability that $R^v \cap S_\mathbf{u} = \emptyset$, ie, that configuration $\mathbf{u}$ is active in R^v? Being in R, $\mathbf{u}$ itself is a μ-thread formed by picking exactly one thread per type out the s available ones in R. The d nodes of $\mathbf{u}$ lie outside $S_\mathbf{u}$, so the only possibility for R^v to intersect $S_\mathbf{u}$ is for any of the $(s-1)\mu$ other threads to pass through $S_\mathbf{u}$. Take one of them: it is a random walk $w_1 \dots w_l$. The starting node w_1 is random, so its distribution forms an eigenvector for the thread's transition matrix with eigenvalue 1 (also true if $l = 1$). This means that each w_i lies in $S_\mathbf{u}$ with probability $|S_\mathbf{u}|/n$. These events are not independent, so we use a union bound to argue that the thread $w_1 \dots w_l$ remains outside $S_\mathbf{u}$ with probability at least $1 - l|S_\mathbf{u}|/n \geq 1 - d|S_\mathbf{u}|/n$. The $(s-1)\mu$ threads that are candidates for passing through $S_\mathbf{u}$ are independent, however, and thus refrain from doing so with probability at least $(1 - d|S_\mathbf{u}|/n)^{(s-1)\mu}$. For any $x \in [0, 1/2]$ we have $(1-x)^{-1} \leq (1+x)^2 \leq \exp(2x)$. Thus, $(1 - d|S_\mathbf{u}|/n) \geq \exp(-2d|S_\mathbf{u}|/n)$, since $|S_\mathbf{u}| \leq n/2d$. It follows that

$$\Pr[\,R^v \cap S_\mathbf{u} = \emptyset \,|\, \mathbf{u} \in R\,] \geq \left(1 - \frac{d|S_\mathbf{u}|}{n}\right)^{(s-1)\mu} \geq e^{-2d(s-1)\mu|S_\mathbf{u}|/n}.$$

If $r_\mathbf{u}$ denotes the probability that both $\mathbf{u} \in R$ and $S_\mathbf{u} \cap R^v = \emptyset$ then, by (7.11), setting $s = \frac{n}{dk}$ yields

$$r_\mathbf{u} = \Pr[\,\mathbf{u} \in R\,] \times \Pr[\,R^v \cap S_\mathbf{u} = \emptyset \,|\, \mathbf{u} \in R\,] \gg \left(\frac{n}{dk}\right)^\mu p_\mathbf{u}\, e^{-2\mu|S_\mathbf{u}|/k}\,;$$

therefore, since $\mu \le d$ and $k \le n/2d$,

$$\sum_{\mathbf{u}\colon |S_{\mathbf{u}}|\le n/2d} r_{\mathbf{u}} \gg \sum_{\mathbf{u}\colon |S_{\mathbf{u}}|\le n/2d} \left(\frac{n}{dk}\right)^{\mu} p_{\mathbf{u}} e^{-2d|S_{\mathbf{u}}|/k} \gg \sum_{\mathbf{u}\colon |S_{\mathbf{u}}|\le k} \left(\frac{n}{dk}\right)^{\mu} p_{\mathbf{u}} \gg \left(\frac{n}{k}\right)^{\mu} g_{\le k}^{(\mu)},$$

as $g_{\le k}^{(\mu)} = \sum_{\mathbf{u}:|S_{\mathbf{u}}|\le k} p_{\mathbf{u}}$. Since $|R^v| \le ds$, by definition and by the monotonicity of f_0, $|\{\mathbf{u} \in R : |S_{\mathbf{u}} \cap R^v| = 0\}| \le f_0(ds)$; therefore,

$$\sum_{\mathbf{u}\colon |S_{\mathbf{u}}|\le n/2d} r_{\mathbf{u}} \le f_0(n/k).$$

Note that this holds uniformly over all time sequences for the μ-thread. $\square$

Proof of Theorem 7.3.1. Recall that our goal is to bound the expectation of a random variable Φ defined as follows: pick a random d-tuple $\mathbf{u} = (u_1, u_2, \ldots, u_d)$ of distinct $u_i \in V$ and perform an infinite random walk in G starting at a random node u_0. If the walk encounters all the nodes in $\mathbf{u}$ in that order before any node in $S_{\mathbf{u}}$, let $\Phi = n^d|S_{\mathbf{u}}|$, otherwise, let $\Phi = 0$. The expectation of Φ is given by

$$\mathbf{E}[\Phi] = \frac{n^d d!}{\binom{n}{d}} \sum_{\mathbf{u}} \frac{1}{n} \sum_{u_0\in V} |S_{\mathbf{u}}| \prod_{i=0}^{d-1} \Pr\left[u_i \overset{u_{i+1}}{\to} S_{\mathbf{u}} \cup \{u_{i+2}, \ldots, u_d\}\right],$$

where $\sum_{\mathbf{u}}$ ranges over all ordered subsets of d distinct nodes: obviously, we may restrict the sum to $\{\mathbf{u} : |S_{\mathbf{u}}| > 0\}$. The sum $d!\binom{n}{d}^{-1}\sum_{\mathbf{u}}$ represents the random choice of $\mathbf{u}$, the sum $n^{-1}\sum_{u_0\in V}$ accounts for the random starting vertex. The product denotes the probability that a random walk from u_0 visits the nodes $u_1, \ldots, u_d$ in that order before it encounters any node in $S_{\mathbf{u}}$. Note that removing elements from S cannot decrease $\Pr\left[u_0 \overset{u}{\to} S\right]$; therefore,

$$\mathbf{E}[\Phi] \ll \sum_{\mathbf{u}} \frac{A_{\mathbf{u}}}{n} |S_{\mathbf{u}}| \prod_{i=1}^{d-1} \Pr\left[u_i \overset{u_{i+1}}{\to} S_{\mathbf{u}}\right],$$

where

$$\begin{aligned}
A_{\mathbf{u}} &= \sum_{u_0\in V} \Pr\left[u_0 \overset{u_1}{\to} S_{\mathbf{u}}\right] \\
&\ll \sum_{u_0\in V} \left(\sum_{0\le t<3t_0} (P^t)_{u_0u_1} + \frac{1}{(1-\lambda)|S_{\mathbf{u}}|} \right) && \text{(Lemma 7.2.2)} \\
&= 3t_0 + \frac{n}{(1-\lambda)|S_{\mathbf{u}}|} && \left(\text{by } \sum_{u_0} (P^t)_{u_0u_1} = 1\right) \\
&\ll \frac{n}{1-\lambda}\left(\frac{1}{|S_{\mathbf{u}}|} + \frac{\log n}{n}\right). && \left(\text{by } t_0 \ll (1-\lambda)^{-1}\log n\right)
\end{aligned}$$

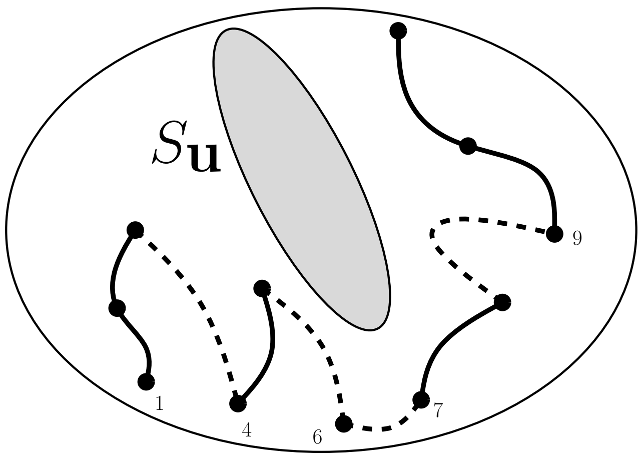


Figure 7.3: The index set $L = \{1, 2, 4, 7, 9, 10\}$ defines a 5-thread.

Thus, using Lemma 7.2.2 once more,

$$\mathbf{E}[\Phi] \ll \frac{1}{1-\lambda} \sum_{\mathbf{u}} \left(1 + \frac{|S_{\mathbf{u}}| \log n}{n}\right) \prod_{i=1}^{d-1} \left\{ \sum_{0 \le t < 3t_0} (P^t)_{u_i u_{i+1}} + \frac{1}{(1-\lambda)|S_{\mathbf{u}}|} \right\}. \tag{7.12}$$

Writing (7.12) as $\mathbf{E}[\Phi] \ll (1-\lambda)^{-1} \sum_{\mathbf{u}} (1 + |S_{\mathbf{u}}|(\log n)/n) B_{\mathbf{u}}$, we begin with the sum $\sum_{\mathbf{u}} B_{\mathbf{u}}$. Expanding the $(d-1)$-fold product $B_{\mathbf{u}}$ produces 2^{d-1} terms of the form

$$\left(\frac{1}{1-\lambda}\right)^j \frac{1}{|S_{\mathbf{u}}|^j} \prod_{i \in L} \sum_{0 \le t < 3t_0} (P^t)_{u_i u_{i+1}} = \sum_{\substack{\theta = (t_i)_{i \in L} \\ 0 \le t_i < 3t_0}} \left(\frac{1}{1-\lambda}\right)^j \frac{1}{|S_{\mathbf{u}}|^j} \prod_{i \in L} (P^{t_i})_{u_i u_{i+1}} \tag{7.13}$$

where $L \subseteq \{1, \dots, d-1\}$ and $j + |L| = d - 1$. Let

$$C_{\mathbf{u}}^{L,\theta} = \left(\frac{1}{1-\lambda}\right)^j \frac{1}{|S_{\mathbf{u}}|^j} \prod_{i \in L} (P^{t_i})_{u_i u_{i+1}},$$

so $\sum_{\mathbf{u}} B_{\mathbf{u}} = \sum_{\mathbf{u}} \sum_L \sum_\theta C_{\mathbf{u}}^{L,\theta}$. The index set L specifies the parameters of a μ-thread (except for its time sequence). Indeed, break $1, \dots, d$ into $\mu = j + 1$ intervals by applying the rule that i and $i + 1$ are in the same interval precisely if $i \in L$. In Figure 7.3, $d = 11$, $\mu = 5$, $j = 4$, $L = \{1, 2, 4, 7, 9, 10\}$, and the threads are $[1, 2, 3], [4, 5], [6], [7, 8], [9, 10, 11]$. All we can say about the time sequences is that their total number of elements $t_1, t_2, \dots$ is exactly $|L| = d - \mu$. We use the superscripts L, θ in the sums to indicate a fixed L and/or a fixed time sequence θ.

$$\sum_{\mathbf{u}} B_{\mathbf{u}} = \sum_{L} \sum_{\theta} \sum_{\mathbf{u}} C_{\mathbf{u}}^{L,\theta} = \sum_{L} \sum_{\theta} \left(\frac{1}{1-\lambda} \right)^{j} \sum_{\mathbf{u}}^{L,\theta} \frac{n^{\mu}}{|S_{\mathbf{u}}|^{j}} \Pr[\,\mu\text{-thread } \hookrightarrow \mathbf{u}\,]\,. \tag{7.14}$$

Note the presence of the factor n^{μ} to make up for the fact that in a μ-thread each thread starts from a random vertex, whereas in $C_{\mathbf{u}}^{L,\theta}$ each thread starts from the corresponding u_i. Assume that $j > 0$. Now, the sum $\sum_{\mathbf{u}}^{L,\theta} |S_{\mathbf{u}}|^{-j} \Pr[\,\mu\text{-thread } \hookrightarrow \mathbf{u}\,]$ can be upper-bounded using summation by parts:

$$\begin{aligned}
\sum_{\mathbf{u}}^{L,\theta} \frac{\Pr[\,\mu\text{-thread } \hookrightarrow \mathbf{u}\,]}{|S_{\mathbf{u}}|^{j}} &= \sum_{k=1}^{n} \frac{g_k^{(\mu)}}{k^j} && \text{(group by } |S_{\mathbf{u}}|) \\
&= \frac{g_{\le n}^{(\mu)}}{n^j} - g_0^{(\mu)} + \sum_{k=1}^{n-1} g_{\le k}^{(\mu)} \Big(\frac{1}{k^j} - \frac{1}{(k+1)^j} \Big) && \text{(sum by parts)} \\
&\ll \frac{1}{n^j} + \sum_{k=1}^{n-1} \frac{g_{\le k}^{(\mu)}}{k^{j+1}} && (g_{\le n}^{(\mu)} \le 1)
\end{aligned} \tag{7.15}$$

$$\ll \frac{1}{n^j} + \frac{1}{n^{\mu}} \sum_{k=1}^{n-1} \frac{f_0(n/k)}{k^{j+1-\mu}}\,. \qquad \text{(Lemma 7.3.2)}$$

Let $\sum_{\mathbf{u}}^{L} B_{\mathbf{u}}$ denote the sum obtained by collecting all summands in $\sum_{\mathbf{u}} B_{\mathbf{u}}$ with a fixed L. By (7.14) and using the identity $\mu = j + 1$,

$$\begin{aligned}
\sum_{\mathbf{u}}^{L} B_{\mathbf{u}} &\ll \sum_{\theta}^{L} \left(\frac{1}{1-\lambda} \right)^{j} n^{\mu} \sum_{\mathbf{u}}^{L,\theta} |S_{\mathbf{u}}|^{-j} \Pr[\mu\text{-thread } \hookrightarrow \mathbf{u}] && \text{(by (7.14))} \\
&\ll (3t_0)^{d-\mu} \left(\frac{1}{1-\lambda} \right)^{j} \Big(n + \sum_{k=1}^{n-1} f_0(n/k) \Big) && ((7.15),\ \mu = j+1, \\
& && \text{and } |\theta| = d - \mu) \\
&\ll (\log n)^{d-\mu} \Big(\frac{1}{1-\lambda} \Big)^{d-1} \left(n + \sum_{k=1}^{n-1} \left(\frac{n}{k} \right)^{\alpha} \right). && \text{(since } f_0(n) = n^{\alpha})
\end{aligned} \tag{7.16}$$

We can now easily cover all cases:

(I) $j > 0$ and $\alpha \le 1$: $\sum_k (n/k)^{\alpha} \ll n \log n$; hence, using $\mu = j + 1$,

$$\sum_{\mathbf{u}}^{L} B_{\mathbf{u}} \ll (1-\lambda)^{1-d} n (\log n)^{d-j} \ll (1-\lambda)^{1-d} n (\log n)^{d-1}.$$

(II) $j > 0$ and $\mu \geq \alpha > 1$: $\sum_k (n/k)^\alpha \ll n^\alpha$; hence

$$\sum_{\mathbf{u}}^{L} B_{\mathbf{u}} \ll (1-\lambda)^{1-d}\, n^\alpha\, (\log n)^{d-\mu} \ll (1-\lambda)^{1-d}\, n^\alpha\, (\log n)^{d-\alpha}.$$

(III) $\alpha > \mu$ or $j = 0$: by (7.14),

$$\sum_{\mathbf{u}}^{L} B_{\mathbf{u}} \;\ll\; (1-\lambda)^{1-d}\, n^\mu\, (\log n)^{d-1-j}.$$

If $\alpha > \mu$, then $\sum_{\mathbf{u}}^{L} B_{\mathbf{u}} = o((1-\lambda)^{1-d} n^\alpha)$. If $j = 0$, then $\sum_{\mathbf{u}}^{L} B_{\mathbf{u}} \ll (1-\lambda)^{1-d}\, n(\log n)^{d-1}$.

Since the bounds are independent of L for which there are only constantly many choices, we conclude that

$$\sum_{\mathbf{u}} B_{\mathbf{u}} \;\ll\; (1-\lambda)^{1-d} \max\Big\{\, n(\log n)^{d-1},\, n^\alpha\, (\log n)^{d-\alpha} \,\Big\}.$$

Going back to (7.12), recall that

$$\mathbf{E}\,[\Phi] \ll (1-\lambda)^{-1} \sum_{\mathbf{u}} \left(1 + \frac{|S_{\mathbf{u}}| \log n}{n}\right) B_{\mathbf{u}}.$$

To handle $D = \frac{\log n}{n} \sum_{\mathbf{u}} B_{\mathbf{u}} |S_{\mathbf{u}}|$, first note that since $|S_{\mathbf{u}}| \leq n$, we never lose more than a factor of $\log n$. However, in Case **(II)** we can do better. Assume $j > 0$ and $\mu \geq \alpha > 1$. We revisit the above calculation. With the additional factor of $|S_{\mathbf{u}}|(\log n)/n$, (7.14) becomes

$$D = \sum_{L} \sum_{\theta} \log n \Big(\frac{1}{1-\lambda}\Big)^j \sum_{\mathbf{u}}^{L,\theta} \frac{n^{\mu-1}}{|S_{\mathbf{u}}|^{j-1}} \Pr[\,\mu\text{-thread } \hookrightarrow \mathbf{u}\,]. \tag{7.17}$$

First, if $j = 1$, then $\mu = 2$ and $\sum_{\mathbf{u}}^{L,\theta} n^{\mu-1} |S_{\mathbf{u}}|^{1-j} \Pr[\mu\text{-thread } \hookrightarrow \mathbf{u}] \leq n$. Therefore in this case $D^L \ll (1-\lambda)^{1-d} n(\log n)^{d-1}$, where D^L denotes the sum obtained by collecting all the terms of D with a fixed L.
Next, we consider the case $j \geq 2$. Similarly to (7.15),

$$\sum_{\mathbf{u}}^{L,\theta} |S_{\mathbf{u}}|^{1-j} \Pr[\mu\text{-thread } \hookrightarrow \mathbf{u}] \ll \frac{1}{n^{j-1}} + \frac{1}{n^\mu} \sum_{k=1}^{n} \frac{f_0(n/k)}{k^{j-\mu}}. \tag{7.18}$$

Hence, (7.16) becomes

$$
\begin{aligned}
&\frac{\log n}{n} \sum_{\mathbf{u}}^{L} B_{\mathbf{u}} |S_{\mathbf{u}}| \\
&\ll \sum_{\theta}^{L} \log n \left(\frac{1}{1-\lambda}\right)^{j} n^{\mu-1} \sum_{\mathbf{u}}^{L,\theta} |S_{\mathbf{u}}|^{1-j} \Pr[\mu\text{-thread } \hookrightarrow \mathbf{u}] && \text{(by (7.17))} \\
&\ll (\log n)^{d+1-\mu} \Big(\frac{1}{1-\lambda}\Big)^{d-1} \left(n + \frac{k}{n} \sum_{k=1}^{n} \Big(\frac{n}{k}\Big)^{\alpha}\right) && \text{(by (7.18))} \\
&= (\log n)^{d+1-\mu} \Big(\frac{1}{1-\lambda}\Big)^{d-1} \left(n + \sum_{k=1}^{n} \Big(\frac{n}{k}\Big)^{\alpha-1}\right),
\end{aligned}
$$

and since $\alpha > 1$, $D^L = o((1-\lambda)^{1-d} n^{\alpha})$.
Thus, accounting for the extra $\log n$-factor in Cases **(I)** and **(III)**,

$$
\mathbf{E}[\Phi] \ll (1-\lambda)^{-d} \max\Big\{ n(\log n)^{d},\ n^{\alpha} (\log n)^{d-\alpha} \Big\}.
$$

This completes the proof of the Master Theorem. $\square$

7.4 Extensions

Segment intersections. The Master Theorem cannot be used for the trapezoidal map of intersecting segments. The reason is that the complexity of an arrangement of n segments depends on both n and the number m of intersections. We show how to extend the Master Theorem to handle this case. The problem can be described by a configuration space that is made of subconfiguration spaces of degrees $3, 4, 5, 6$ with $f_0(n, m) \ll n + m$. We need to strengthen Lemma 7.3.2:

Lemma 7.4.1. *Let $g_{\leq k}^{(\mu)}$ be as defined in Lemma 7.3.2; for any μ-thread, any corresponding time sequence and any $k > 0$,*

$$
g_{\leq k}^{(\mu)} \ll (k/n)^{\mu} (n+m)/k.
$$

Proof. We use the same notation as in Lemma 7.3.2. We only need a better upper bound on $\sum_{\mathbf{u}} r_{\mathbf{u}}$, the expected complexity of the trapezoidal map for the sample R^v. To do this, we bound the expected number of intersections among the line segments x_z, $z \in R^v$. Let I be an intersection and let x_u be one of its defining segments: I can only be present in the trapezoidal map for R^v if $u \in R^v$. This happens with probability at most $1 - (1 - d/n)^s$: we have s independent samples of d nodes, each of which could be u with probability $1/n$. Since $d/n \leq 1$, we have $1-(1-d/n)^s \le ds/n = 1/k$. By linearity of expectation, it follows that the expected number of intersections is $O(m/k)$, which gives the desired upper bound of $O((n+m)/k)$ on the expected complexity of the trapezoidal map for R^v. Together with the lower bound from the proof of Lemma 7.3.2, this completes the proof. □

The desired result follows now by repeating the proof of the Master Theorem with the bound $g_{\le k}^{(\mu)} \ll (n+m)/k$ in (7.15). Then, (7.16) becomes

$$\sum_{\mathbf{u}}^{L} B_{\mathbf{u}} \ll (\log n)^{d-\mu} \left(\frac{1}{1-\lambda}\right)^{d-1} \left(n + \sum_{k=1}^{n-1} \frac{n+m}{k}\right),$$

and as in Cases **(I)** and **(III)** we find $\sum_{\mathbf{u}} B_{\mathbf{u}} \ll (1-\lambda)^{1-d}(n+m)(\log n)^{d-1}$. Accounting for the additional $\log n$-factor (and the factor $(1-\lambda)^{-1}$), this yields $\mathbf{E}[\Phi] \ll (1-\lambda)^{-d}(n+m)(\log n)^d$. To summarize, the m intersections among n segments are computed in time $O\left((n+m)(\gamma^{-1}\log n)^6\right)$, as we claimed earlier.

Revisiting the Clarkson-Shor bound. While proving the Master Theorem, we obtained a variant of the Clarkson-Shor bound suited for our Markov model (Lemma 7.3.2). We believe that this lemma is of independent interest and could lead to new bounds on the number of k-sets when certain restrictions on the defining elements are imposed. Here is a toy example: let $P \subseteq \mathbb{R}^3$ be a set of n points in general position. Let H be the set of planes in $\mathbb{R}^3$ spanned by triplets of the form $(x, y, \mathtt{n}(x))$ for $x, y \in P$, where $\mathtt{n}(x)$ denotes a neighbor of x in the Euclidean minimum spanning tree (EMST) of P. A plane $h \in H$ *conflicts* with a point $p \in P$ if p lies below h. Let $f_{\le k}$ denote the number of planes in H that conflict with at most k points.

Corollary 7.4.2. *$f_{\le k} \ll nk$.*

Let H' denote the planes spanned by triplets of the form $(x, y, \mathtt{nn}(x))$, $x, y \in P$, where $\mathtt{nn}(x)$ denotes the nearest neighbor of x in P. Let $f'_{\le k}$ count the planes in H' with at most k conflicts. Since the EMST contains the nearest neighbor graph [74], we also have

Corollary 7.4.3. *$f'_{\le k} \ll nk$.*

Compare this with the well-known Clarkson-Shor bound of $O(nk^2)$ for the unrestricted case.

Proof of Corollary 7.4.2. As our event graph G we take the EMST of P. It is connected and has bounded degree [6, Lemma 4]. Let m be the number of edges in

G. We choose $d = 3$ and $\mu = 2$. The first thread has size two with time sequence (1), the second thread has size one. For each thread the probability of picking $v \in V$ as the initial vertex is $\deg(v)/2m$. In other words, the sampling is defined as follows: pick $v \in V$ with probability $\deg(v)/2m$ and take one random step in G. Then pick another random node v according to the same distribution. This yields a triplet of points spanning a plane in H, and each triplet appears with probability $\Theta(1/n^2)$. We have $f_0(n) \ll n$, since every plane that is spanned by a triplet in P^3 and has no conflicts supports a facet of the lower convex hull of P, and since the number of such facets is $O(n)$ and each facet is supported by exactly one plane. Thus, by Lemma 7.3.2, the probability of sampling a plane in conflict with at most k points is $O((n/k)(k/n)^2) = O(k/n)$. Since every plane is sampled with probability $\Omega(1/n^2)$, the claim follows.

Technically, Lemma 7.3.2 applies only to regular graphs, while G has bounded, but possibly varying, degree. However, our discussion easily generalizes to the non-regular case—at a loss of only a constant factor. We will show this in Lemma 7.4.4 below. □

Lemma 7.4.4. *Let G be a connected graph with n nodes, m edges, and degree bounded by r, and let f_0 be monotonically increasing. We define μ-threads as in Lemma 7.3.2, the only difference being that the initial node of each thread is sampled according to the distribution π with $\pi_v = \deg(v)/2m$. For $k > 0$, any μ-thread, and any corresponding time sequence, we have $g^{(\mu)}_{\le k} \ll (k/n)^\mu f_0(n/k)$.*

Proof. Consider the proof of Lemma 7.3.2. We may assume that $k \le n/2dr$. Set $s = n/dkr$. Then Equation (7.11) still holds, since for a given $\mathbf{u} \in V^d$, the probability $p_\mathbf{u}$ that $\mathbf{u}$ is chosen by a random μ-thread is now of the form $\prod_{1\le i\le\mu} \frac{d_i p_{\mathbf{u},i}}{2m}$, where d_i is the degree of the node in $\mathbf{u}$ corresponding to the initial vertex of the i-th thread, and since by our choice of s we have $d_i p_{\mathbf{u},i} s/2m \le r p_{\mathbf{u},i} s/2(n-1) \le 1$.

Next, we need to bound the probability that a configuration $\mathbf{u}$ with $|S_\mathbf{u}| \le n/2dr$ is active, given that $\mathbf{u} \in R$. Since we sample according to the stationary distribution of G, each node of a μ-thread lies in $S_\mathbf{u}$ with probability at most $r|S_\mathbf{u}|/2m \le r|S_\mathbf{u}|/n$. Proceeding as before, we now get

$$\Pr[\, R^v \cap S_\mathbf{u} = \emptyset \,|\, \mathbf{u} \in R\,] \;\ge\; \exp(-2d(s-1)\mu r|S_\mathbf{u}|/n).$$

and

$$r_\mathbf{u} \gg \left(\frac{n}{dkr}\right)^\mu p_\mathbf{u} \exp(-2d|S_\mathbf{u}|/k).$$

Thus, as before,

$$\sum_{\mathbf{u}\colon |S_\mathbf{u}| \le n/2dr} r_\mathbf{u} \gg \left(\frac{n}{k}\right)^\mu g^{(\mu)}_{\le k}$$

and

$$\sum_{\mathbf{u}\colon |S_{\mathbf{u}}|\le n/2dr} r_{\mathbf{u}} \le f_0(ds) = f_0(n/rk) \le f_0(n/k),$$

since f_0 is monotone. This finishes the extension of Lemma 7.3.2 to the bounded degree case. □

Chapter 8

Conclusions

In the preceding chapters we have seen many different ways how low entropy can impact the design and analysis of geometric algorithms. However, we believe that these results only constitute a few first steps in this direction, and that there remain many interesting questions to be explored. Let us mention some examples.

Hereditary algorithms. First, note that all the algorithms that appear in Chapter 3 are *randomized.* It remains an intriguing open problem to find deterministic analogues for them. In fact, any $o(n \log n)$ time deterministic algorithm for hereditary Delaunay triangulations would already be interesting. The problem with derandomizing the splitting algorithm lies in our use of backwards analysis: we know that when we delete a random blue point from the bichromatic hull, it takes constant expected time to reinsert it into the recursively computed blue hull. However, if a deterministic algorithm chooses an arbitrary blue point, the reinsertion time might be as bad as linear, and such a strategy could result in a quadratic running time. Thus, if we want to derandomize the algorithm in Chapter 3, we need to be able to identify blue points that can be reinserted efficiently, or we need to find a quick way to delete and reinsert the blue points in batches. Timothy Chan [38] gave an algorithm for splitting (arbitrary) triangulations that uses planar separators, and it would be nice to see whether it can be adapted to the Delaunay case. The approach is based on the algorithm by Bar-Yehuda and Chazelle [19] that we generalized to Delaunay triangulations in Section 3.5. By Chan's method, if we could find a deterministic version of the algorithm in Section 3.5, this would also lead to an improved deterministic Delaunay splitting algorithm.

Another interesting problem is splitting a convex polytope into more than two parts. In Corollary 3.3.6, we saw an algorithm that runs in $O(n(\log\log n)^2)$ time, irrespective of the number of desired parts. The $(\log\log n)^2$-factor comes from the recursion and the bootstrapping, and there is little reason to believe that this running time is best possible. Therefore, it would be interesting to know whether a different approach could lead to a linear-time algorithm, or whether any superlinear lower bound for the problem can be established.

Self-improving algorithms. Although we now have self-improving algorithms for several interesting problems, it still remains wide open how far the self-improving paradigm can be pushed. For simple problems, like sorting and DTs, we have a general scheme which may extend to similar problems, with the caveat that we need many advanced tools like entropy-optimal data structures and hereditary algorithms for our approach to work. In joint work with Clarkson and Seshadhri [57] we also managed to obtain an optimal self-improving algorithm for planar convex hulls. The main new issue here is *output-sensitivity*: in a self-improving algorithm for convex hulls, not all points are equal. Some points lie deep inside the convex hull, while others are close to the boundary. The algorithm needs a way to detect this and focus on the points which are likely to be extremal (ie, on the hull). This intuition can be implemented using some quite elaborate conceptual work and sophisticated data structures, overall resulting in a quite complicated algorithm. It seems highly desirable to have a simpler algorithm that obtains the same (or a slightly weaker) result. Furthermore, so far our algorithm is confined to the plane. Can it be extended to three dimensions?

This result also raises a more philosophical question: usually, when we evaluate the performance of an output-sensitive algorithm for convex hulls, we focus on the number of points that appear on the resulting hull. However, in our context it turns out that this notion is not enough, because points inside the hull, but close to the boundary, matter almost as much as the extremal points. This forces us to consider a different notion of output-sensitivity, and it might be interesting to explore this concept further.

And, of course, it remains a big open-ended question to explore what other problems can be solved in the self-improving regime.

Transdichotomous algorithms. We saw that planar Delaunay triangulations can be found in time $O(n\,\texttt{sort}(n))$ on a word RAM by using a specialized algorithm that exploits the specific properties of Delaunay triangulations. However, for the more general problem of finding convex hulls in $\mathbb{R}^3$, we do not know anything better than Chan and Pătraşcu's algorithm, which takes $n2^{O(\sqrt{\log\log n})}$ time. Does this mean that the transdichotomous model offers a finer distinction between these two problems, giving a justification for the feeling that three-dimensional convex hulls are harder than planar DTs, even though both have $\Theta(n \log n)$ complexity in the algebraic decision tree model? Such a result would be very intriguing, especially considering that the same intuition also suggests that there should be a complexity gap between sorting and planar DTs, which is disproved by the results in Chapter 5. Is this just a coincidence? Why should there be a difference between DTs and convex hulls, but not between DTs and sorting? So is it more likely that a faster convex hull algorithm exists? How should it look like?

Finally, despite the recent progress, we still have only very few transdichotomous algorithms for general (non-orthogonal) geometric problems. Can we identify any other problems that can benefit from the word RAM model? The method by Chan and Pătraşcu is based on planar point location, but our work suggests that this avenue might not always be optimal. Can we make this rigorous by proving non-trivial lower bounds for the planar point location problem (either offline of online)? Such a result seems out of reach for current techniques for data structure lower bounds.

Restricted input models. As we saw in Chapter 6, the problem of preprocessing a restricted point set for faster Delaunay triangulation is essentially solved, with relatively simple and optimal algorithms for a wide range of input regions. However, it would still be nice to have a deterministic—possibly more complicated—algorithm for general input regions (which takes us back to the problem of finding a deterministic algorithm for hereditary planar DTs).

Furthermore, the general concept of preprocessing restricted point sets in order to find a certain structure more efficiently later on seems to have potential, and there are several candidate problems that merit further investigation. For example, we can ask whether we can handle planar convex hulls for more general input regions than the ones given by the algorithm for restricted DTs, because in this case the lower bounds do not apply any longer. In that setting, can we get better trade-offs for regions that are not fat? Other problems to consider in this context would be again three-dimensional convex hulls convex hulls, or planar trapezoidal decompositions. For example, can we preprocess a set of disjoint unit spheres in 3-space such that the convex hull of a point set with exactly one point from each sphere can be found in $o(n \log n)$ time? What would be the right notion for restricted trapezoidal decompositions?

Finally, there are some similarities between restricted input models and *kinetic* data structures [20]. Can we leverage kinetic techniques for designing algorithms for restricted input models? And conversely, can some of the methods we have developed help with designing better kinetic data structures?

Markov incremental constructions. Although our result in Chapter 7 needs sophisticated machinery and reveals some non-obvious properties of randomized incremental constructions and random sources with low local entropy, it still falls short of giving an optimal result. Therefore, the most pressing open question in Markov incremental constructions is to remove the additional $\log n$ factors in the running time: can this be done, or is there an inherent price we need to pay for the lack of perfect randomness? Would it help to require additional properties of the event graph (for example, high girth)? For this, it seems we would need a much better understanding of the short term behavior of random walks and of the kinds of geometric configurations that could occur. And what about backwards analysis? Is it irretrievably lost, or is there an analogous concept in the Markov setting, hopefully leading to a simplified analysis?

Furthermore, in Chapter 7 we have only considered the *semi-dynamic* setting in which points are inserted, but not deleted. Can we say anything interesting about a model that allows both insertions and deletions? It appears that to answer that question we need to be much more careful in analyzing the active subsets that can occur in a given node of the event graph.

Bibliography

[1] A. Aggarwal. Lecture notes in computational geometry. *MIT Research Seminar Series MIT/LCS/RSS*, 3, August 1988.

[2] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4(6):591–604, 1989.

[3] N. Ailon, B. Chazelle, K. L. Clarkson, D. Liu, W. Mulzer, and C. Seshadhri. Self-improving algorithms. Manuscript at `arXiv:0907.0884`, 2009.

[4] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Self-improving algorithms. In *Proc. 17th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 261–270, 2006.

[5] S. Albers and T. Hagerup. Improved parallel integer sorting without concurrent writing. *Inform. and Comput.*, 136(1):25–51, 1997.

[6] D. Aldous and J. M. Steele. Asymptotics for Euclidean minimal spanning trees on random points. *Probab. Theory Related Fields*, 92(2):247–258, 1992.

[7] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Linear-time triangulation of a simple polygon made easier via randomization. In *Proc. 16th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 201–212, 2000.

[8] N. Amenta, D. Attali, and O. Devillers. Complexity of Delaunay triangulation for points on lower-dimensional polyhedra. In *Proc. 18th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1106–1113, 2007.

[9] N. Amenta, S. Choi, and G. Rote. Incremental constructions con BRIO. In *Proc. 19th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 211–219, 2003.

[10] A. Amir, A. Efrat, P. Indyk, and H. Samet. Efficient regular data structures and algorithms for dilation, location, and proximity problems. *Algorithmica*, 30(2):164–187, 2001.

[11] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman. Sorting in linear time? *J. Comput. System Sci.*, 57(1):74–93, 1998.

[12] A. Andersson, P. B. Miltersen, and M. Thorup. Fusion trees can be implemented with AC^0 instructions only. *Theoret. Comput. Sci.*, 215(1–2):337–344, 1999.

[13] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.

[14] S. Arora and B. Barak. *Computational complexity: A Modern Approach.* Cambridge University Press, 2009.

[15] S. Arya, T. Malamatos, and D. M. Mount. A simple entropy-based algorithm for planar point location. *ACM Trans. Algorithms*, 3(2):Art. 17, 17 pp., 2007.

[16] S. Arya, T. Malamatos, D. M. Mount, and K. C. Wong. Optimal expected-case planar point location. *SIAM J. Comput.*, 37(2):584–610 (electronic), 2007.

[17] D. Attali and J.-D. Boissonnat. A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. *Discrete Comput. Geom.*, 31(3):369–384, 2004.

[18] D. Bandyopadhyay and J. Snoeyink. Almost-Delaunay simplices: Nearest neighbor relations for imprecise points. In *Proc. 15th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 403–412, 2004.

[19] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *Internat. J. Comput. Geom. Appl.*, 4(4):475–481, 1994.

[20] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.

[21] K. E. Batcher. Sorting networks and their applications. In *Proc. AFIPS Spring Joint Computer Conferences*, pages 307–314, 1968.

[22] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 16th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 80–86, 1983.

[23] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: Algorithms and Applications.* Springer-Verlag, Berlin, third edition, 2008.

[24] M. de Berg, H. David, M. J. Katz, M. H. Overmars, A. F. van der Stappen, and J. Vleugels. Guarding scenes against invasive hypercubes. *Comput. Geom. Theory Appl.*, 26(2):99–117, 2003.

[25] M. de Berg, M. van Kreveld, and J. Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *J. Algorithms*, 18(2):256–277, 1995.

[26] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comput. System Sci.*, 48(3):384–409, 1994.

[27] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *Internat. J. Comput. Geom. Appl.*, 9(6):517–532, 1999.

[28] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec. Applications of random sampling to on-line algorithms in computational geometry. *Discrete Comput. Geom.*, 8(1):51–71, 1992.

[29] J.-D. Boissonnat and M. Teillaud. The hierarchical representation of objects: the Delaunay tree. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 260–268, 1986.

[30] J.-D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoret. Comput. Sci.*, 112(2):339–354, 1993.

[31] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry.* Cambridge University Press, 1998.

[32] A. Z. Broder and A. R. Karlin. Bounds on the cover time. *J. Theoret. Probab.*, 2(1):101–120, 1989.

[33] R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory Comput. Syst.*, 38(4):411–423, 2005.

[34] K. Buchin. *Organizing Point Sets: Space-Filling Curves, Delaunay Tessellations of Random Point Sets, and Flow Complexes.* PhD thesis, Freie Universität Berlin, 2007. `http://www.diss.fu-berlin.de/diss/receive/FUDISS_thesis_000000003494`.

[35] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995.

[36] T. M. Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM J. Comput.*, 30(2):561–575, 2000.

[37] T. M. Chan. Closest-point problems simplified on the RAM. In *Proc. 13th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 472–473, 2002.

[38] T. M. Chan. Three problems about simple polygons. *Comput. Geom. Theory Appl.*, 35(3):209–217, 2006.

[39] T. M. Chan. Well-separated pair decomposition in linear time? *Inform. Process. Lett.*, 107(5):138–141, 2008.

[40] T. M. Chan and M. Pătraşcu. Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. *SIAM J. Comput.*, 39(2):703–729, 2009.

[41] T. M. Chan and M. Pătraşcu. Voronoi diagrams in $n2^{O(\sqrt{\lg\lg n})}$ time. In *Proc. 39th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 31–39, 2007.

[42] P. Chassaing. Optimality of move-to-front for self-organizing data structures with locality of references. *Ann. Appl. Probab.*, 3(4):1219–1240, 1993.

[43] B. Chazelle. Filtering search: a new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.

[44] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[45] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[46] B. Chazelle. *The discrepancy method: randomness and complexity.* Cambridge University Press, New York, NY, USA, 2000.

[47] B. Chazelle, O. Devillers, F. Hurtado, M. Mora, V. Sacristán, and M. Teillaud. Splitting a Delaunay triangulation in linear time. *Algorithmica*, 34(1):39–46, 2002.

[48] O. Cheong, K. Mulmuley, and E. A. Ramos. Randomization and derandomization. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, chapter 40, pages 895–926. CRC Press, Inc., Boca Raton, FL, USA, 2nd edition, 2004.

[49] L. P. Chew. Building Voronoi Diagrams for Convex Polygons in Linear Expected Time. Technical Report PCS-TR90-147, Dartmouth College, Computer Science, Hanover, NH, 1990.

[50] L. P. Chew and S. Fortune. Sorting helps for Voronoi diagrams. *Algorithmica*, 18(2):217–228, 1997.

[51] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21(3):405–420, 1999.

[52] F. Chin and C. A. Wang. Finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time. *SIAM J. Comput.*, 28(2):471–486, 1998.

[53] F. R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92).* American Mathematical Society, Providence, RI, USA, 1997.

[54] K. L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 226–232, 1983.

[55] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

[56] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. *Comput. Geom. Theory Appl.*, 3(4):185–212, 1993.

[57] K. L. Clarkson, W. Mulzer, and C. Seshadhri. Self-improving algorithms for convex hulls. In *Proc. 21st Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 1546–1565, 2010.

[58] K. L. Clarkson and C. Seshadhri. Self-improving algorithms for Delaunay triangulations. In *Proc. 24th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 148–155, 2008.

[59] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry. II. *Discrete Comput. Geom.*, 4(5):387–421, 1989.

[60] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete Comput. Geom.*, 37(1):43–58, 2007.

[61] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, third edition, 2009.

[62] T. M. Cover and J. A. Thomas. *Elements of information theory.* Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, second edition, 2006.

[63] O. Devillers. Randomization yields simple $O(n \log^* n)$ algorithms for difficult $\Omega(n)$ problems. *Internat. J. Comput. Geom. Appl.*, 2(1):97–111, 1992.

[64] O. Devillers. The Delaunay hierarchy. *Internat. J. Found. Comput. Sci.*, 13:163–180, 2002.

[65] O. Devillers and P. Guigue. The shuffling buffer. *Internat. J. Comput. Geom. Appl.*, 11(5):555–572, 2001.

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

[67] H. N. Djidjev and A. Lingas. On computing Voronoi diagrams for sorted point sets. *Internat. J. Comput. Geom. Appl.*, 5(3):327–337, 1995.

[68] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoret. Comput. Sci.*, 27(3):241–253, 1983.

[69] D. P. Dobkin and D. G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6(3):381–392, 1985.

[70] K. Dobrindt and M. Yvinec. Remembering conflicts in history yields dynamic algorithms. In *Proc. 4th Annu. Internat. Sympos. Algorithms Comput. (ISAAC)*, pages 21–30, 1993.

[71] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inform. Process. Lett.*, 6(3):80–82, 1977.

[72] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Math. Systems Theory*, 10(2):99–127, 1976/77.

[73] D. Eppstein. Approximating the minimum weight Steiner triangulation. *Discrete Comput. Geom.*, 11(2):163–191, 1994.

[74] D. Eppstein, M. S. Paterson, and F. F. Yao. On nearest-neighbor graphs. *Discrete Comput. Geom.*, 17(3):263–282, 1997.

[75] R. A. Finkel and J. L. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[76] H. Fournier and A. Vigneron. A tight lower bound for computing the diameter of a 3D convex polytope. *Algorithmica*, 49(3):245–257, 2007.

[77] D. Fox, W. Burgard, and S. Thrun. Markov localization for reliable robot navigation and people detection. In *Selected Papers from the International Workshop on Sensor Based Intelligent Robots*, pages 1–20, London, UK, 1999. Springer-Verlag.

[78] P. G. Franciosa, C. Gaibisso, G. Gambosi, and M. Talamo. A convex hull algorithm for points with approximately known positions. *Internat. J. Comput. Geom. Appl.*, 4(2):153–163, 1994.

[79] M. L. Fredman. How good is the information theory bound in sorting? *Theoret. Comput. Sci.*, 1(4):355–361, 1975/76.

[80] M. L. Fredman and D. E. Willard. Surpassing the information theoretic bound with fusion trees. *J. Comput. System Sci.*, 47(3):424–436, 1993.

[81] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. System Sci.*, 48(3):533–551, 1994.

[82] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Systems Theory*, 17(1):13–27, 1984.

[83] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 135–143, 1984.

[84] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7(4):381–413, 1992.

[85] L. J. Guibas, D. Salesin, and J. Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *Proc. 5th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 208–217, 1989.

[86] L. J. Guibas, D. Salesin, and J. Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, 9:534–560, 1993.

[87] Y. Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004.

[88] Y. Han and M. Thorup. Integer sorting in $O(n\sqrt{\log \log n})$ expected time and linear space. In *Proc. 43rd Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 135–144, 2002.

[89] M. Held and J. S. B. Mitchell. Triangulating input-constrained planar point sets. *Inform. Process. Lett.*, 109(1):54–56, 2008.

[90] R. A. Horn and C. R. Johnson. *Matrix analysis.* Cambridge University Press, Cambridge, 1990.

[91] G. Hotz. Search trees and search graphs for Markov sources. *Elektronische Informationsverarbeitung und Kybernetik*, 29(5):283–292, 1993.

[92] J. Iacono and S. Langerman. Dynamic point location in fat hyperrectangles with integer coordinates. In *Proc. 12th Canad. Conf. Comput. Geom. (CCCG)*, pages 181–186, 2000.

[93] H. W. Jensen. *Realistic image synthesis using photon mapping.* A K Peters Ltd., Natick, MA, 2001.

[94] S. Kapoor and E. M. Reingold. Stochastic rearrangement rules for self-organizing data structures. *Algorithmica*, 6(2):278–291, 1991.

[95] A. R. Karlin, S. J. Phillips, and P. Raghavan. Markov paging. *SIAM J. Comput.*, 30(3):906–922, 2000.

[96] R. G. Karlsson. *Algorithms in a restricted universe.* PhD thesis, University of Waterloo, 1985.

[97] R. G. Karlsson and M. H. Overmars. Scanline algorithms on a grid. *BIT*, 28(2):227–241, 1988.

[98] D. Kirkpatrick and S. Reisch. Upper bounds for sorting integers on random access machines. *Theoret. Comput. Sci.*, 28(3):263–276, 1984.

[99] D. G. Kirkpatrick, M. M. Klawe, and R. E. Tarjan. Polygon triangulation in $O(n \log \log n)$ time with simple data structures. *Discrete Comput. Geom.*, 7(4):329–346, 1992.

[100] R. Klein and A. Lingas. A linear-time randomized algorithm for the bounded Voronoi diagram of a simple polygon. *Internat. J. Comput. Geom. Appl.*, 6(3):263–278, 1996.

[101] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 3rd edition, 1997.

[102] D. E. Knuth. *The art of computer programming: sorting and searching*, volume 3. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2nd edition, 1998.

[103] A. N. Kolmogorov. On the notion of algorithm. *Uspekhi Mat. Nauk.*, 8:175–176, 1953.

[104] L. K. Konneker and Y. L. Varol. A note on heuristics for dynamic organization of data structures. *Inform. Process. Lett.*, 12(5):213–216, 1981.

[105] M. van Kreveld and M. Löffler. Largest bounding box, smallest diameter, and related problems on imprecise points. In *Proc. 10thWorkshop on Algorithms and Data Structures (WADS)*, pages 447–458, 2007.

[106] M. J. van Kreveld, M. Löffler, and J. S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. In *Proc. 19th Annu. Internat. Sympos. Algorithms Comput. (ISAAC)*, pages 544–555, 2008.

[107] H. Kruger. Basic measures for imprecise point sets in $\mathbb{R}^d$. Master's thesis, Utrecht University, 2008.

[108] K. Lam, M. Y. Leung, and M. K. Siu. Self-organizing files with dependent accesses. *J. Appl. Probab.*, 21(2):343–359, 1984.

[109] J. van Leeuwen and A. Tsakalides. An optimal pointer machine algorithm for finding nearest common ancestors. Technical Report RUU-CS-88-17, Department of Information and Computing Sciences, Utrecht University, 1988.

[110] M. Löffler and J. Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom. Theory Appl.*, 43(3):234–242, 2010.

[111] P. Lu, X. Zeng, X. Huang, and Y. Wang. Navigation in 3D game by Markov model based head pose estimating. In *Proc. Third International Conference on Image and Graphics (ICIG)*, pages 493–496, 2004.

[112] J. Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002.

[113] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Comput. Geom. Theory Appl.*, 16(4-5):498–516, 1996.

[114] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer-Verlag, 1984.

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

[116] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, 1997.

[117] M. Mitzenmacher and E. Upfal. *Probability and computing*. Cambridge University Press, Cambridge, 2005.

[118] G. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., Ottawa, Canada, 1966.

[119] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995.

[120] K. Mulmuley. A fast planar partition algorithm. I. *J. Symbolic Comput.*, 10(3-4):253–280, 1990.

[121] K. Mulmuley. A fast planar partition algorithm. II. *J. ACM*, 38(1):74–103, 1991.

[122] K. Mulmuley. *Computational Geometry: An Introduction through Randomized Algorithms.* Prentice-Hall, Englewood Cliffs, 1994.

[123] K. Mulmuley. Randomized geometric algorithms and pseudorandom generators. *Algorithmica*, 16(4-5):450–463, 1996.

[124] T. Nagai and N. Tokura. Tight error bounds of geometric problems on convex objects with imprecise coordinates. In *Jap. Conf. on Discrete and Comput. Geom.*, pages 252–263, 2000.

[125] Y. Ostrovsky-Berman and L. Joskowicz. Uncertainty envelopes. In *Proc. 21st European Workshop Comput. Geom. (EWCG)*, pages 175–178, 2005.

[126] M. H. Overmars. Computational geometry on a grid: An overview. Technical Report RUU-CS-87-04, Rijksuniversiteit Utrecht, 1987.

[127] R. M. Phatarfod, A. J. Pryde, and D. Dyte. On the move-to-front scheme with Markov dependent requests. *J. Appl. Probab.*, 34(3):790–794, 1997.

[128] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Texts and Monographs in Computer Science. Springer-Verlag, 1985.

[129] R. Raman. Priority queues: Small, monotone and trans-dichotomous. In *Proc. 4th Annu. European Sympos. Algorithms (ESA)*, pages 121–137, 1996.

[130] E. A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. 15th Annu. ACM Sympos. Comput. Geom. (SoCG)*, pages 390–399, 1999.

[131] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 3rd edition, 2009.

[132] A. Schönhage. On the power of random access machines. In *Proc. 6th Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 520–529, 1979.

[133] A. Schönhage. Storage modification machines. *SIAM J. Comput.*, 9(3):490–508, 1980.

[134] F. Schulz and E. Schömer. Self-organizing data structures with dependent accesses. In *Proc. 23rd Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 526–537, 1996.

[135] O. Schwarzkopf. Dynamic maintenance of geometric structures made easy. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 197–206, 1991.

[136] R. Sedgewick and M. Schidlowsky. *Algorithms in Java, Third Edition, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[137] R. Seidel. A method for proving lower bounds for certain geometric problems. Technical Report TR84-592, Cornell University, Ithaca, NY, USA, 1984.

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

[139] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete Comput. Geom.*, 6(5):423–434, 1991.

[140] R. Seidel. Backwards analysis of randomized geometric algorithms. In *New trends in discrete and computational geometry*, volume 10 of *Algorithms Combin.*, pages 37–67. Springer, Berlin, 1993.

[141] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16(4–5):464–497, 1996.

[142] G. S. Shedler and C. Tung. Locality in page reference strings. *SIAM J. Comput.*, 1(3):218–241, 1972.

[143] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. System Sci.*, 18(2):110–127, 1979.

[144] M. Thorup. Faster deterministic sorting and priority queues in linear space. In *Proc. 9th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 550–555, 1998.

[145] M. Thorup. On AC^0 implementations of fusion trees and atomic heaps. In *Proc. 14th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 699–707, 2003.

[146] E. Veach and L. J. Guibas. Metropolis light transport. In *Proc. 24th annual conference on Computer graphics and interactive techniques (SIGGRPAH)*, pages 65–76, 1997.

[147] C. Wellington, A. Courville, and A. T. Stentz. A generative model of terrain for autonomous navigation in vegetation. *Int. J. Rob. Res.*, 25(12):1287–1304, 2006.

[148] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science (Graz, 1991)*, volume 555 of *Lecture Notes in Comput. Sci.*, pages 359–370. Springer, Berlin, 1991.

[149] D. E. Willard. Examining computational geometry, van Emde Boas trees, and hashing from the perspective of the fusion tree. *SIAM J. Comput.*, 29(3):1030–1049, 2000.