Computational Geometry
Lecture Notes[1] HS 2012

Bernd Gärtner <gaertner@inf.ethz.ch>
Michael Hoffmann <hoffmann@inf.ethz.ch>

Tuesday 15th January, 2013

[1]Some parts are based on material provided by Gabriel Nivasch and Emo Welzl. We also thank Tobias Christ, Anna Gundert, and May Szedlák for pointing out errors in preceding versions.

Contents

1 Fundamentals **7**

1.1 Models of Computation 7

1.2 Basic Geometric Objects 8

2 Polygons **11**

2.1 Classes of Polygons 11

2.2 Polygon Triangulation 15

2.3 The Art Gallery Problem 19

3 Convex Hull **23**

3.1 Convexity 24

3.2 Planar Convex Hull 27

3.3 Trivial algorithms 29

3.4 Jarvis' Wrap 29

3.5 Graham Scan (Successive Local Repair) 31

3.6 Lower Bound 33

3.7 Chan's Algorithm 33

4 Line Sweep **37**

4.1 Interval Intersections 38

4.2 Segment Intersections 38

4.3 Improvements 42

4.4 Algebraic degree of geometric primitives 42

4.5 Red-Blue Intersections 45

5 Plane Graphs and the DCEL **51**

5.1 The Euler Formula 52

5.2 The Doubly-Connected Edge List 53

5.2.1 Manipulating a DCEL 54

5.2.2 Graphs with Unbounded Edges 57

5.2.3 Remarks 58

6 Delaunay Triangulations **61**
6.1 The Empty Circle Property . . . 64
6.2 The Lawson Flip algorithm . . . 66
6.3 Termination of the Lawson Flip Algorithm: The Lifting Map . . . 67
6.4 Correctness of the Lawson Flip Algorithm . . . 68
6.5 The Delaunay Graph . . . 70
6.6 Every Delaunay Triangulation Maximizes the Smallest Angle . . . 72
6.7 Constrained Triangulations . . . 75

7 Delaunay Triangulation: Incremental Construction **79**
7.1 Incremental construction . . . 79
7.2 The History Graph . . . 82
7.3 The structural change . . . 83

8 The Configuration Space Framework **85**
8.1 The Delaunay triangulation — an abstract view . . . 85
8.2 Configuration Spaces . . . 86
8.3 Expected structural change . . . 87
8.4 Bounding location costs by conflict counting . . . 89
8.5 Expected number of conflicts . . . 90

9 Trapezoidal Maps **95**
9.1 The Trapezoidal Map . . . 95
9.2 Applications of trapezoidal maps . . . 96
9.3 Incremental Construction of the Trapezoidal Map . . . 96
9.4 Using trapezoidal maps for point location . . . 98
9.5 Analysis of the incremental construction . . . 99
9.5.1 Defining The Right Configurations . . . 99
9.5.2 Update Cost . . . 102
9.5.3 The History Graph . . . 103
9.5.4 Cost of the **Find** step . . . 104
9.5.5 Applying the General Bounds . . . 104
9.6 Analysis of the point location . . . 106
9.7 The trapezoidal map of a simple polygon . . . 108

10 Voronoi Diagrams **115**
10.1 Post Office Problem . . . 115
10.2 Voronoi Diagram . . . 116
10.3 Duality . . . 119
10.4 Lifting Map . . . 120
10.5 Point location in a Voronoi Diagram . . . 121
10.5.1 Kirkpatrick's Hierarchy . . . 122

11 Linear Programming **129**
11.1 Linear Separability of Point Sets . . . 129
11.2 Linear Programming . . . 130
11.3 Minimum-area Enclosing Annulus . . . 133
11.4 Solving a Linear Program . . . 135

12 A randomized Algorithm for Linear Programming **137**
12.1 Helly's Theorem . . . 138
12.2 Convexity, once more . . . 139
12.3 The Algorithm . . . 140
12.4 Runtime Analysis . . . 141
12.4.1 Violation Tests . . . 142
12.4.2 Basis Computations . . . 143
12.4.3 The Overall Bound . . . 143

13 Line Arrangements **145**
13.1 Arrangements . . . 146
13.2 Construction . . . 147
13.3 Zone Theorem . . . 147
13.4 The Power of Duality . . . 149
13.5 Sorting all Angular Sequences. . . . 150
13.6 Segment Endpoint Visibility Graphs . . . 151
13.7 Ham Sandwich Theorem . . . 153
13.8 3-Sum . . . 155
13.9 3-Sum hardness . . . 156

14 Davenport-Schinzel Sequences **161**
14.1 Davenport-Schinzel Sequences . . . 162

15 Epsilon Nets **167**
15.1 Motivation . . . 167
15.2 Range spaces and ε-nets. . . . 168
15.2.1 Examples . . . 168
15.2.2 No point in a large range. . . . 169
15.2.3 Smallest enclosing balls. . . . 170
15.3 Either almost all is needed or a constant suffices. . . . 170
15.4 What makes the difference: VC-dimension . . . 171
15.4.1 The size of projections for finite VC-dimension. . . . 172
15.5 VC-dimension of Geometric Range Spaces . . . 174
15.5.1 Halfspaces. . . . 174
15.5.2 Balls. . . . 175
15.6 Small ε-Nets, an Easy Warm-up Version . . . 176
15.6.1 Smallest enclosing balls, again . . . 177

15.7 Even Smaller ε-Nets 178

Chapter 1

Fundamentals

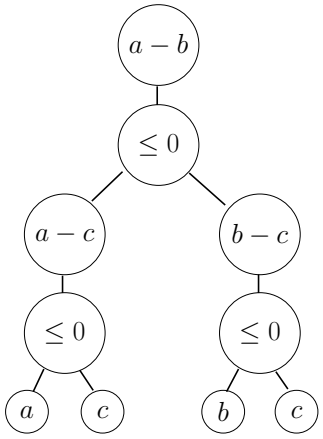
1.1 Models of Computation

Real RAM Model. A memory cell stores a real number. Any single arithmetic operation or comparison can be computed in constant time. In addition, sometimes also roots, logarithms, other analytic functions, indirect addressing (integral), or floor and ceiling are used.

This is a quite powerful (and somewhat unrealistic) model of computation, as a single real number in principle can encode an arbitrary amount of information. Therefore we have to ensure that we do not abuse the power of this model.

Algebraic Computation Trees (Ben-Or [1]). A computation is regarded as a binary tree.

- The leaves contain the (possible) results of the computation.
- Every node v with one child has an operation of the form $+, -, *, /, \sqrt{}, \ldots$ associated to it. The operands of this operation are constant input values, or among the ancestors of v in the tree.
- Every node v with two children has associated to it a branching of the form > 0, $\geqslant 0$, or $= 0$. The branch is with respect to the result of v's parent node. If the expression yields `true`, the computation continues with the left child of v; otherwise, it continues with the right child of v.



If every branch is based on a linear function in the input values, we face a *linear computation tree*. Analogously one can define, say, quadratic computation trees. The term *decision tree* is used if all of the results are either `true` or `false`.

The complexity of a computation or decision tree is the maximum number of vertices along any root-to-leaf path. It is well known that $\Omega(n \log n)$ comparisons are required to sort n numbers. But also for some problems that appear easier than sorting at first glance, the same lower bound holds. Consider, for instance, the following problem.

Element Uniqueness

Input: $\{x_1, \ldots, x_n\} \subset \mathbb{R}$, $n \in \mathbb{N}$.

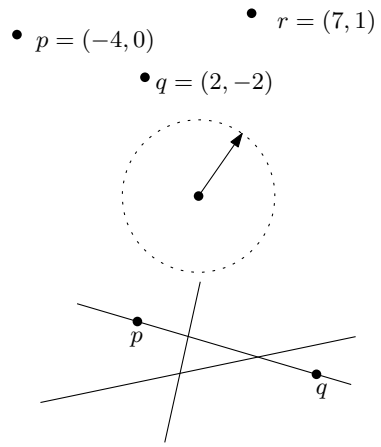
Output: Is $x_i = x_j$, for some $i, j \in \{1, \ldots, n\}$ with $i \neq j$?

Ben-Or [1] has shown that any algebraic decision tree to solve Element Uniqueness for n elements has complexity $\Omega(n \log n)$.

1.2 Basic Geometric Objects

We will mostly be concerned with the d-dimensional Euclidean space $\mathbb{R}^d$, for small $d \in \mathbb{N}$; typically, $d = 2$ or $d = 3$. The basic objects of interest in $\mathbb{R}^d$ are the following.

Points. A point p, typically described by its d Cartesian coordinates $p = (x_1, \ldots, x_d)$.

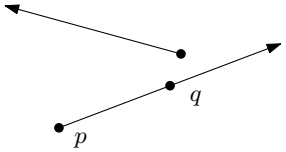


Directions. A vector $v \in \mathcal{S}^{d-1}$ (the $(d-1)$-dimensional unit sphere), typically described by its d Cartesian coordinates $v = (x_1, \ldots, x_d)$, with $\|v\| = \sqrt{\sum_{i=1}^{d} {x_i}^2} = 1$.

Lines. A line is a one-dimensional affine subspace. It can be described by two distinct points p and q as the set of all points r that satisfy $r = p + \lambda(q - p)$, for some $\lambda \in \mathbb{R}$.

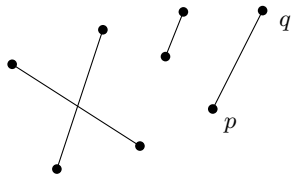
While any pair of distinct points defines a unique line, a line in $\mathbb{R}^2$ contains infinitely many points and so it may happen that a collection of three or more points lie on a line. Such a collection of points is termed *collinear*[1].

Rays. If we remove a single point from a line and take the closure of one of the connected components, then we obtain a ray. It can be described by two distinct points p and q as the set of all points r that satisfy $r = p + \lambda(q - p)$, for some $\lambda \geqslant 0$. The *orientation* of a ray is the direction $(q - p)/\|q - p\|$.



[1] Not *colinear*, which refers to a notion in the theory of coalgebras.

Line segment. A line segment is the intersection of two collinear rays of opposite orientation. It can be described by two points p and q as the set of all points r that satisfy $r = p + \lambda(q - p)$, for some $\lambda \in [0, 1]$. We will denote the line segment through p and q by $\overline{pq}$. Depending on the context we may allow or disallow *degenerate* line segments consisting of a single point only ($p = q$ in the above equation).



Hyperplanes. A hyperplane $\mathcal{H}$ is a $(d-1)$-dimensional affine subspace. It can be described algebraically by $d + 1$ coefficients $\lambda_1, \ldots, \lambda_{d+1} \in \mathbb{R}$, where $\|(\lambda_1, \ldots, \lambda_{d+1})\| = 1$, as the set of all points $(x_1, \ldots, x_d)$ that satisfy the linear equation $\mathcal{H} : \sum_{i=1}^{d} \lambda_i x_i = \lambda_{d+1}$.

If the above equation is converted into an inequality, we obtain the algebraic description of a *halfspace* (in $\mathbb{R}^2$: halfplane).

Spheres and balls. A sphere is the set of all points that are equidistant to a fixed point. It can be described by a point c (center) and a number $\rho \in \mathbb{R}$ (radius) as the set of all points p that satisfy $\|p - c\| = \rho$. The *ball* of radius ρ around p consists of all points p that satisfy $\|p - c\| \leqslant \rho$.

References

[1] Michael Ben-Or, Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pp. 80–86, 1983, URL `http://dx.doi.org/10.1145/800061.808735`.

Chapter 2

Polygons

Although we can think of a line $\ell \subset \mathbb{R}^2$ as an infinite point set that consists of all points in $\mathbb{R}^2$ that are on ℓ, there still exists a finite description for ℓ. Such a description is, for instance, provided by the three coefficients $a, b, c \in \mathbb{R}$ of an equation of the form $ax + by = c$, with $(a, b) \neq (0, 0)$. Actually this holds true for all of the fundamental geometric objects that were mentioned in the previous section: Each of them has constant *description complexity* (or, informally, just *size*), that is, it can be described by a constant[1] number of parameters.

In this course we will typically deal with objects that are not of constant size. Often these are formed by merely aggregating constant-size objects, for instance, points to form a finite set of points. But sometimes we also demand additional structure that goes beyond aggregation only. Probably the most fundamental geometric objects of this type are what we call *polygons*. You probably learned this term in school, but what *is* a polygon precisely? Consider the examples shown in Figure 2.1. Are all of these polygons? If not, where would you draw the line?

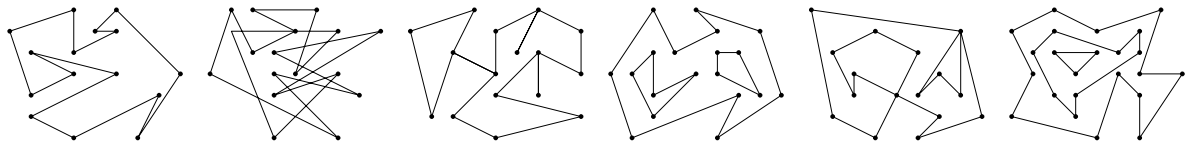


Figure 2.1: *What is a polygon?*

2.1 Classes of Polygons

Obviously, there is not *the* right answer to such a question and certainly there are different types of polygons. Often the term polygon is used somewhat sloppily in place

[1] Unless specified differently, we will always assume that the dimension is (a small) constant. In a high-dimensional space $\mathbb{R}^d$, one has to account for a description complexity of $\Theta(d)$.

of what we call a *simple polygon*, defined below.

Definition 2.1 *A* **simple polygon** *is a compact region* $P \subset \mathbb{R}^2$ *that is bounded by a simple closed curve* $\gamma : [0,1] \to \mathbb{R}^2$ *that consists of a finite number of line segments. A* **curve** *is a continuous map* $\gamma : [0,1] \to \mathbb{R}^2$*. A curve* γ *is* **closed***, if* $\gamma(0) = \gamma(1)$ *and it is* **simple** *if it is injective on* $[0,1)$*, that is, the curve does not intersect itself.*

Out of the examples shown above only Polygon 2.1a is simple. For each of the remaining polygons it is impossible to combine the bounding segments into a simple closed curve.

The term *compact* for subsets of $\mathbb{R}^d$ means bounded and closed. A subset of $P \subset \mathbb{R}^d$ is *bounded*, if it is contained in the ball of radius r around the origin, for some finite $r > 0$. Being closed means that the boundary is considered to be part of the polygon. In order to formally define these terms, let us briefly review a few basic notions from topology.

The standard topology of $\mathbb{R}^d$ is defined in terms of the Euclidean metric. A point $p \in \mathbb{R}^d$ is *interior* to a set $P \subseteq \mathbb{R}^d$, if there exists an ε-ball $B_\varepsilon(p) = \{x \in \mathbb{R}^d : \|x-p\| < \varepsilon\}$ around p, for some $\varepsilon > 0$, that is completely contained in P. A set is *open*, if all of its points are interior; and it is *closed*, if its complement is open.

Exercise 2.2 *Determine for each of the following sets whether they are open or closed in* $\mathbb{R}^2$*. a)* $B_1(0)$ *b)* $\{(1,0)\}$ *c)* $\mathbb{R}^2$ *d)* $\mathbb{R}^2 \setminus \mathbb{Z}^2$ *e)* $\mathbb{R}^2 \setminus \mathbb{Q}^2$ *f)* $\{(x,y) : x \in \mathbb{R}, y \geqslant 0\}$

Exercise 2.3 *Show that the union of countably many open sets in* $\mathbb{R}^d$ *is open. Show that the union of a finite number of closed sets in* $\mathbb{R}^d$ *is closed. (These are two of the axioms that define a topology. So the statements are needed to assert that the metric topology is a topology, indeed.) What follows for intersections of open and closed sets? Finally, show that the union of countably many closed sets in* $\mathbb{R}^d$ *is not necessarily closed.*

The *boundary* ∂P of a set $P \subset \mathbb{R}^d$ consists of all points that are neither interior to P nor to its complement $\mathbb{R}^d \setminus P$. By definition, for every $p \in \partial P$ every ball $B_\varepsilon(p)$ contains both points from P and from $\mathbb{R}^d \setminus P$. Sometimes one wants to consider a set $P \subset \mathbb{R}^d$ open although it is not. In that case one can resort to the *interior* P° of P that is formed by the subset of points interior to P. Similarly, the *closure* $\overline{P}$ of P is defined by $\overline{P} = P \cup \partial P$.

Lower-dimensional objects, such as line segments in $\mathbb{R}^2$ or triangles in $\mathbb{R}^3$, do not possess any interior point (because the ε-balls needed around any such point are full-dimensional). Whenever we want to talk about the interior of a lower-dimensional object, we use the qualifier *relative* and consider it relative to the smallest affine subspace that contains the object.

For instance, the smallest affine subspace that contains a line segment is a line and so the relative interior of a line segment in $\mathbb{R}^2$ consists of all points except the endpoints, just like for an interval in $\mathbb{R}^1$. Similarly, for a triangle in $\mathbb{R}^3$ the smallest affine subspace that contains it is a plane. Hence its relative interior is just the interior of the triangle, considered as a two-dimensional object.

Exercise 2.4 *Show that for any $P \subset \mathbb{R}^d$ the interior P° is open. (Why is there something to show to begin with?) Show that for any $P \subset \mathbb{R}^d$ the closure $\overline{P}$ is closed.*

When describing a simple polygon P it is sufficient to describe only its boundary ∂P. As ∂P by definition is a simple closed curve γ that consists of finitely many line segments, we can efficiently describe it as a sequence $p_1, \ldots, p_n$ of points, such that γ is formed by the line segments $\overline{p_1p_2}, \overline{p_2p_3}, \ldots, \overline{p_{n-1}p_n}, \overline{p_np_1}$. These points are referred to as the *vertices* of the polygon, and the segments connecting them are referred as the *edges* of the polygon.

Knowing the boundary, it is easy to tell apart the (bounded) interior from the (unbounded) exterior. This is asserted even for much more general curves by the well-known Jordan-Curve Theorem.

Theorem 2.5 (Jordan 1887) *Any simple closed curve $\gamma : [0,1] \to \mathbb{R}^2$ divides the plane into exactly two connected components whose common boundary is formed by γ.*

In full generality, the proof of the deceptively obvious claim is surprisingly difficult. We will not prove it here, the interested reader can find a proof, for instance, in the book of Mohar and Thomassen [11]. There exist different generalizations of the theorem and there also has been some debate about to which degree the original proof of Jordan is actually correct. For simple polygons the situation is easier, though. The essential idea can be worked out algorithmically, which we leave as an exercise.

Exercise 2.6 *Describe an algorithm to decide whether a point lies inside or outside of a simple polygon. More precisely, given a simple polygon $P \subset \mathbb{R}^2$ as a list of its vertices $(v_1, v_2, \ldots, v_n)$ in counterclockwise order and a query point $q \in \mathbb{R}^2$, decide whether q is inside P, on the boundary of P, or outside. The runtime of your algorithm should be $O(n)$.*

There are good reasons to ask for the boundary of a polygon to form a simple curve: For instance, in the example depicted in Figure 2.1b there are several regions for which it is completely unclear whether they should belong to the interior or to the exterior of the polygon. A similar problem arises for the interior regions in Figure 2.1f. But there are more general classes of polygons that some of the remaining examples fall into. We will discuss two such classes here. The first comprises polygons like the one from Figure 2.1d.

Definition 2.7 *A region $P \subset \mathbb{R}^2$ is a* **simple polygon with holes** *if it can be described as $P = F \setminus \bigcup_{H \in \mathcal{H}} H^\circ$, where $\mathcal{H}$ is a finite collection of pairwise disjoint simple polygons (called holes) and F is a simple polygon for which $F^\circ \supset \bigcup_{H \in \mathcal{H}} H$.*

The way this definition heavily depends on the notion of simple polygons makes it straightforward to derive a similar trichotomy as the Jordan Curve Theorem provides for simple polygons, that is, every point in the plane is either inside, or on the boundary, or outside of P (exactly one of these three).

The second class describes polygons that are "almost-simple" in the sense that they are arbitrarily close to a simple polygon. In many algorithmic scenarios such polygons can be treated very similarly to simple polygons.

Definition 2.8 *A* **weakly simple polygon** *is a bounded region* $P \subset \mathbb{R}^2$ *such that*

1. P *is connected,* ∂P *is connected, and* $\overline{P}$ *is simply-connected;*
2. ∂P *consists of a finite number of line segments, no two of which intersect except at a common endpoint; and*
3. *there exists a* $k \in \mathbb{N}$ *such that for every* $\varepsilon > 0$ *there exists a simple polygon* Q_ε *on at most* k *vertices for which the symmetric difference* $(Q_\varepsilon \cup P) \setminus (Q_\varepsilon \cap P)$ *has area less than* ε*.*

It remains to define the terms connected and simply-connected. A set $P \subseteq \mathbb{R}^d$ is *connected*[2] if for every pair $p, q \in P$ there is a curve within P that connects p and q. A set $P \subseteq \mathbb{R}^d$ is *simply-connected* if it is connected and if for every simple closed curve $\gamma : [0,1] \to P$ the bounded region enclosed by γ (well-defined by Theorem 2.5) is completely contained in P. For instance, the simple polygon with one hole depicted in Figure 2.2a is not simply-connected, because the red curve around the hole encloses points that do not belong to the polygon. The polygon P shown in Figure 2.2b is not simple but weakly simple; the simple polygon shown in orange provides a pretty good approximation that can be made arbitrarily close (but soon would be indistinguishable from P).

Figure 2.2: *Weakly simple or not?*

Exercise 2.9 *Which of the shapes depicted in Figure 2.1 are weakly simple polygons?*

Exercise 2.10 *Show that the class of weakly simple polygons would change if we demanded a weakly simple polygon to be closed. Would it change the definition if we asked for a weakly simple polygon to be open?*

[2] In general, a topological space is connected if it cannot be described as a disjoint union of open subsets. The property defined here is called path-connected. But as for $\mathbb{R}^d$ both notions are equivalent, we stick to the more intuitive and geometric one.

Exercise 2.11 *Show that in Definition 2.8 the condition that the closure is simply-connected is necessary in the following sense: If dropped then there exist two "weakly simple" polygons that have the same boundary but different interior.*

2.2 Polygon Triangulation

From a topological point of view, a simple polygon is nothing but a disk and so it is a very elementary object. But geometrically a simple polygon can be—as if mocking the label we attached to it—a pretty complicated shape, see Figure 2.3 for an example. While there is an easy and compact one-dimensional representation in terms of the boundary, as a sequence of vertices/points, it is often desirable to work with a more structured representation of the whole two-dimensional shape.

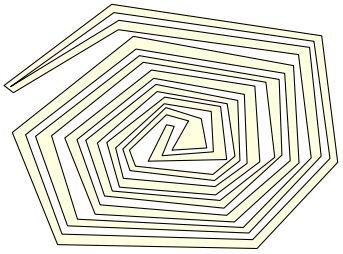


Figure 2.3: *A simple (?) polygon.*

For instance, it is not straightforward to compute the area of a general simple polygon. In order to do so, one usually describes the polygon in terms of simpler geometric objects, for which computing the area is easy. Good candidates for such shapes are triangles, rectangles, and trapezoids. Indeed, it is not hard to show that every simple polygon admits a "nice" partition into triangles, which we call a triangulation.

Definition 2.12 *A* **triangulation** *of a simple polygon* P *is a collection* $\mathcal{T}$ *of triangles, such that*

(1) $P = \bigcup_{T \in \mathcal{T}} T$*;*

(2) the vertices of all triangles in T *are also vertices of* P*; and*

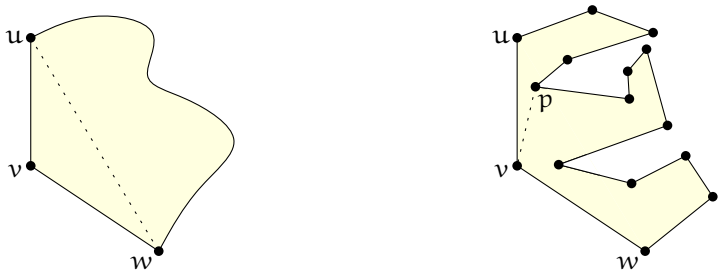
(3) for every distinct pair $T, U \in \mathcal{T}$*, the intersection* $T \cap U$ *is either a common vertex, or a common edge, or empty.*

If we are given a triangulation of a simple polygon P it is easy to compute the area of P by simply summing up the area of all triangles from $\mathcal{T}$. Triangulations are an incredibly useful tool in planar geometry, and one reason for their importance is that every simple polygon admits one.

Theorem 2.13 *Every simple polygon has a triangulation.*

Proof. Let P be a simple polygon on n vertices. We prove the statement by induction on n. For $n = 3$ we face a triangle P that is a triangulation by itself. For $n > 3$ consider the lexicographically smallest vertex v of P, that is, among all vertices of P with a smallest x-coordinate the one with smallest y-coordinate. Denote the neighbors of v (next vertices) along ∂P by u and w. Consider the line segment $\overline{uw}$. We distinguish two cases.

Case 1: except for its endpoints u and w, the segment $\overline{uw}$ lies completely in P°. Then $\overline{uw}$ splits P into two smaller polygons, the triangle uvw and a simple polygon P' on $n-1$ vertices (Figure 2.4a). By the inductive hypothesis, P' has a triangulation that together with T yields a triangulation of P.



(a) Case 1. **(b)** Case 2.

Figure 2.4: *Cases in the proof of Theorem 2.13.*

Case 2: the relative interior of $\overline{uw}$ does not lie completely in P° (Figure 2.4b). By choice of v, the polygon P is contained in the closed halfplane to the right of the vertical line through v. Therefore, as the segments $\overline{uv}$ and $\overline{vw}$ are part of a simple closed curve defining ∂P, every point sufficiently close to v and between the rays vu and vw must be in P°.

On the other hand, since $\overline{uw} \not\subset P^\circ$, there is some point from ∂P in the interior of the triangle $T = uvw$ (by the choice of v the points u, v, w are not collinear and so T is a triangle, indeed) or on the line segment $\overline{uw}$. In particular, as ∂P is composed of line segments, there is a vertex of P in T° or on $\overline{uw}$ (otherwise, a line segment would have to intersect the line segment $\overline{uw}$ twice, which is impossible). Let p denote a leftmost such vertex. Then the open line segment $\overline{vp}$ is contained in T° and, thus, it splits P into two polygons P_1 and P_2 on less than n vertices each (in one of them, u does not appear as a vertex, whereas w does not appear as a vertex in the other). By the inductive hypothesis, both P_1 and P_2 have triangulations and their union yields a triangulation of P. □

The configuration from Case 1 above is called an *ear*: Three consecutive vertices u, v, w of a simple polygon P such that the relative interior of $\overline{uw}$ lies in P°. In fact, we could have skipped the analysis for Case 2 by referring to the following theorem.

Theorem 2.14 (Meisters [9, 10]) *Every simple polygon that is not a triangle has two non-overlapping ears, that is, two ears A and B such that $A^\circ \cap B^\circ = \emptyset$.*

But knowing Theorem 2.13 we can obtain Theorem 2.14 as a direct consequence of the following

Theorem 2.15 *Every triangulation of a simple polygon on $n \geqslant 4$ vertices contains at least two (triangles that are) ears.*

Exercise 2.16 *Prove Theorem 2.15.*

Exercise 2.17 *Let P be a simple polygon with vertices $v_1, v_2, \ldots, v_n$ (in counterclockwise order), where v_i has coordinates (x_i, y_i). Show that the area of P is*

$$\frac{1}{2}\sum_{i=1}^{n} x_{i+1}y_i - x_i y_{i+1},$$

where $(x_{n+1}, y_{n+1}) = (x_1, y_1)$.

The number of edges and triangles in a triangulation of a simple polygon are completely determined by the number of vertices, as the following simple lemma shows.

Lemma 2.18 *Every triangulation of a simple polygon on $n \geqslant 3$ vertices consists of $n-2$ triangles and $2n-3$ edges.*

Proof. Proof by induction on n. The statement is true for $n = 3$. For $n > 3$ consider a simple polygon P on n vertices and an arbitrary triangulation T of P. Any edge uv in T that is not an edge of P (and there must be such an edge because P is not a triangle) partitions P into two polygons P_1 and P_2 with n_1 and n_2 vertices, respectively. Since $n_1, n_2 < n$ we conclude by the inductive hypothesis that T partitions P_1 into $n_1 - 2$ triangles and P_2 into $n_2 - 2$ triangles, using $2n_1 - 3$ and $2n_2 - 3$ edges, respectively.

All vertices of P appear in exactly one of P_1 or P_2, except for u and v, which appear in both. Therefore $n_1+n_2 = n+2$ and so the number of triangles in T is $(n_1-2)+(n_2-2) = (n_1+n_2)-4 = n+2-4 = n-2$. Similarly, all edges of T appear in exactly one of P_1 or P_2, except for the edge uv, which appears in both. Therefore the number of edges in T is $(2n_1-3)+(2n_2-3)-1 = 2(n_1+n_2)-7 = 2(n+2)-7 = 2n-3$. □

The universal presence of triangulations is something particular about the plane: The natural generalization of Theorem 2.13 to dimension three and higher does not hold. What is this generalization, anyway?

Tetrahedralizations in $\mathbb{R}^3$. A simple polygon is a planar object that is a topological disk that is locally bounded by patches of lines. The corresponding term in $\mathbb{R}^3$ is a *polyhedron*, and although we will not formally define it here yet, a literal translation of the previous sentence yields an object that topologically is a ball and is locally bounded by patches of planes. A triangle in $\mathbb{R}^2$ corresponds to a tetrahedron in $\mathbb{R}^3$ and a *tetrahedralization* is a nice partition into tetrahedra, where "nice" means that the union of the tetrahedra covers the object, the vertices of the tetrahedra are vertices of the polyhedron, and any

two distinct tetrahedra intersect in either a common triangular face, or a common edge, or a common vertex, or not at all.[3]

Unfortunately, there are polyhedra in $\mathbb{R}^3$ that do not admit a tetrahedralization. The following construction is due to Schönhardt [12]. It is based on a triangular prism, that is, two congruent triangles placed in parallel planes where the corresponding sides of both triangles are connected by a rectangle (Figure 2.5a). Then one triangle is twisted/rotated slightly within its plane. As a consequence, the rectangular faces are not plane anymore, but they obtain an inward dent along their diagonal in direction of the rotation (Figure 2.5b). The other (former) diagonals of the rectangular faces—labeled ab', bc', and

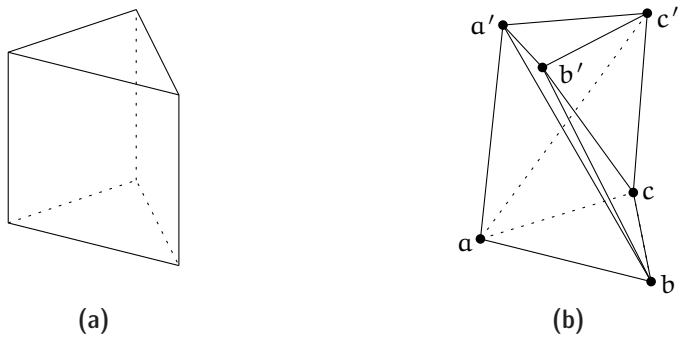


Figure 2.5: *The Schönhardt polyhedron cannot be subdivided into tetrahedra without adding new vertices.*

ca' in Figure 2.5b—are now epigonals, that is, they lie in the exterior of the polyhedron. Since these epigonals are the only edges between vertices that are not part of the polyhedron, there is no way to add edges to form a tetrahedron for a subdivision. Clearly the polyhedron is not a tetrahedron by itself, and so we conclude that it does not admit a subdivision into tetrahedra without adding new vertices. If adding new vertices—so-called Steiner vertices—is allowed, then there is no problem to construct a tetrahedralization, and this holds true in general.

Algorithms. Knowing that a triangulation exists is nice, but it is much better to know that it can also be constructed efficiently.

Exercise 2.19 *Convert Theorem 2.13 into an $O(n^2)$ time algorithm to construct a triangulation for a given simple polygon on n vertices.*

The runtime achieved by the straightforward application of Theorem 2.13 is not optimal. We will revisit this question at several times during this course and discuss improved algorithms for the problem of triangulating a simple polygon.

[3]These "nice" subdivisions can be defined in an abstract combinatorial setting, where they are called *simplicial complices*.

The best (in terms of worst-case runtime) algorithm known due to Chazelle [4] computes a triangulation in linear time. But this algorithm is very complicated and we will not discuss it here. There is also a somewhat simpler randomized algorithm to compute a triangulation in expected linear time [2], which we will not discuss in detail, either. Instead you will later see a much simpler algorithm with a pretty-close-to linear runtime bound. The question of whether there exists a simple (which is not really a well-defined term, of course, except that Chazelle's Algorithm does not qualify) deterministic linear time algorithm to triangulate a simple polygon remains open [7].

Polygons with holes. It is interesting to note that the complexity of the problem changes to $\Theta(n \log n)$, if the polygon may contain holes [3]. This means that there is an algorithm to construct a triangulation for a given simple polygon with holes on a total of n vertices (counting both the vertices on the outer boundary and those of holes) in $O(n \log n)$ time. But there is also a lower bound of $\Omega(n \log n)$ operations that holds in all models of computation in which there exists the corresponding lower bound for comparison-based sorting. This difference in complexity is a very common pattern: There are many problems that are (sometimes much) harder for simple polygons with holes than for simple polygons. So maybe the term "simple" has some justification, after all...

Genaral triangle covers. What if we drop the "niceness" conditions required for triangulations and just want to describe a given simple polygon as a union of triangles? It turns out this is a rather drastic change and, for instance, it is unlikely that we can efficiently find an optimal/minimal description of this type: Christ has shown [5] that it is NP-hard to decide whether for a simple polygon P on n vertices and a positive integer k, there exists a set of at most k triangles whose union is P. In fact, the problem is not even known to be in NP, because it is not clear whether the coordinates of solutions can always be encoded compactly.

2.3 The Art Gallery Problem

In 1973 Victor Klee posed the following question: "How many guards are necessary, and how many are sufficient to patrol the paintings and works of art in an art gallery with n walls?" From a geometric point of view, we may think of an "art gallery with n walls" as a simple polygon bounded by n edges, that is, a simple polygon P with n vertices. And a guard can be modeled as a point where we imagine the guard to stand and observe everything that is in sight. In sight, finally, refers to the walls of the gallery (edges of the polygon) that are opaque and, thus, prevent a guard to see what is behind. In other words, a guard (point) g can watch over every point $p \in P$, for which the line segment $\overline{gp}$ lies completely in P°, see Figure 2.6.

It is not hard to see that $\lfloor n/3 \rfloor$ guards are necessary in general.

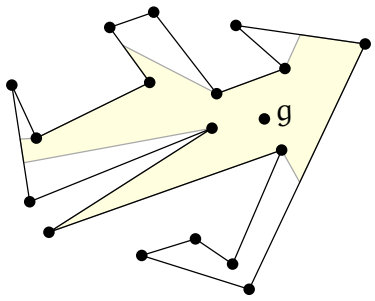


Figure 2.6: *The region that a guard* g *can observe.*

Exercise 2.20 *Describe a family* $(P_n)_{n\geqslant 3}$ *of simple polygons such that* P_n *has* n *vertices and at least* $\lfloor n/3 \rfloor$ *guards are needed to guard it.*

What is more surprising: $\lfloor n/3 \rfloor$ guards are always sufficient as well. Chvátal [6] was the first to prove that, but then Fisk [8] gave a much simpler proof using—you may have guessed it—triangulations. Fisk's proof was considered so beautiful that it was included into "Proofs from THE BOOK" [1], a collection inspired by Paul Erdős' belief in "a place where God keeps aesthetically perfect proofs". The proof is based on the following lemma.

Lemma 2.21 *Every triangulation of a simple polygon is 3-colorable. That is, each vertex can be assigned one of three colors in such a way that adjacent vertices receive different colors.*

Proof. Induction on n. For $n = 3$ the statement is obvious. For $n > 3$, by Theorem 2.15 the triangulation contains an ear uvw. Cutting off the ear creates a triangulation of a polygon on $n - 1$ vertices, which by the inductive hypothesis admits a 3-coloring. Now whichever two colors the vertices u and w receive in this coloring, there remains a third color to be used for v. □

Theorem 2.22 (Fisk [8]) *Every simple polygon on* n *vertices can be guarded using at most* $\lfloor n/3 \rfloor$ *guards.*

Proof. Consider a triangulation of the polygon and a 3-coloring of the vertices as ensured by Lemma 2.21. Take the smallest color class, which clearly consists of at most $\lfloor n/3 \rfloor$ vertices, and put a guard at each vertex. As every point of the polygon is contained in at least one triangle and every triangle has exactly one vertex in the guarding set, the whole polygon is guarded. □

Questions

1. *What is a simple polygon/a simple polygon with holes* Explain the definitions and provide some examples of members and non-members of the respective classes.

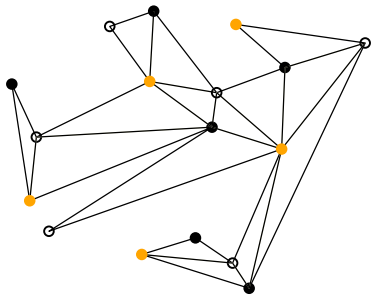


Figure 2.7: *A triangulation of a simple polygon on* 17 *vertices and a 3-coloring of it. The vertices shown solid orange form the smallest color class and guard the polygon using* $\lfloor 17/3 \rfloor = 5$ *guards.*

For a given polygon you should be able to tell which of these classes it belongs to or does not belong to and argue why this is the case.

2. *What is a closed/open/bounded/connected/simply-connected set in* $\mathbb{R}^d$*? What is the interior/closure of a point set?* Explain the definitions and provide some illustrative examples. For a given set you should be able to argue which of the properties mentioned it possesses.

3. *What is a triangulation of a simple polygon? Does it always exist?* Explain the definition and provide some illustrative examples. Present the proof of Theorem 2.13 in detail.

4. *How about higher dimensional generalizations? Can every polyhedron in* $\mathbb{R}^3$ *be nicely subdivided into tetrahedra?* Explain Schönhardt's construction.

5. *How many points are needed to guard a simple polygon?* Present the proofs of Theorem 2.15, Lemma 2.21, and Theorem 2.22 in detail.

References

[1] Martin Aigner and Günter M. Ziegler, *Proofs from THE BOOK*. Springer-Verlag, Berlin, 3rd edn., 2003.

[2] Nancy M. Amato, Michael T. Goodrich, and Edgar A. Ramos, A randomized algorithm for triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, **26**, 2, (2001), 245–265, URL `http://dx.doi.org/10.1007/s00454-001-0027-x`.

[3] Takao Asano, Tetsuo Asano, and Ron Y. Pinter, Polygon triangulation: efficiency and minimality. *J. Algorithms*, **7**, 2, (1986), 221–231, URL `http://dx.doi.org/10.1016/0196-6774(86)90005-2`.

[4] Bernard Chazelle, Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, **6**, 5, (1991), 485–524, URL `http://dx.doi.org/10.1007/BF02574703`.

[5] Tobias Christ, Beyond triangulation: covering polygons with triangles. In *Proc. 12th Algorithms and Data Struct. Sympos.*, vol. 6844 of *Lecture Notes Comput. Sci.*, pp. 231–242, Springer-Verlag, 2011, URL http://dx.doi.org/10.1007/978-3-642-22300-6_20.

[6] Václav Chvátal, A combinatorial theorem in plane geometry. *J. Combin. Theory Ser. B*, **18**, 1, (1975), 39–41, URL http://dx.doi.org/10.1016/0095-8956(75)90061-1.

[7] Erik D. Demaine, Joseph S. B. Mitchell, and Joseph O'Rourke, The Open Problems Project, Problem #10. http://cs.smith.edu/~orourke/TOPP/P10.html.

[8] Steve Fisk, A short proof of Chvátal's watchman theorem. *J. Combin. Theory Ser. B*, **24**, 3, (1978), 374, URL http://dx.doi.org/10.1016/0095-8956(78)90059-X.

[9] Gary H. Meisters, Polygons have ears. *Amer. Math. Monthly*, **82**, 6, (1975), 648–651, URL http://www.jstor.org/stable/2319703.

[10] Gary H. Meisters, Principal vertices, exposed points, and ears. *Amer. Math. Monthly*, **87**, 4, (1980), 284–285, URL http://www.jstor.org/stable/2321563.

[11] Bojan Mohar and Carsten Thomassen, *Graphs on surfaces*. Johns Hopkins University Press, Baltimore, 2001.

[12] Erich Schönhardt, Über die Zerlegung von Dreieckspolyedern in Tetraeder. *Math. Ann.*, **98**, (1928), 309–312, URL http://dx.doi.org/10.1007/BF01451597.

Chapter 3

Convex Hull

There exists an incredible variety of point sets and polygons. Among them, some have certain properties that make them "nicer" than others in some respect. For instance, look at the two polygons shown below.

(a) A convex polygon. **(b)** A non-convex polygon.

Figure 3.1: *Examples of polygons: Which do you like better?*

As it is hard to argue about aesthetics, let us take a more algorithmic stance. When designing algorithms, the polygon shown on the left appears much easier to deal with than the visually and geometrically more complex polygon shown on the right. One particular property that makes the left polygon nice is that one can walk between any two vertices along a straight line without ever leaving the polygon. In fact, this statement holds true not only for vertices but for any two points within the polygon. A polygon or, more generally, a set with this property is called *convex*.

Definition 3.1 *A set* $P \subseteq \mathbb{R}^d$ *is* **convex** *if and only if* $\overline{pq} \subseteq P$*, for any* $p, q \in P$*.*

An alternative, equivalent way to phrase convexity would be to demand that for every line $\ell \subset \mathbb{R}^d$ the intersection $\ell \cap P$ be connected. The polygon shown in Figure 3.1b is not convex because there are some pairs of points for which the connecting line segment is not completely contained within the polygon.

Indeed there are many problems that are comparatively easy to solve for convex sets but very hard in general. We will encounter some particular instances of this phenomenon

later in the course. However, not all polygons are convex and a discrete set of points is never convex, unless it consists of at most one point only. In such a case it is useful to make a given set P convex, that is, approximate P with or, rather, encompass P within a convex set $H \supseteq P$. Ideally, H differs from P as little as possible, that is, we want H to be a smallest convex set enclosing P.

At this point let us step back for a second and ask ourselves whether this wish makes sense at all: Does such a set H (always) exist? Fortunately, we are on the safe side because the whole space $\mathbb{R}^d$ is certainly convex. It is less obvious, but we will see below that H is actually unique. Therefore it is legitimate to refer to H as **the** smallest convex set enclosing P or—shortly—the *convex hull* of P.

3.1 Convexity

Consider $P \subset \mathbb{R}^d$. The following terminology should be familiar from linear algebra courses.

Linear hull.

$$\text{lin}(P) := \left\{ q \;\middle|\; q = \sum \lambda_i p_i \wedge \forall i : p_i \in P, \lambda_i \in \mathbb{R} \right\},$$

the set of all *linear combinations* of P (smallest linear subspace containing P). For instance, if $P = \{p\} \subset \mathbb{R}^2 \setminus \{0\}$ then $\text{lin}(P)$ is the line through p and the origin.

Affine hull.

$$\text{aff}(P) := \left\{ q \;\middle|\; q = \sum \lambda_i p_i \wedge \sum \lambda_i = 1 \wedge \forall i : p_i \in P, \lambda_i \in \mathbb{R} \right\},$$

the set of all *affine combinations* of P (smallest affine subspace containing P). For instance, if $P = \{p, q\} \subset \mathbb{R}^2$ and $p \neq q$ then $\text{aff}(P)$ is the line through p and q.

Convex hull.

Proposition 3.2 *A set $P \subseteq \mathbb{R}^d$ is convex if and only if $\sum_{i=1}^n \lambda_i p_i \in P$, for all $n \in \mathbb{N}$, $p_1, \ldots, p_n \in P$, and $\lambda_1, \ldots, \lambda_n \geqslant 0$ with $\sum_{i=1}^n \lambda_i = 1$.*

Proof. "$\Leftarrow$": obvious with $n = 2$.

"$\Rightarrow$": Induction on n. For $n = 1$ the statement is trivial. For $n \geqslant 2$, let $p_i \in P$ and $\lambda_i \geqslant 0$, for $1 \leqslant i \leqslant n$, and assume $\sum_{i=1}^n \lambda_i = 1$. We may suppose that $\lambda_i > 0$, for all i. (Simply omit those points whose coefficient is zero.) We need to show that $\sum_{i=1}^n \lambda_i p_i \in P$.

Define $\lambda = \sum_{i=1}^{n-1} \lambda_i$ and for $1 \leqslant i \leqslant n-1$ set $\mu_i = \lambda_i / \lambda$. Observe that $\mu_i \geqslant 0$ and $\sum_{i=1}^{n-1} \mu_i = 1$. By the inductive hypothesis, $q := \sum_{i=1}^{n-1} \mu_i p_i \in P$, and thus by convexity of P also $\lambda q + (1-\lambda) p_n \in P$. We conclude by noting that $\lambda q + (1-\lambda) p_n = \lambda \sum_{i=1}^{n-1} \mu_i p_i + \lambda_n p_n = \sum_{i=1}^{n} \lambda_i p_i$. □

Observation 3.3 *For any family* $(P_i)_{i\in I}$ *of convex sets the intersection* $\bigcap_{i\in I} P_i$ *is convex.*

Definition 3.4 *The* **convex hull** $\mathrm{conv}(P)$ *of a set* $P \subset \mathbb{R}^d$ *is the intersection of all convex supersets of* P*.*

By Observation 3.3, the convex hull is convex, indeed. The following proposition provides an algebraic description of the convex hull, similar as given above for the linear and affine hull.

Proposition 3.5 *For any* $P \subseteq \mathbb{R}^d$ *we have*

$$\mathrm{conv}(P) = \left\{ \sum_{i=1}^{n} \lambda_i p_i \;\middle|\; n \in \mathbb{N} \wedge \sum_{i=1}^{n} \lambda_i = 1 \wedge \forall i \in \{1,\dots,n\} : \lambda_i \geqslant 0 \wedge p_i \in P \right\}.$$

The elements of the set on the right hand side are referred to as *convex combinations* of P.

Proof. "$\supseteq$": Consider a convex set $C \supseteq P$. By Proposition 3.2 (only-if direction) the right hand side is contained in C. As C was arbitrary, the claim follows.

"$\subseteq$": Denote the set on the right hand side by R. We show that R forms a convex set. Let $p = \sum_{i=1}^{n} \lambda_i p_i$ and $q = \sum_{i=1}^{n} \mu_i p_i$ be two convex combinations. (We may suppose that both p and q are expressed over the same p_i by possibly adding some terms with a coefficient of zero.)

Then for $\lambda \in [0,1]$ we have $\lambda p + (1-\lambda)q = \sum_{i=1}^{n} (\lambda\lambda_i + (1-\lambda)\mu_i)p_i \in R$, as $\underbrace{\lambda\lambda_i}_{\geqslant 0} + \underbrace{(1-\lambda)}_{\geqslant 0}\underbrace{\mu_i}_{\geqslant 0} \geqslant 0$, for all $1 \leqslant i \leqslant n$, and $\sum_{i=1}^{n} (\lambda\lambda_i + (1-\lambda)\mu_i) = \lambda + (1-\lambda) = 1$. □

Definition 3.6 *The convex hull of a finite point set* $P \subset \mathbb{R}^d$ *forms a* **convex polytope**. *Each* $p \in P$ *for which* $p \notin \mathrm{conv}(P \setminus \{p\})$ *is called a* **vertex** *of* $\mathrm{conv}(P)$*. A vertex of* $\mathrm{conv}(P)$ *is also called an* **extremal point** *of* P*. A convex polytope in* $\mathbb{R}^2$ *is called a* **convex polygon**. *A convex polytope in* $\mathbb{R}^3$ *is called a* **convex polyhedron**.

Essentially, the following proposition shows that the term vertex above is well defined.

Proposition 3.7 *A convex polytope in* $\mathbb{R}^d$ *is the convex hull of its vertices.*

Proof. Let $P = \{p_1,\dots,p_n\}$, $n \in \mathbb{N}$, such that without loss of generality $p_1,\dots,p_k$ are the vertices of $\mathcal{P} := \mathrm{conv}(P)$. We prove by induction on n that $\mathrm{conv}(p_1,\dots,p_n) \subseteq \mathrm{conv}(p_1,\dots,p_k)$. For $n = k$ the statement is trivial.

For $n > k$, p_n is not a vertex of $\mathcal{P}$ and hence p_n can be expressed as a convex combination $p_n = \sum_{i=1}^{n-1} \lambda_i p_i$. Thus for any $x \in \mathcal{P}$ we can write $x = \sum_{i=1}^{n} \mu_i p_i = \sum_{i=1}^{n-1} \mu_i p_i + \mu_n \sum_{i=1}^{n-1} \lambda_i p_i = \sum_{i=1}^{n-1} (\mu_i + \mu_n\lambda_i) p_i$. As $\sum_{i=1}^{n-1} (\mu_i + \mu_n\lambda_i) = 1$, we conclude inductively that $x \in \mathrm{conv}(p_1,\dots,p_{n-1}) \subseteq \mathrm{conv}(p_1,\dots,p_k)$. □

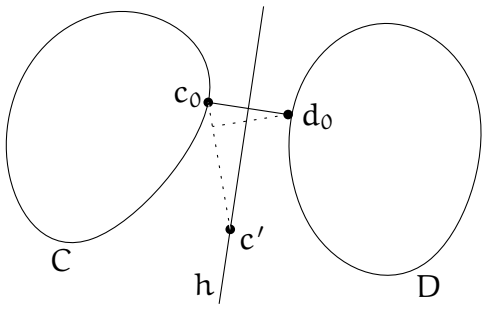
Theorem 3.8 (Carathéodory [3]) *For any* $P \subset \mathbb{R}^d$ *and* $q \in \text{conv}(P)$ *there exist* $k \leqslant d+1$ *points* $p_1, \ldots, p_k \in P$ *such that* $q \in \text{conv}(p_1, \ldots, p_k)$.

Exercise 3.9 *Prove Theorem 3.8.*

Theorem 3.10 (Separation Theorem) *Any two compact convex sets* $C, D \subset \mathbb{R}^d$ *with* $C \cap D = \emptyset$ *can be separated strictly by a hyperplane, that is, there exists a hyperplane* h *such that* C *and* D *lie in the opposite open halfspaces bounded by* h.

Proof. Consider the distance function $d : C \times D \to \mathbb{R}$ with $(c, d) \mapsto \|c - d\|$. Since $C \times D$ is compact and d is continuous and strictly bounded from below by 0, d attains its minimum at some point $(c_0, d_0) \in C \times D$ with $d(c_0, d_0) > 0$. Let h be the hyperplane perpendicular to the line segment $\overline{c_0 d_0}$ and passing through the midpoint of c_0 and d_0.

If there was a point, say, c' in $C \cap h$, then by convexity of C the whole line segment $\overline{c_o c'}$ lies in C and some point along this segment is closer to d_0 than is c_0, in contradiction to the choice of c_0. The figure shown to the right depicts the situation in $\mathbb{R}^2$. If, say, C has points on both sides of h, then by convexity of C it has also a point on h, but we just saw that there is no such point. Therefore, C and D must lie in different open halfspaces bounded by h. □



Actually, the statement above holds for arbitrary (not necessarily compact) convex sets, but the separation is not necessarily strict (the hyperplane may have to intersect the sets) and the proof is a bit more involved (cf. [7], but also check the errata on Matoušek's webpage).

Exercise 3.11 *Show that the Separation Theorem does not hold in general, if not both of the sets are convex.*

Exercise 3.12 *Prove or disprove:*

(a) The convex hull of a compact subset of $\mathbb{R}^d$ *is compact.*

(b) The convex hull of a closed subset of $\mathbb{R}^d$ *is closed.*

Altogether we obtain various equivalent definitions for the convex hull, summarized in the following theorem.

Theorem 3.13 *For a compact set* $P \subset \mathbb{R}^d$ *we can characterize* $\text{conv}(P)$ *equivalently as one of*

(a) the smallest (w. r. t. set inclusion) convex subset of $\mathbb{R}^d$ *that contains* P*;*

(b) the set of all convex combinations of points from P*;*

(c) the set of all convex combinations formed by $d+1$ *or fewer points from* P*;*

(d) the intersection of all convex supersets of P*;*

(e) the intersection of all closed halfspaces containing P*.*

Exercise 3.14 *Prove Theorem 3.13.*

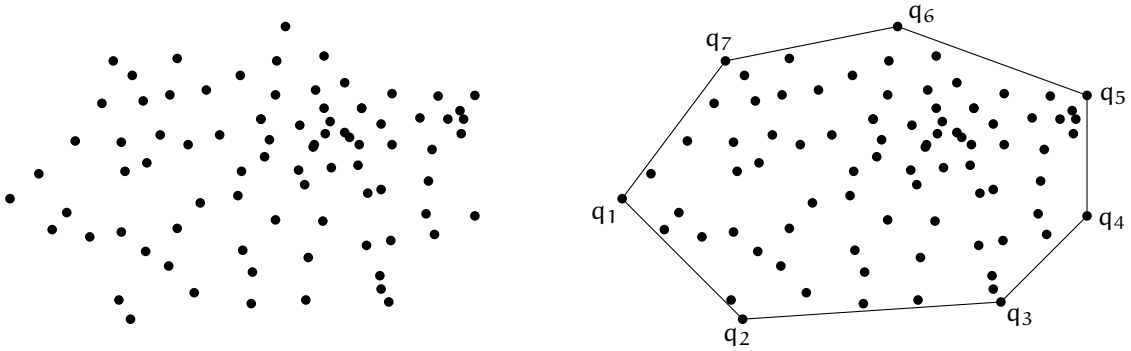
3.2 Planar Convex Hull

Although we know by now what is the convex hull of point set, it is not yet clear how to construct it algorithmically. As a first step, we have to find a suitable representation for convex hulls. In this section we focus on the problem in $\mathbb{R}^2$, where the convex hull of a finite point set forms a convex polygon. A convex polygon is easy to represent, for instance, as a sequence of its vertices in counterclockwise orientation. In higher dimensions finding a suitable representation for convex polytopes is a much more delicate task.

Problem 3.15 (Convex hull)

Input: $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, $n \in \mathbb{N}$.

Output: Sequence $(q_1, \ldots, q_h)$, $1 \leqslant h \leqslant n$, of the vertices of $\mathrm{conv}(P)$ (ordered counterclockwise).



(a) Input. **(b)** Output.

Figure 3.2: *Convex Hull of a set of points in* $\mathbb{R}^2$*.*

Another possible algorithmic formulation of the problem is to ignore the structure of the convex hull and just consider it as a point set.

Problem 3.16 (Extremal points)

Input: $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$, $n \in \mathbb{N}$.

Output: Set $Q \subseteq P$ of the vertices of $\mathrm{conv}(P)$.

Degeneracies. A couple of further clarifications regarding the above problem definitions are in order.

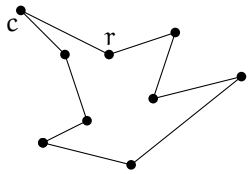
First of all, for efficiency reasons an input is usually specified as a sequence of points. Do we insist that this sequence forms a set or are duplications of points allowed?

What if three points are collinear? Are all of them considered extremal? According to our definition from above, they are not and that is what we will stick to. But note that there may be cases where one wants to include all such points, nevertheless.

By the Separation Theorem, every extremal point p can be separated from the convex hull of the remaining points by a halfplane. If we take such a halfplane and shift its defining line such that it passes through p, then all points from P other than p should lie in the resulting open halfplane. In $\mathbb{R}^2$ it turns out convenient to work with the following "directed" reformulation.

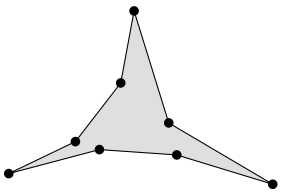
Proposition 3.17 *A point $p \in P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$ is* **extremal** *for P $\iff$ there is a directed line g through p such that $P \setminus \{p\}$ is to the left of g.*

The *interior angle* at a vertex v of a polygon P is the angle between the two edges of P incident to v whose corresponding angular domain lies in P°. If this angle is smaller than π, the vertex is called *convex*; if the angle is larger than π, the vertex is called *reflex*. For instance, the vertex c in the polygon depicted to the right is a convex vertex, whereas the vertex labeled r is a reflex vertex.



Exercise 3.18

A simple polygon $S \subset \mathbb{R}^2$ is star-shaped *if and only if there exists a point $c \in S$, such that for every point $p \in S$ the line segment $\overline{cp}$ is contained in S. A simple polygon with exactly three convex vertices is called a pseudotriangle (see the example shown on the right).*



In the following we consider subsets of $\mathbb{R}^2$. Prove or disprove:

a) Every convex vertex of a simple polygon lies on its convex hull.

b) Every star-shaped set is convex.

c) Every convex set is star-shaped.

d) The intersection of two convex sets is convex.

e) The union of two convex sets is convex.

f) The intersection of two star-shaped sets is star-shaped.

g) The intersection of a convex set with a star-shaped set is star-shaped.

h) Every triangle is a pseudotriangle.

i) Every pseudotriangle is star-shaped.

3.3 Trivial algorithms

One can compute the extremal points using Carathéodory's Theorem as follows: Test for every point $p \in P$ whether there are $q, r, s \in P \setminus \{p\}$ such that p is inside the triangle with vertices q, r, and s. Runtime $O(n^4)$.

Another option, inspired by the Separation Theorem: test for every pair $(p, q) \in P^2$ whether all points from $P \setminus \{p, q\}$ are to the left of the directed line through p and q (or on the line segment $\overline{pq}$). Runtime $O(n^3)$.

Exercise 3.19 *Let $P = (p_0, \ldots, p_{n-1})$ be a sequence of n points in $\mathbb{R}^2$. Someone claims that you can check by means of the following algorithm whether or not P describes the boundary of a convex polygon in counterclockwise order:*

```
bool is_convex(p_0, ..., p_{n-1}) {
    for i = 0, ..., n-1:
        if (p_i, p_{(i+1) mod n}, p_{(i+2) mod n}) form a rightturn:
            return false;
    return true;
}
```

Disprove the claim and describe a correct algorithm to solve the problem.

Exercise 3.20 *Let $P \subset \mathbb{R}^2$ be a convex polygon, given as an array p[0]...p[n-1] of its n vertices in counterclockwise order.*

a) Describe an $O(\log(n))$ time algorithm to determine whether a point q lies inside, outside or on the boundary of P.

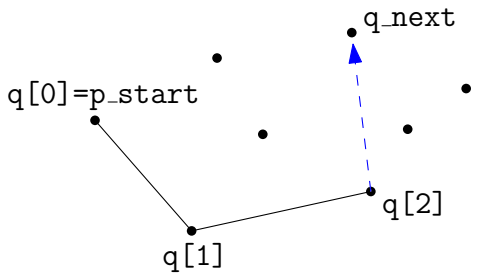
b) Describe an $O(\log(n))$ time algorithm to find a (right) tangent to P from a query point q located outside P. That is, find a vertex $p[i]$, such that P is contained in the closed halfplane to the left of the oriented line $qp[i]$.

3.4 Jarvis' Wrap

We are now ready to describe a first simple algorithm to construct the convex hull. It works as follows:

> Find a point p_1 that is a vertex of $\text{conv}(P)$ (e.g., the one with smallest x-coordinate). "Wrap" P starting from p_1, i.e., always find the next vertex of $\text{conv}(P)$ as the one that is rightmost with respect to the direction given by the previous two vertices.

Besides comparing x-coordinates, the only geometric primitive needed is an *orientation* test: Denote by $\texttt{rightturn}(p, q, r)$, for three points $p, q, r \in \mathbb{R}^2$, the predicate that is true if and only if r is (strictly) to the right of the oriented line pq.



Code for Jarvis' Wrap.

`p[0..N)` contains a sequence of N points.
`p_start` point with smallest x-coordinate.
`q_next` some *other* point in `p[0..N)`.

```
int h = 0;
Point_2 q_now = p_start;
do {
  q[h] = q_now;
  h = h + 1;

  for (int i = 0; i < N; i = i + 1)
    if (rightturn_2(q_now, q_next, p[i]))
      q_next = p[i];

  q_now = q_next;
  q_next = p_start;
} while (q_now != p_start);
```

`q[0,h)` describes a convex polygon bounding the convex hull of `p[0..N)`.

Analysis. For every output point the above algorithm spends n rightturn tests, which is $\Rightarrow O(nh)$ in total.

Theorem 3.21 [6] *Jarvis' Wrap computes the convex hull of n points in $\mathbb{R}^2$ using $O(nh)$ rightturn tests, where h is the number of hull vertices.*

In the worst case we have $h = n$, that is, $O(n^2)$ rightturn tests. Jarvis' Wrap has a remarkable property that is called *output sensitivity*: the runtime depends not only on the size of the input but also on the size of the output. For a huge point set it constructs

the convex hull in optimal linear time, if the convex hull consists of a constant number of vertices only. Unfortunately the worst case performance of Jarvis' Wrap is suboptimal, as we will see soon.

Degeneracies. The algorithm may have to cope with various degeneracies.

- Several points have smallest x-coordinate $\Rightarrow$ lexicographic order:

$$(p_x, p_y) < (q_x, q_y) \iff p_x < q_x \vee p_x = q_x \wedge p_y < q_y \ .$$

- Three or more points collinear $\Rightarrow$ choose the point that is farthest among those that are rightmost.

Predicates. Besides the lexicographic comparison mentioned above, the Jarvis' Wrap (and most other 2D convex hull algorithms for that matter) need one more geometric predicate: the rightturn or—more generally—orientation test. The computation amounts to evaluating a polynomial of degree two, see the exercise below. We therefore say that the orientation test has *algebraic degree* two. In contrast, the lexicographic comparison has degree one only. The algebraic degree not only has a direct impact on the efficiency of a geometric algorithm (lower degree $\leftrightarrow$ less multiplications), but also an indirect one because high degree predicates may create large intermediate results, which may lead to overflows and are much more costly to compute with exactly.

Exercise 3.22 *Prove that for three points* $(p_x, p_y), (q_x, q_y), (r_x, r_y) \in \mathbb{R}^2$*, the sign of the determinant*

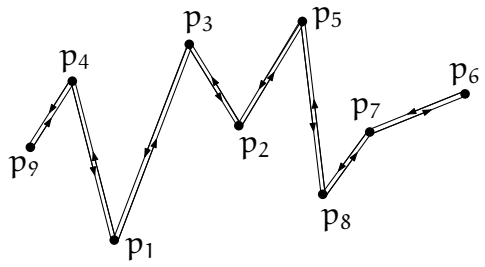
$$\begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix}$$

determines if r *lies to the right, to the left or on the directed line through* p *and* q*.*

3.5 Graham Scan (Successive Local Repair)

There exist many algorithms that exhibit a better worst-case runtime than Jarvis' Wrap. Here we discuss only one of them: a particularly elegant and easy-to-implement variant of the so-called *Graham Scan* [5]. This algorithm is referred to as *Successive Local Repair* because it starts with some polygon enclosing all points and then step-by-step repairs the deficiencies of this polygon, by removing non-convex vertices. It goes as follows:

Sort points lexicographically and remove duplicates: $(p_1, \ldots, p_n)$.



$p_9\, p_4\, p_1\, p_3\, p_2\, p_5\, p_8\, p_7\, p_6\, p_7\, p_8\, p_5\, p_2\, p_3\, p_1\, p_4\, p_9$

As long as there is a (consecutive) triple (p, q, r) such that r is to the right of or on the directed line $\overrightarrow{pq}$, remove q from the sequence.

Code for Graham Scan.

`p[0..N)` lexicographically sorted sequence of pairwise distinct points, $N \geqslant 2$.

```
q[0] = p[0];
int h = 0;
// Lower convex hull (left to right):
for (int i = 1; i < N; i = i + 1) {
  while (h>0 && !leftturn_2(q[h-1], q[h], p[i]))
    h = h - 1;
  h = h + 1;
  q[h] = p[i];
}

// Upper convex hull (right to left):
for (int i = N-2; i >= 0; i = i - 1) {
  while (!leftturn_2(q[h-1], q[h], p[i]))
    h = h - 1;
  h = h + 1;
  q[h] = p[i];
}
```

`q[0,h)` describes a convex polygon bounding the convex hull of `p[0..N)`.

Analysis.

Theorem 3.23 *The convex hull of a set $P \subset \mathbb{R}^2$ of n points can be computed using $O(n \log n)$ geometric operations.*

Proof.

1. Sorting and removal of duplicate points: $O(n \log n)$.

2. At the beginning we have a sequence of $2n-1$ points; at the end the sequence consists of h points. Observe that for every positive orientation test, one point is discarded from the sequence for good. Therefore, we have exactly $2n-h-1$ such shortcuts/positive orientation tests. In addition there are at most $2n-2$ negative tests (#iterations of the outer `for` loops). Altogether we have at most $4n-h-3$ orientation tests.

In total the algorithm uses $O(n \log n)$ geometric operations. Note that the number of orientation tests is linear only, but $O(n \log n)$ lexicographic comparisons are needed. □

3.6 Lower Bound

It is not hard to see that the runtime of Graham Scan is asymptotically optimal in the worst-case.

Theorem 3.24 $\Omega(n \log n)$ *geometric operations are needed to construct the convex hull of* n *points in* $\mathbb{R}^2$ *(in the algebraic computation tree model).*

Proof. Reduction from sorting (for which it is known that $\Omega(n \log n)$ comparisons are needed in the algebraic computation tree model). Given n real numbers $x_1, \ldots, x_n$, construct a set $P = \{p_i \mid 1 \leqslant i \leqslant n\}$ of n points in $\mathbb{R}^2$ by setting $p_i = (x_i, x_i^2)$. This construction can be regarded as embedding the numbers into $\mathbb{R}^2$ along the x-axis and then projecting the resulting points vertically onto the unit parabola. The order in which the points appear along the lower convex hull of P corresponds to the sorted order of the x_i. Therefore, if we could construct the convex hull in $o(n \log n)$ time, we could also sort in $o(n \log n)$ time. □

Clearly this reduction does not work for the Extremal Points problem. But using a reduction from Element Uniqueness (see Section 1.1) instead, one can show that $\Omega(n \log n)$ is also a lower bound for the number of operations needed to compute the set of extremal points only. This was first shown by Avis [1] for linear computation trees, then by Yao [8] for quadratic computation trees, and finally by Ben-Or [2] for general algebraic computation trees.

3.7 Chan's Algorithm

Given matching upper and lower bounds we may be tempted to consider the algorithmic complexity of the planar convex hull problem settled. However, this is not really the case: Recall that the lower bound is a worst case bound. For instance, the Jarvis' Wrap runs in $O(nh)$ time an thus beats the $\Omega(n \log n)$ bound in case that $h = o(\log n)$. The question remains whether one can achieve both output dependence and optimal worst case performance at the same time. Indeed, Chan [4] presented an algorithm to achieve this runtime by cleverly combining the "best of" Jarvis' Wrap and Graham Scan. Let us

look at this algorithm in detail. The algorithm consists of two steps that are executed one after another.

Divide. *Input:* a set $P \subset \mathbb{R}^2$ of n points and a number $H \in \{1, \ldots, n\}$.

1. Divide P into $k = \lceil n/H \rceil$ sets $P_1, \ldots, P_k$ with $|P_i| \leqslant H$.

2. Construct $\text{conv}(P_i)$ for all i, $1 \leqslant i \leqslant k$.

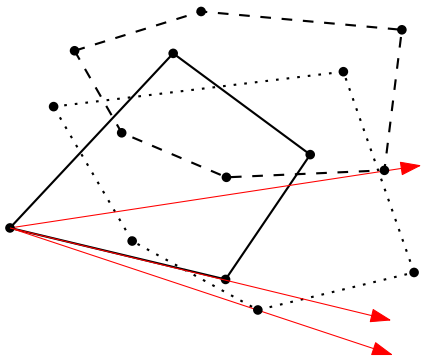
Analysis. Step 1 takes $O(n)$ time. Step 2 can be handled using Graham Scan in $O(H \log H)$ time for any single P_i, that is, $O(n \log H)$ time in total.

Conquer. *Output:* H vertices of $\text{conv}(P)$, or the message that $\text{conv}(P)$ has less than H vertices.

1. Find the lexicographically smallest point in $\text{conv}(P_i)$ for all i, $1 \leqslant i \leqslant k$.

2. Starting from the lexicographically smallest point of P find the first H points of $\text{conv}(P)$ oriented counterclockwise (simultaneous Jarvis' Wrap on the sequences $\text{conv}(P_i)$).

Determine in every wrap step the point q_i of tangency from the current point of $\text{conv}(P)$ to $\text{conv}(P_i)$, for all $1 \leqslant i \leqslant k$. We have seen in Exercise 3.20 how to compute q_i in $O(\log |\text{conv}(P_i)|) = O(\log H)$ time. Among the k candidates $q_1, \ldots, q_k$ we find the next vertex of $\text{conv}(P)$ in $O(k)$ time.



Analysis. Step 1 takes $O(n)$ time. Step 2 consists of at most H wrap steps. Each wrap step needs $O(k \log H + k) = O(k \log H)$ time, which amounts to $O(Hk \log H) = O(n \log H)$ time for Step 2 in total.

Remark. Using a more clever search strategy instead of many tangency searches one can handle the conquer phase in $O(n)$ time, see Exercise 3.25 below. However, this is irrelevant as far as the asymptotic runtime is concerned, given that already the divide step takes $O(n \log H)$ time.

Exercise 3.25 *Consider k convex polygons $P_1, \ldots P_k$, for some constant $k \in \mathbb{N}$, where each polygon is given as a list of its vertices in counterclockwise orientation. Show how to construct the convex hull of $P_1 \cup \ldots \cup P_k$ in $O(n)$ time, where $n = \sum_{i=1}^{k} n_i$ and n_i is the number of vertices of P_i, for $1 \leqslant i \leqslant k$.*

Searching for h. While the runtime bound for $H = h$ is exactly what we were heading for, it looks like in order to actually run the algorithm we would have to know h, which—in general—we do not. Fortunately we can circumvent this problem rather easily, by applying what is called a *doubly exponential search*. It works as follows.

Call the algorithm from above iteratively with parameter $H = \min\{2^{2^t}, n\}$, for $t = 0, \ldots$, until the conquer step finds all extremal points of P (i.e., the wrap returns to its starting point).

Analysis: Let 2^{2^s} be the last parameter for which the algorithm is called. Since the previous call with $H = 2^{2^{s-1}}$ did not find all extremal points, we know that $2^{2^{s-1}} < h$, that is, $2^{s-1} < \log h$, where h is the number of extremal points of P. The total runtime is therefore at most

$$\sum_{i=0}^{s} cn \log 2^{2^i} = cn \sum_{i=0}^{s} 2^i = cn(2^{s+1} - 1) < 4cn \log h = O(n \log h),$$

for some constant $c \in \mathbb{R}$. In summary, we obtain the following theorem.

Theorem 3.26 *The convex hull of a set $P \subset \mathbb{R}^2$ of n points can be computed using $O(n \log h)$ geometric operations, where h is the number of convex hull vertices.*

Questions

6. *How is convexity defined? What is the convex hull of a set in $\mathbb{R}^d$?* Give at least three possible definitions.

7. *What does it mean to compute the convex hull of a set of points in $\mathbb{R}^2$?* Discuss input and expected output and possible degeneracies.

8. *How can the convex hull of a set of n points in $\mathbb{R}^2$ be computed efficiently?* Describe and analyze (incl. proofs) Jarvis' Wrap, Successive Local Repair, and Chan's Algorithm.

9. *Is there a linear time algorithm to compute the convex hull of n points in $\mathbb{R}^2$?* Prove the lower bound and define/explain the model in which it holds.

10. *Which geometric primitive operations are used to compute the convex hull of n points in $\mathbb{R}^2$?* Explain the two predicates and how to compute them.

References

[1] David Avis, Comments on a lower bound for convex hull determination. *Inform. Process. Lett.*, **11**, 3, (1980), 126, URL http://dx.doi.org/10.1016/0020-0190(80)90125-8.

[2] Michael Ben-Or, Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. Theory Comput.*, pp. 80–86, 1983, URL http://dx.doi.org/10.1145/800061.808735.

[3] C. Carathéodory, Über den Variabilitätsbereich der Fourierschen Konstanten von positiven harmonischen Funktionen. *Rendiconto del Circolo Matematico di Palermo*, **32**, (1911), 193–217.

[4] Timothy M. Chan, Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete Comput. Geom.*, **16**, 4, (1996), 361–368, URL http://dx.doi.org/10.1007/BF02712873.

[5] Ronald L. Graham, An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, **1**, 4, (1972), 132–133, URL http://dx.doi.org/10.1016/0020-0190(72)90045-2.

[6] Ray A. Jarvis, On the identification of the convex hull of a finite set of points in the plane. *Inform. Process. Lett.*, **2**, 1, (1973), 18–21, URL http://dx.doi.org/10.1016/0020-0190(73)90020-3.

[7] Jiří Matoušek, *Lectures on discrete geometry*. Springer-Verlag, New York, NY, 2002.

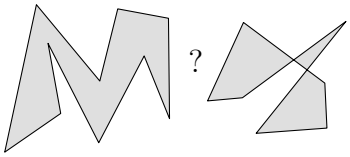
[8] Andrew C. Yao, A lower bound to finding convex hulls. *J. ACM*, **28**, 4, (1981), 780–787, URL http://dx.doi.org/10.1145/322276.322289.

Chapter 4

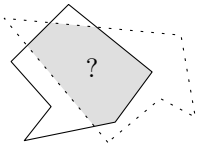
Line Sweep

In this chapter we will discuss a simple and widely applicable paradigm to design geometric algorithms: the so-called *Line-Sweep* (or Plane-Sweep) technique. It can be used to solve a variety of different problems, some examples are listed below. The first part may come as a reminder to many of you, because you should have heard something about line-sweep in one of the basic CS courses already. However, we will soon proceed and encounter a couple of additional twists that were most likely not covered there.

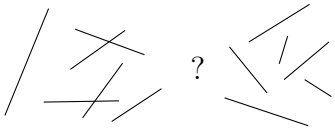
Consider the following geometric problems.



Problem 4.1 Given a sequence $P = (p_1, \ldots, p_n)$ of points in $\mathbb{R}^2$, does P describe the boundary of a simple polygon?

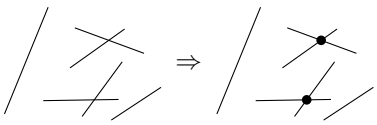


Problem 4.2 (Polygon Intersection) Given two simple polygons P and Q in $\mathbb{R}^2$ as a (counterclockwise) sequence of their vertices, is $P \cap Q = \emptyset$?



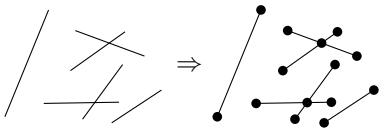
Problem 4.3 (Segment Intersection Test) Given a set of n closed line segments in $\mathbb{R}^2$, do any two of them intersect?

Remark: In principle it is clear what is meant by "two segments intersect". But there are a few special cases that one may have to consider carefully. For instance, does it count if an endpoint lies on another segment? What if two segments share an endpoint? What about overlapping segments and segments of length zero? In general, let us count all these as intersections. However, sometimes we may want to exclude some of these cases. For instance, in a simple polygon test, we do not want to consider the shared endpoint between two consecutive edges of the boundary as an intersection.

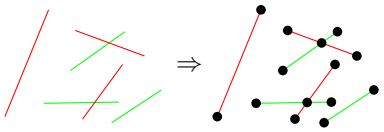


Problem 4.4 (Segment Intersections) Given a set of n closed line segments in $\mathbb{R}^2$, compute all pairs of segments that intersect.

Problem 4.5 (Segment Arrangement) Given a set of n closed line segments in $\mathbb{R}^2$, construct the arrangement induced by S, that is, the subdivision of $\mathbb{R}^2$ induced by S.



Problem 4.6 (Map Overlay) Given two sets S and T of n and m, respectively, pairwise interior disjoint line segments in $\mathbb{R}^2$, construct the arrangement induced by $S \cup T$.



In the following we will use Problem 4.4 as our flagship example.

Trivial Algorithm. Test all the $\binom{n}{2}$ pairs of segments from S in $O(n^2)$ time and $O(n)$ space. For Problem 4.4 this is worst-case optimal because there may by $\Omega(n^2)$ intersecting pairs.

But in case that the number of intersecting pairs is, say, linear in n there is still hope to obtain a subquadratic algorithm. Given that there is a lower bound of $\Omega(n \log n)$ for *Element Uniqueness* (Given $x_1, \ldots, x_n \in \mathbb{R}$, is there an $i \neq j$ such that $x_i = x_j$?) in the algebraic computation tree model, all we can hope for is an output-sensitive runtime of the form $O(n \log n + k)$, where k denotes the number of intersecting pairs (output size).

4.1 Interval Intersections

As a warmup let us consider the corresponding problem in $\mathbb{R}^1$.

Problem 4.7 Given a set I of n intervals $[\ell_i, r_i] \subset \mathbb{R}$, $1 \leqslant i \leqslant n$. Compute all pairs of intervals from I that intersect.

Theorem 4.8 *Problem 4.7 can be solved in* $O(n \log n + k)$ *time and* $O(n)$ *space, where* k *is the number of intersecting pairs from* $\binom{I}{2}$.

Proof. First observe that two real intervals intersect if and only if one contains the right endpoint of the other.

Sort the set $\{(\ell_i, 0) \mid 1 \leqslant i \leqslant n\} \cup \{(r_i, 1) \mid 1 \leqslant i \leqslant n\}$ in increasing lexicographic order and denote the resulting sequence by P. Store along with each point from P its origin (i). Walk through P from start to end while maintaining a list L of intervals that contain the current point $p \in P$.

Whenever $p = (\ell_i, 0)$, $1 \leqslant i \leqslant n$, insert i into L. Whenever $p = (r_i, 1)$, $1 \leqslant i \leqslant n$, remove i from L and then report for all $j \in L$ the pair $\{i, j\}$ as intersecting. □

4.2 Segment Intersections

How can we transfer the (optimal) algorithm for the corresponding problem in $\mathbb{R}^1$ to the plane? In $\mathbb{R}^1$ we moved a point from left to right and at any point resolved the situation locally around this point. More precisely, at any point during the algorithm, we knew all

intersections that are to the left of the current (moving) point. A point can be regarded a hyperplane in $\mathbb{R}^1$, and the corresponding object in $\mathbb{R}^2$ is a line.

General idea. Move a line ℓ (so-called *sweep line*) from left to right over the plane, such that at any point during this process all intersections to the left of ℓ have been reported.

Sweep line status. The list of intervals containing the current point corresponds to a list L of segments (sorted by y-coordinate) that intersect the current sweep line ℓ. This list L is called *sweep line status* (SLS). Considering the situation locally around L, it is obvious that only segments that are adjacent in L can intersect each other. This observation allows to reduce the overall number of intersection tests, as we will see. In order to allow for efficient insertion and removal of segments, the SLS is usually implemented as a balanced binary search tree.

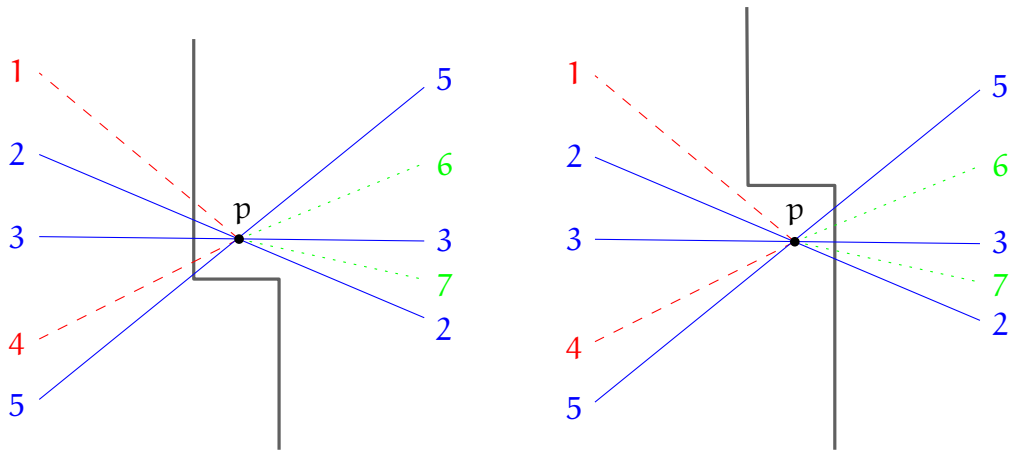
Event points. The order of segments in SLS can change at certain points only: whenever the sweep line moves over a segment endpoint or a point of intersection of two segments from S. Such a point is referred to as an *event point* (EP) of the sweep. Therefore we can reduce the conceptually continuous process of moving the sweep line over the plane to a discrete process that moves the line from EP to EP. This discretization allows for an efficient computation.

At any EP several events can happen simultaneously: several segments can start and/or end and at the same point a couple of other segments can intersect. In fact the sweep line does not even make a difference between any two event points that have the same x-coordinate. To properly resolve the order of processing, EPs are considered in lexicographic order and wherever several events happen at a single point, these are considered simultaneously as a single EP. In this light, the sweep line is actually not a line but an infinitesimal step function (see Figure 4.1).

Event point schedule. In contrast to the one-dimensional situation, in the plane not all EP are known in advance because the points of intersection are discovered during the algorithm only. In order to be able to determine the next EP at any time, we use a priority queue data structure, the so-called *event point schedule* (EPS).

Along with every EP p store a list $\text{end}(p)$ of all segments that end at p, a list $\text{begin}(p)$ of all segments that begin at p, and a list $\text{int}(p)$ of all segments in SLS that intersect at p a segment that is adjacent to it in SLS.

Along with every segment we store pointers to all its appearances in an $\text{int}(\cdot)$ list of some EP. As a segment appears in such a list only if it intersects one of its neighbors there, every segment needs to store at most two such pointers.



(a) Before. **(b)** After.

Figure 4.1: *Handling an event point* p. *Ending segments are shown red (dashed), starting segments green (dotted), and passing segments blue (solid).*

Invariants. The following conditions can be shown to hold before and after any event point is handled. (We will not formally prove this here.) In particular, the last condition at the end of the sweep implies the correctness of the algorithm.

1. L is the sequence of segments from S that intersect ℓ, ordered by y-coordinate of their point of intersection.
2. E contains all endpoints of segments from S and all points where two segments that are adjacent in L intersect to the right of ℓ.
3. All pairs from $\binom{S}{2}$ that intersect to the left of ℓ have been reported.

Event point handling. An EP p is processed as follows.

1. If $\text{end}(p) \cup \text{int}(p) = \emptyset$, localize p in L.
2. Report all pairs of segments from $\text{end}(p) \cup \text{begin}(p) \cup \text{int}(p)$ as intersecting.
3. Remove all segments in $\text{end}(p)$ from L.
4. Reverse the subsequence in L that is formed by the segments from $\text{int}(p)$.
5. Insert segments from $\text{begin}(p)$ into L, sorted by slope.
6. Test the topmost and bottommost segment in L from $\text{begin}(p) \cup \text{int}(p)$ for intersection with its successor and predecessor, respectively, and update EP if necessary.

Updating EPS. Insert an EP p corresponding to an intersection of two segments s and t. Without loss of generality let s be above t at the current position of ℓ.

1. If p does not yet appear in E, insert it.
2. If s is contained in an $\text{int}(\cdot)$ list of some other EP q, where it intersects a segment t' from above: Remove both s and t' from the $\text{int}(\cdot)$ list of q and possibly remove q from E (if $\text{end}(q) \cup \text{begin}(q) \cup \text{int}(q) = \emptyset$). Proceed analogously in case that t is contained in an $\text{int}(\cdot)$ list of some other EP, where it intersects a segment from below.
3. Insert s and t into $\text{int}(p)$.

Sweep.

1. Insert all segment endpoints into $\text{begin}(\cdot)/\text{end}(\cdot)$ list of a corresponding EP in E.
2. As long as $E \neq \emptyset$, handle the first EP and then remove it from E.

Runtime analysis. Initialization: $O(n \log n)$. Processing of an EP p:

$$O(\#\text{intersecting pairs} + |\text{end}(p)| \log n + |\text{int}(p)| + |\text{begin}(p)| \log n + \log n).$$

In total: $O(k + n \log n + k \log n) = O((n+k) \log n)$, where k is the number of intersecting pairs in S.

Space analysis. Clearly $|S| \leqslant n$. At begin we have $|E| \leqslant 2n$ and $|S| = 0$. Furthermore the number of additional EPs corresponding to points of intersection is always bounded by $2|S|$. Thus the space needed is $O(n)$.

Theorem 4.9 *Problem 4.4 and Problem 4.5 can be solved in* $O((n+k) \log n)$ *time and* $O(n)$ *space.* □

Theorem 4.10 *Problem 4.1, Problem 4.2 and Problem 4.3 can be solved in* $O(n \log n)$ *time and* $O(n)$ *space.* □

Exercise 4.11 *Flesh out the details of the sweep line algorithm for Problem 4.2 that is referred to in Theorem 4.10. What if you have to construct the intersection rather than just to decide whether or not it is empty?*

Exercise 4.12 *You are given* n *axis–parallel rectangles in* $\mathbb{R}^2$ *with their bottom sides lying on the* x*–axis. Construct their union in* $O(n \log n)$ *time.*

4.3 Improvements

The basic ingredients of the line sweep algorithm go back to work by Bentley and Ottmann [2] from 1979. The particular formulation discussed here, which takes all possible degeneracies into account, is due to Mehlhorn and Näher [9].

Theorem 4.10 is obviously optimal for Problem 4.3, because this is just the 2-dimensional generalization of Element Uniqueness (see Section 1.1). One might suspect there is also a similar lower bound of $\Omega(n \log n)$ for Problem 4.1 and Problem 4.2. However, this is not the case: Both can be solved in $O(n)$ time, albeit using a very complicated algorithm of Chazelle [4] (the famous triangulation algorithm).

Similarly, it is not clear why $O(n \log n+k)$ time should not be possible in Theorem 4.9. Indeed this was a prominent open problem in the 1980's that has been solved in several steps by Chazelle and Edelsbrunner in 1988. The journal version of their paper [5] consists of 54 pages and the space usage is suboptimal $O(n + k)$.

Clarkson and Shor [6] and independently Mulmuley [10, 11] described randomized algorithms with expected runtime $O(n \log n + k)$ using $O(n)$ and $O(n + k)$, respectively, space.

An optimal deterministic algorithm, with runtime $O(n \log n + k)$ and using $O(n)$ space, is known since 1995 only due to Balaban [1].

4.4 Algebraic degree of geometric primitives

We already have encountered different notions of complexity during this course: We can analyze the time complexity and the space complexity of algorithms and both usually depend on the size of the input. In addition we have seen examples of output-sensitive algorithms, whose complexity also depends on the size of the output. In this section, we will discuss a different kind of complexity that is the algebraic complexity of the underlying geometric primitives. In this way, we try to shed a bit of light into the bottom layer of geometric algorithms that is often swept under the rug because it consists of constant time operations only. But it would be a mistake to disregard this layer completely, because in most—if not every—real world application it plays a crucial role and the value of the constants involved can make a big difference.

In all geometric algorithms there are some fundamental geometric *predicates* and/or *constructions* on the bottom level. Both are operations on geometric objects, the difference is only in the result: The result of a construction is a geometric object (for instance, the point common to two non-parallel lines), whereas the result of a predicate is Boolean (*true* or *false*).

Geometric predicates and constructions in turn are based on fundamental arithmetic operations. For instance, we formulated planar convex hull algorithms in terms of an orientation predicate—given three points $p, q, r \in \mathbb{R}^2$, is r strictly to the right of the oriented line pr— which can be implemented using multiplication and addition/subtraction of coordinates/numbers. When using limited precision arithmetic, it is important to keep

an eye on the size of the expressions that occur during an evaluation of such a predicate.

In Exercise 3.22 we have seen that the orientation predicate can be computed by evaluating a polynomial of degree two in the input coordinates and, therefore, we say that

Proposition 4.13 *The rightturn/orientation predicate for three points in* $\mathbb{R}^2$ *has algebraic degree two.*

The degree of a predicate depends on the algebraic expression used to evaluate it. Any such expression is intimately tied to the representation of the geometric objects used. Where not stated otherwise, we assume that geometric objects are represented as described in Section 1.2. So in the proposition above we assume that points are represented using Cartesian coordinates. The situation might be different, if, say, a polar representation is used instead.

But even once a representation is agreed upon, there is no obvious correspondence to an algebraic expression that describes a given predicate. There are many different ways to form polynomials over the components of any representation, but it is desirable to keep the degree of the expression as small as possible. Therefore we define the *(algebraic) degree* of a geometric predicate or construction as the minimum degree of a polynomial that defines it. So there still is something to be shown in order to complete Proposition 4.13.

Exercise 4.14 *Show that there is no polynomial of degree at most one that describes the orientation test in* $\mathbb{R}^2$*.*

Hint: Let $p = (0, 0)$*,* $q = (x, 0)$*, and* $r = (0, y)$ *and show that there is no linear function in* x *and* y *that distinguishes the region where* pqr *form a rightturn from its complement.*

The degree of a predicate corresponds to the size of numbers that arise during its evaluation. If all input coordinates are k-bit integers then the numbers that occur during an evaluation of a degree d predicate on these coordinates are of size about[1] dk. If the number type used for the computation can represent all such numbers, the predicate can be evaluated exactly and thus always correctly. For instance, when using a standard IEEE double precision floating point implementation which has a mantissa length of 53 bit then the above orientation predicate can be evaluated exactly if the input coordinates are integers between 0 and 2^{25}, say.

Let us now get back to the line segment intersection problem. It needs a few new geometric primitives: most prominently, constructing the intersection point of two line segments.

[1] It is only *about* dk because not only multiplications play a role but also additions. As a rule of thumb, a multiplication may double the bitsize, while an addition may increase it by one.

Two segments. Given two line segments $s = \lambda a + (1-\lambda)b, \lambda \in [0,1]$ and $t = \mu c + (1-\mu)d, \mu \in [0,1]$, it is a simple exercise in linear algebra to compute $s \cap t$. Note that $s \cap t$ is either empty or a single point or a line segment.

For $a = (a_x, a_y)$, $b = (b_x, b_y)$, $c = (c_x, c_y)$, and $d = (d_x, d_y)$ we obtain two linear equations in two variables λ and μ.

$$\begin{aligned} \lambda a_x + (1-\lambda)b_x &= \mu c_x + (1-\mu)d_x \\ \lambda a_y + (1-\lambda)b_y &= \mu c_y + (1-\mu)d_y \end{aligned}$$

Rearranging terms yields

$$\begin{aligned} \lambda(a_x - b_x) + \mu(d_x - c_x) &= d_x - b_x \\ \lambda(a_y - b_y) + \mu(d_y - c_y) &= d_y - b_y \end{aligned}$$

Assuming that the lines underlying s and t have distinct slopes (that is, they are neither identical nor parallel) we have

$$D = \begin{vmatrix} a_x - b_x & d_x - c_x \\ a_y - b_y & d_y - c_y \end{vmatrix} = \begin{vmatrix} a_x & a_y & 1 & 0 \\ b_x & b_y & 1 & 0 \\ c_x & c_y & 0 & 1 \\ d_x & d_y & 0 & 1 \end{vmatrix} \neq 0$$

and using Cramer's rule

$$\lambda = \frac{1}{D} \begin{vmatrix} d_x - b_x & d_x - c_x \\ d_y - b_y & d_y - c_y \end{vmatrix} \text{ and } \mu = \frac{1}{D} \begin{vmatrix} a_x - b_x & d_x - b_x \\ a_y - b_y & d_y - b_y \end{vmatrix}.$$

To test if s and t intersect, we can—after having sorted out the degenerate case in which both segments have the same slope—compute λ and μ and then check whether $\lambda, \mu \in [0,1]$.

Observe that both λ and D result from multiplying two differences of input coordinates. Computing the x-coordinate of the point of intersection via $b_x + \lambda(a_x - b_x)$ uses another multiplication. Overall this computation yields a fraction whose numerator is a polynomial of degree three in the input coordinates and whose denominator is a polynomial of degree two in the input coordinates.

In order to maintain the sorted order of event points in the EPS, we need to compare event points lexicographically. In case that both are intersection points, this amounts to comparing two fractions of the type discussed above. In order to formulate such a comparison as a polynomial, we have to cross-multiply the denominators, and so obtain a polynomial of degree $3 + 2 = 5$. It can be shown (but we will not do it here) that this bound is tight and so we conclude that

Proposition 4.15 *The algebraic degree of the predicate that compares two intersection points of line segments lexicographically is five.*

Therefore the coordinate range in which this predicate can be evaluated exactly using IEEE double precision numbers shrinks down to integers between 0 and about $2^{10} = 1'024$.

Exercise 4.16 *What is the algebraic degree of the predicate checking whether two line segments intersect? (Above we were interested in the actual intersection point, but now we consider the predicate that merely answers the question whether two segments intersect or not by yes or no).*

Exercise 4.17 *What is the algebraic degree of the InCircle predicate? More precisely, you are given three points* $\mathrm{p}, \mathrm{q}, \mathrm{r}$ *in the plane that define a circle* C *and a fourth point* s*. You want to know if* s *is inside* C *or not. What is the degree of the polynomial(s) you need to evaluate to answer this question?*

4.5 Red-Blue Intersections

Although the Bentley-Ottmann sweep appears to be rather simple, its implementation is not straightforward. For once, the original formulation did not take care of possible degeneracies—as we did in the preceding section. But also the algebraic degree of the predicates used is comparatively high. In particular, comparing two points of intersection lexicographically is a predicate of degree five, that is, in order to compute it, we need to evaluate a polynomial of degree five. When evaluating such a predicate with plain floating point arithmetic, one easily gets incorrect results due to limited precision roundoff errors. Such failures frequently render the whole computation useless. The line sweep algorithm is problematic in this respect, as a failure to detect one single point of intersection often implies that no other intersection of the involved segments to the right is found.

In general, predicates of degree four are needed to construct the arrangement of line segments because one needs to determine the orientation of a triangle formed by three segments. This is basically an orientation test where two points are segment endpoints and the third point is an intersection point of segments. Given that the coordinates of the latter are fractions, whose numerator is a degree three polynomial and the common denominator is a degree two polynomial, we obtain a degree four polynomial overall.

Motivated by the Map overlay application we consider here a restricted case. In the *red-blue intersection* problem, the input consists of two sets R (red) and B (blue) of segments such that the segments in each set are *interior-disjoint*, that is, for any pair of distinct segments their relative interior is disjoint. In this case there are no triangles of intersecting segments, and it turns out that predicates of degree two suffice to construct the arrangement. This is optimal because already the intersection test for two segments is a predicate of degree two.

Predicates of degree two. Restricting to degree two predicates has certain consequences. While it is possible to determine the position of a segment endpoint relative to a(nother)

segment using an orientation test, one cannot, for example, compare a segment endpoint with a point of intersection lexicographically. Even computing the coordinates for a point of intersection is not possible. Therefore the output of intersection points is done implicitly, as "intersection of s and t".

Graphically speaking we can deform any segment—keeping its endpoints fixed—as long as it remains monotone and it does not reach or cross any segment endpoint (it did not touch before). With help of degree two predicates there is no way to tell the difference.

Witnesses. Using such transformations the processing of intersection points is deferred as long as possible (lazy computation). The last possible point $w(s,t)$ where an intersection between two segments s and t has to be processed we call the *witness* of the intersection. The witness $w(s,t)$ is the lexicographically smallest segment endpoint that is located within the closed wedge formed by the two intersecting segments s and t to the right of the point of intersection (Figure 4.2). Note that each such wedge contains at least two segment endpoints, namely the right endpoints of the two intersecting segments. Therefore for any pair of intersecting segments its witness is well-defined.

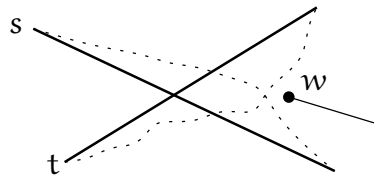


Figure 4.2: *The witness* $w(s,t)$ *of an intersection* $s \cap t$.

As a consequence, only the segment endpoints are EPs and the EPS can be determined by lexicographic sorting during initialization. On the other hand, the SLS structure gets more complicated because its order of segments does not necessarily reflect the order in which the segments intersect the sweep line.

Invariants. The invariants of the algorithm have to take the relaxed notion of order into account. Denote the sweep line by ℓ.

1. L is the sequence of segments from S intersecting ℓ; s appears before t in $L \Longrightarrow s$ intersects ℓ above t or s intersects t and the witness of this intersection is to the right of ℓ.

2. All intersections of segments from S whose witness is to the left of ℓ have been reported.

SLS Data Structure. The SLS structure consist of three levels. We use the fact that segments of the same color do not interact, except by sharing endpoints possibly.

1. Collect adjacent segments of the same color in *bundles*, stored as balanced search trees. For each bundle store pointers to the topmost and bottommost segment. (As the segments within one bundle are interior-disjoint, their order remains static and thus correct under possible deformations due to lazy computation.)

2. All bundles are stored in a doubly linked list, sorted by y-coordinate.

3. All red bundles are stored in a balanced search tree (*bundle tree*).

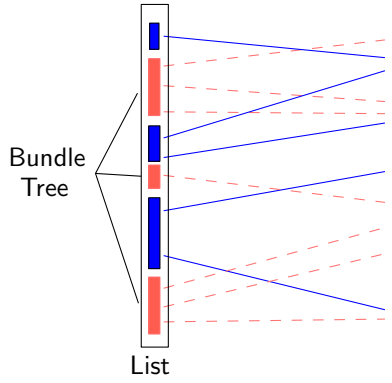


Figure 4.3: *Graphical representation of the SLS data structure.*

The search tree structure should support insert, delete, split and merge in (amortized) logarithmic time each. For instance, splay trees [12] meet these requirements. (If you have never heard about splay trees so far, you do not need to know for our purposes here. Just treat them as a black box. However, you should take a note and read about splay trees still. They are a fascinating data structure.)

EP handling. An EP p is processed as follows.

1. Localize p in bundle tree $\rightarrow \leqslant 2$ bundles containing p (all red bundles in between must end at p). For the localization we use the pointers to the topmost and bottommost segment within a bundle.

2. Localize p in $\leqslant 2$ red bundles found and split them at p. (If p is above the topmost or below the bottommost segment of the bundle, there is no split.) All red bundles are now either above, ending, or below with respect to p.

3. Localize p within the blue bundles by linear search.

4. Localize p in the $\leqslant 2$ blue bundles found and split them at p. (If p is above the topmost or below the bottommost segment of the bundle, there is no split.) All bundles are now either above, ending, or below with respect to p.

5. Run through the list of bundles around p. More precisely, start from one of the bundles containing p found in Step 1 and walk up in the doubly linked list of bundles until two successive bundles (hence of opposite color) are both above p. Similarly, walk down in the doubly linked list of bundles, until two successive bundles are both below p. Handle all adjacent pairs of bundles (A, B) that are in wrong order and report all pairs of segments as intersecting. (Exchange A and B in the bundle list and merge them with their new neighbors.)

6. Report all pairs from $\text{begin}(p) \times \text{end}(p)$ as intersecting.

7. Remove ending bundles and insert starting segments, sorted by slope and bundled by color and possibly merge with the closest bundle above or below.

Remark: As both the red and the blue segments are interior-disjoint, at every EP there can be at most one segment that passes through the EP. Should this happen, for the purpose of EP processing split this segment into two, one ending and one starting at this EP. But make sure to not report an intersection between the two parts!

Analysis. Sorting the EPS: $O(n \log n)$ time. Every EP generates a constant number of tree searches and splits of $O(\log n)$ each. Every exchange in Step 5 generates at least one intersection. New bundles are created only by inserting a new segment or by one of the constant number of splits at an EP. Therefore $O(n)$ bundles are created in total. The total number of merge operations is $O(n)$, as every merge kills one bundle and $O(n)$ bundles are created overall. The linear search in steps 3 and 5 can be charged either to the ending bundle or—for continuing bundles—to the subsequent merge operation. In summary, we have a runtime of $O(n \log n + k)$ and space usage is linear obviously.

Theorem 4.18 *For two sets R and B, each consisting of interior-disjoint line segments in $\mathbb{R}^2$, one can find all intersecting pairs of segments in $O(n \log n + k)$ time and linear space, using predicates of maximum degree two. Here $n = |R| + |B|$ and k is the number of intersecting pairs.*

Remarks. The first optimal algorithm for the red-blue intersection problem was published in 1988 by Harry Mairson and Jorge Stolfi [7]. In 1994 Timothy Chan [3] described a trapezoidal-sweep algorithm that uses predicates of degree three only. The approach discussed above is due to Andrea Mantler and Jack Snoeyink [8] from 2000.

Exercise 4.19 *Let S be a set of n segments each of which is either horizontal or vertical. Describe an $O(n \log n)$ time and $O(n)$ space algorithm that counts the number of pairs in $\binom{S}{2}$ that intersect.*

Questions

11. *How can one test whether a polygon on* n *vertices is simple?* Describe an $O(n \log n)$ time algorithm.

12. *How can one test whether two simple polygons on altogether* n *vertices intersect?* Describe an $O(n \log n)$ time algorithm.

13. *How does the line sweep algorithm work that finds all* k *intersecting pairs among* n *line segments in* $\mathbb{R}^2$*?* Describe the algorithm, using $O((n + k) \log n)$ time and $O(n)$ space. In particular, explain the data structures used, how event points are handled, and how to cope with degeneracies.

14. *Given two line segments* s *and* t *in* $\mathbb{R}^2$ *whose endpoints have integer coordinates in* $[0, 2^b)$*; suppose* s *and* t *intersect in a single point* q*, what can you say about the coordinates of* q*?* Give a good (tight up to small additive number of bits) upper bound on the size of these coordinates. (No need to prove tightness, that is, give an example which achieves the bound.)

15. *What is the degree of a predicate and why is it an important parameter? What is the degree of the orientation test and the incircle test in* $\mathbb{R}^2$*?* Explain the term and give two reasons for its relevance. Provide tight upper bounds for the two predicates mentioned. (No need to prove tightness.)

16. *What is the map overlay problem and how can it be solved optimally using degree two predicates only?* Give the problem definition and explain the term "interior-disjoint". Explain the bundle-sweep algorithm, in particular, the data structures used, how an event point is processed, and the concept of witnesses.

References

[1] Ivan J. Balaban, An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pp. 211–219, 1995, URL http://dx.doi.org/10.1145/220279.220302.

[2] Jon L. Bentley and Thomas A. Ottmann, Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, **C-28**, 9, (1979), 643–647, URL http://dx.doi.org/10.1109/TC.1979.1675432.

[3] Timothy M. Chan, A simple trapezoid sweep algorithm for reporting red/blue segment intersections. In *Proc. 6th Canad. Conf. Comput. Geom.*, pp. 263–268, 1994, URL https://cs.uwaterloo.ca/~tmchan/red_blue.ps.gz.

[4] Bernard Chazelle, Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, **6**, 5, (1991), 485–524, URL http://dx.doi.org/10.1007/BF02574703.

[5] Bernard Chazelle and H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane. *J. ACM*, **39**, 1, (1992), 1–54, URL http://dx.doi.org/10.1145/147508.147511.

[6] Kenneth L. Clarkson and Peter W. Shor, Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, **4**, (1989), 387–421, URL http://dx.doi.org/10.1007/BF02187740.

[7] Harry G. Mairson and Jorge Stolfi, Reporting and counting intersections between two sets of line segments. In R. A. Earnshaw, ed., *Theoretical Foundations of Computer Graphics and CAD*, vol. 40 of *NATO ASI Series F*, pp. 307–325, Springer-Verlag, Berlin, Germany, 1988.

[8] Andrea Mantler and Jack Snoeyink, Intersecting red and blue line segments in optimal time and precision. In J. Akiyama, M. Kano, and M. Urabe, eds., *Proc. Japan Conf. Discrete Comput. Geom.*, vol. 2098 of *Lecture Notes Comput. Sci.*, pp. 244–251, Springer Verlag, 2001, URL http://dx.doi.org/10.1007/3-540-47738-1_23.

[9] Kurt Mehlhorn and Stefan Näher, Implementation of a sweep line algorithm for the straight line segment intersection problem. Report MPI-I-94-160, Max-Planck-Institut Inform., Saarbrücken, Germany, 1994.

[10] Ketan Mulmuley, A Fast Planar Partition Algorithm, I. *J. Symbolic Comput.*, **10**, 3-4, (1990), 253–280, URL http://dx.doi.org/10.1016/S0747-7171(08)80064-8.

[11] Ketan Mulmuley, A fast planar partition algorithm, II. *J. ACM*, **38**, (1991), 74–103, URL http://dx.doi.org/10.1145/102782.102785.

[12] Daniel D. Sleator and Robert E. Tarjan, Self-adjusting binary search trees. *J. ACM*, **32**, 3, (1985), 652–686, URL http://dx.doi.org/10.1145/3828.3835.

Chapter 5

Plane Graphs and the DCEL

So far we have been talking about geometric structures such as triangulations of polygons and arrangements of line segments without paying much attention to how to represent these structures. This will change now. As all of these structures can be regarded as plane graphs, we start by reviewing a bit of terminology from (planar) graph theory. We assume that this is—at least somewhat—familiar ground for you. If you would like to see more details and examples, please refer to any standard textbook on graph theory, such as Bondy and Murty [2], Diestel [3], or West [10].

A (simple undirected) *graph* is a pair $G = (V, E)$ where V is a finite set of *vertices* and E is the set of *edges*, $E \subseteq \binom{V}{2} := \{\{u, v\} \mid u, v \in V, u \neq v\}$. For brevity, one often writes uv rather than $\{u, v\}$ to denote an edge. The two vertices defining an edge are *adjacent* to each other and they are *incident* to the edge.

For a vertex $v \in V$, denote by $N_G(v)$ the *neighborhood* of v in G, that is, the set of vertices from G that are adjacent to v. Similarly, for a set $W \subset V$ of vertices define $N_G(W) := \bigcup_{w \in W} N_G(w)$. The *degree* $\deg_G(v)$ of a vertex $v \in V$ is the size of its neighborhood, that is, the number of edges from E incident to v. The subscript is often omitted when it is clear to which graph it refers to. As every edge is incident to exactly two vertices, we have the following simple but useful relationship, often called *handshaking-lemma*: $\sum_{v \in V} \deg(v) = 2|E|$.

A *walk* in a graph G is a sequence $W = (v_1, \ldots, v_k)$, $k \in \mathbb{N}$, of vertices such that v_i and v_{i+1} are adjacent in G, for all $1 \leqslant i < k$. A *path* is a walk whose vertices are pairwise distinct. A *cycle* is a walk whose vertices are pairwise distinct, except for $v_1 = v_k$. A graph is *connected*, if there is a path between any pair of vertices. If a graph is not connected, it is *disconnected*. A disconnected graph can be decomposed into maximal connected subgraphs, its (connected) *components*.

A *tree* is a connected graph that does not have any cycle. It is not hard to show that trees on n vertices are exactly the graphs on n vertices that have $n - 1$ edges.

In geometry we are typically concerned with graphs that are embedded on some surface, here $\mathbb{R}^2$. An *embedding* or *drawing* is just a "nice" mapping of a graph G into the plane. More formally, each vertex v is mapped to a distinct point $\phi(v) \in \mathbb{R}^2$ (that is, $\phi : V \to \mathbb{R}^2$ is injective) and each edge uv is mapped to an *arc*—a simple Jordan

curve—$\phi(uv) \subset \mathbb{R}^2$ from $\phi(u)$ to $\phi(v)$ such that no vertex is mapped to the relative interior of $\phi(uv)$.

A graph is *planar* if it admits a *crossing-free* drawing, that is, a drawing in which no two arcs intersect except at common endpoints. For example, K_4 (the complete graph on four vertices) is planar, as demonstrated by the drawing shown in Figure 5.1a.

(a) Crossing-free drawing. **(b)** Straight-line drawing.

Figure 5.1: *Embeddings of* K_4 *into the plane.*

If a graph is planar, then by Fáry-Wagner's Theorem [4, 8] there also exists a *straight-line* drawing, that is, a drawing in which all arcs are line segments. In order to obtain such a drawing for K_4, we have to put one vertex inside the convex hull of the other three, see Figure 5.1b.

Sometimes we refer to graphs not as abstract graphs but in a concrete embedding. If this embedding is a crossing-free embedding into the plane, the embedded graph is referred to as a *plane graph*. Note the distinction between "planar" and "plane": The former refers to an abstract graph and expresses the possibility of a crossing-free drawing, whereas the latter refers to a geometric object that is a concrete crossing-free drawing of a graph in the plane.

A *plane straight-line graph* (PSLG) is a crossing-free straight-line drawing of a graph in the plane. Both graphs in Figure 5.1 are plane, but only the one shown on the right is also a plane straight-line graph.

5.1 The Euler Formula

If we remove all vertices (points) and edges (arcs) of a plane graph G from $\mathbb{R}^2$, then what remains is a finite collection of open sets. These sets are referred to as the *faces* of G. For instance, both plane graphs in Figure 5.1 have 4 vertices, 6 edges and 4 faces (one of which is unbounded). In general, if $|V|$ is the number of vertices of a connected plane graph, $|E|$ its number of edges and $|F|$ the number of faces, then the Euler Formula states that

$$|V| - |E| + |F| = 2.$$

In the example, we get $4 - 6 + 4 = 2$.

If you do not insist on being too formal, the proof is simple and works by induction on the number of edges. If we fix the number of vertices, the base case occurs for $|V| - 1$

edges where the plane graph is a tree. Then we have $|F| = 1$ and the formula holds. A graph with more edges always contains a cycle and therefore at least one bounded face. Choose one edge from a bounded face and remove it. The resulting graph is still connected and has one edge less but also one face less since the edge removal merges the bounded face with another one. Consequently, since the Euler Formula holds for the smaller graph by induction, it also holds for the larger graph.

The Euler Formula can be used to prove the following important fact about planar graphs.

Lemma 5.1 *Any planar graph on $n \geqslant 3$ vertices has at most $3n-6$ edges and at most $2n-4$ faces.*

This lemma shows that the overall complexity—the sum of the number of vertices, edges, and faces—of a planar graph is linear in n. A planar graph with a maximal number $(3n-6)$ of edges is called *maximal planar*.

Exercise 5.2 *Prove Lemma 5.1 using the Euler formula.*

Exercise 5.3 *Prove that every planar graph has a vertex of degree at most 5.*

Exercise 5.4 *Show that K_5 (the complete graph on five vertices) is not planar.*

5.2 The Doubly-Connected Edge List

Many algorithms in computational geometry work with plane graphs. The *doubly-connected edge list* (DCEL) is a data structure to represent a plane graph in such a way that it is easy to traverse and to manipulate. In order to avoid unnecessary complications, let us discuss only connected graphs here that contain at least two vertices. It is not hard to extend the data structure to cover all plane graphs. For simplicity we also assume that we deal with a straight-line embedding and so the geometry of edges is defined by the mapping of their endpoints already. For more general embeddings, the geometric description of edges has to be stored in addition.

The main building block of a DCEL is a list of *halfedges*. Every actual edge is represented by two halfedges going in opposite direction, and these are called *twins*, see Figure 5.2. Along the boundary of each face, halfedges are oriented counterclockwise.

A DCEL stores a list of halfedges, a list of vertices, and a list of faces. These lists are unordered but interconnected by various pointers. A vertex v stores a pointer $\text{halfedge}(v)$ to an arbitrary halfedge originating from v. Every vertex also knows its coordinates, that is, the point $\text{point}(v)$ it is mapped to in the represented embedding. A face f stores a pointer $\text{halfedge}(f)$ to an arbitrary halfedge within the face. A halfedge h stores *five* pointers:

- a pointer $\text{target}(h)$ to its target vertex,

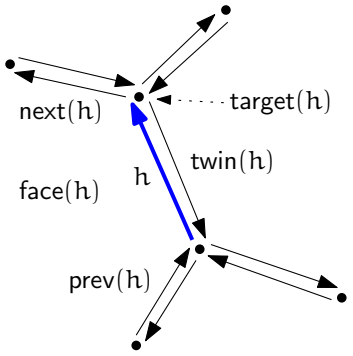


Figure 5.2: *A halfedge in a DCEL.*

- a pointer face(h) to the incident face,
- a pointer twin(h) to its twin halfedge,
- a pointer next(h) to the halfedge following h along the boundary of face(h), and
- a pointer prev(h) to the halfedge preceding h along the boundary of face(h).

A constant amount of information is stored for every vertex, (half-)edge, and face of the graph. Therefore the whole DCEL needs storage proportional to $|V| + |E| + |F|$, which is $O(n)$ for a plane graph with n vertices by Lemma 5.1.

This information is sufficient for most tasks. For example, traversing all edges around a face f can be done as follows:

```
s ← halfedge(f)
h ← s
do
    something with h
    h ← next(h)
while h ≠ s
```

Exercise 5.5 *Give pseudocode to traverse all edges incident to a given vertex v of a DCEL.*

Exercise 5.6 *Why is the previous halfedge* prev(·) *stored explicitly and the source vertex of a halfedge is not?*

5.2.1 Manipulating a DCEL

In many geometric algorithms, plane graphs appear not just as static objects but rather they evolve over the course of the algorithm. Therefore the data structure used to represent the graph must allow for efficient update operations to change it.

First of all, we need to be able to generate new vertices, edges, and faces, to be added to the corresponding list within the DCEL and—symmetrically—the ability to delete an existing entity. Then it should be easy to add a new vertex v to the graph within some face f. As we maintain a connected graph, we better link the new vertex to somewhere, say, to an existing vertex u. For such a connection to be possible, we require that the open line segment uv lies completely in f.

Of course, two halfedges are to be added connecting u and v. But where exactly? Given that from a vertex and from a face only some arbitrary halfedge is directly accessible, it turns out convenient to use a halfedge in the interface. Let h denote the halfedge incident to f for which $\mathsf{target}(h) = u$. Our operation becomes then (see also Figure 5.3)

add-vertex-at(v, h)
Precondition: the open line segment $\overline{\mathsf{point}(v)\mathsf{point}(u)}$, where $u := \mathsf{target}(h)$, lies completely in $f := \mathsf{face}(h)$.
Postcondition: a new vertex v has been inserted into f, connected by an edge to u.

(a) before **(b)** after

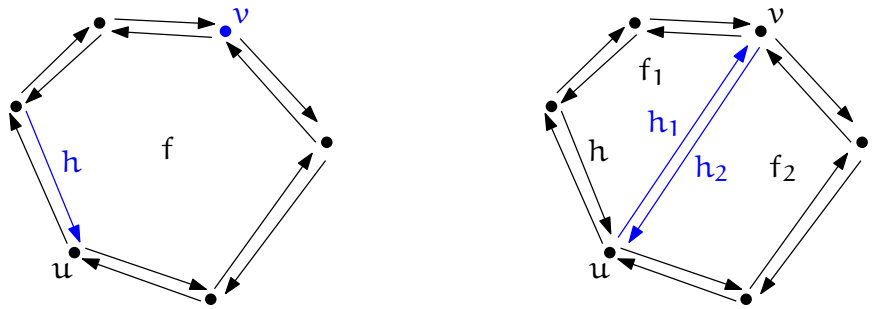
Figure 5.3: *Add a new vertex connected to an existing vertex* u.

and it can be realized by manipulating a constant number of pointers as follows.

```
add-vertex-at(v, h) {
    h1 ← a new halfedge
    h2 ← a new halfedge
    halfedge(v) ← h2
    twin(h1) ← h2
    twin(h2) ← h1
    target(h1) ← v
    target(h2) ← u
    face(h1) ← f
    face(h2) ← f
    next(h1) ← h2
    next(h2) ← next(h)
    prev(h1) ← h
```

$\mathsf{prev}(h_2) \leftarrow h_1$
$\mathsf{next}(h) \leftarrow h_1$
$\mathsf{prev}(\mathsf{next}(h_2)) \leftarrow h_2$
}

Similarly, it should be possible to add an edge between two existing vertices u and v, provided the open line segment uv lies completely within a face f of the graph, see Figure 5.4. Since such an edge insertion splits f into two faces, the operation is called *split-face*. Again we use the halfedge h that is incident to f and for which $\mathsf{target}(h) = u$.



(a) before **(b)** after

Figure 5.4: *Split a face by an edge* uv.

Our operation becomes then

$\mathsf{split\text{-}face}(h, v)$
Precondition: v is incident to $f := \mathsf{face}(h)$ but not adjacent to $u := \mathsf{target}(h)$.
The open line segment $\overline{\mathsf{point}(v)\mathsf{point}(u)}$ lies completely in f.
Postcondition: f has been split by a new edge uv.

The implementation is slightly more complicated compared to add-vertex-at above, because the face f is destroyed and so we have to update the face information of all incident halfedges. In particular, this is not a constant time operation, but its time complexity is proportional to the size of f.

$\mathsf{split\text{-}face}(h, v)$ {
$f_1 \leftarrow$ a new face
$f_2 \leftarrow$ a new face
$h_1 \leftarrow$ a new halfedge
$h_2 \leftarrow$ a new halfedge
$\mathsf{halfedge}(f_1) \leftarrow h_1$
$\mathsf{halfedge}(f_2) \leftarrow h_2$
$\mathsf{twin}(h_1) \leftarrow h_2$
$\mathsf{twin}(h_2) \leftarrow h_1$
$\mathsf{target}(h_1) \leftarrow v$

```
    target(h_2) ← u
    next(h_2) ← next(h)
    prev(next(h_2)) ← h_2
    prev(h_1) ← h
    next(h) ← h_1
    i ← h_2
    loop
        face(i) ← f_2
        if target(i) = v break the loop
        i ← next(i)
    endloop
    next(h_1) ← next(i)
    prev(next(h_1)) ← h_1
    next(i) ← h_2
    prev(h_2) ← i
    i ← h_1
    do
        face(i) ← f_1
        i ← next(i)
    until target(i) = u
    delete the face f
}
```

In a similar fashion one can realize the inverse operation join-face(h) that removes the edge (represented by the halfedge) h, thereby joining the faces face(h) and face(twin(h)).

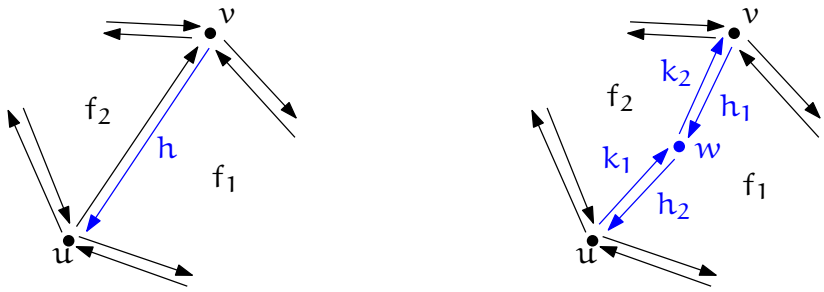
It is easy to see that every connected plane graph on at least two vertices can be constructed using the operations add-vertex-at and split-face, starting from an embedding of K_2 (two vertices connected by an edge).

Exercise 5.7 *Give pseudocode for the operation join-face*(h)*. Also specify preconditions, if needed.*

Exercise 5.8 *Give pseudocode for the operation split-edge*(h)*, that splits the edge (represented by the halfedge)* h *into two by a new vertex* w*, see Figure 5.5.*

5.2.2 Graphs with Unbounded Edges

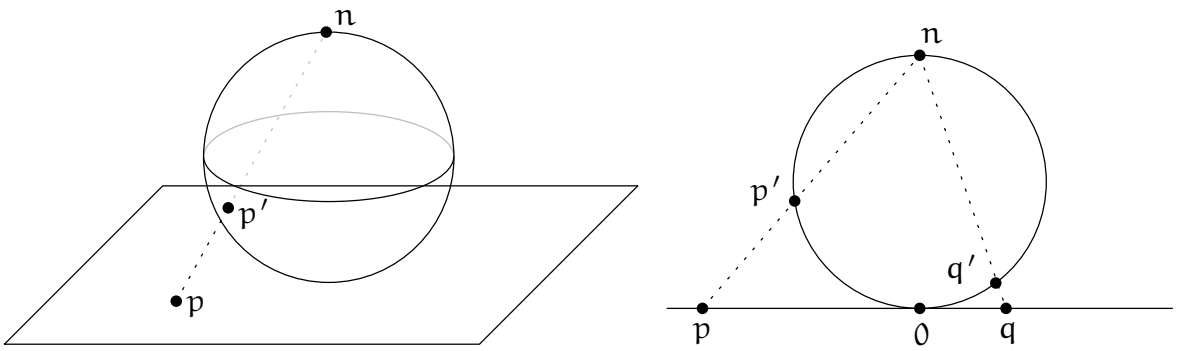
In some cases it is convenient to consider plane graphs, in which some edges are not mapped to a line segment but to an unbounded curve, such as a ray. This setting is not really much different from the one we studied before, except that one vertex is placed "at infinity". One way to think of it is in terms of *stereographic projection*: Imagine $\mathbb{R}^2$ being the x/y-plane in $\mathbb{R}^3$ and place a unit sphere S such that its southpole touches the origin. We obtain a bijective continuous mapping between $\mathbb{R}^2$ and $S \setminus \{n\}$, where n



(a) before (b) after

Figure 5.5: *Split an edge by a new vertex.*

is the northpole of S, as follows: A point $p \in \mathbb{R}^2$ is mapped to the point p' that is the intersection of the line through p and n with S, see Figure 5.6. The further away a point



(a) Three-dimensional view. (b) Cross-section view.

Figure 5.6: *Stereographic projection.*

in $\mathbb{R}^2$ is from the origin, the closer its image on S gets to n. But there is no way to reach n except in the limit. Therefore, we can imagine drawing the graph on S instead of in $\mathbb{R}^2$ and putting the "infinite vertex" at n.

All this is just for the sake of a proper geometric interpretation. As far as a DCEL representation of such a graph is concerned, there is no need to consider spheres or, in fact, anything beyond what we have discussed before. The only difference to the case with all finite edges is that there is this special infinite vertex, which does not have any point/coordinates associated to it. But other than that, the infinite vertex is treated in exactly the same way as the finite vertices: it has in– and outgoing halfedges along which the unbounded faces can be traversed (Figure 5.7).

5.2.3 Remarks

It is actually not so easy to point exactly to where the DCEL data structure originates from. Often Muller and Preparata [7] are credited, but while they use the term DCEL,

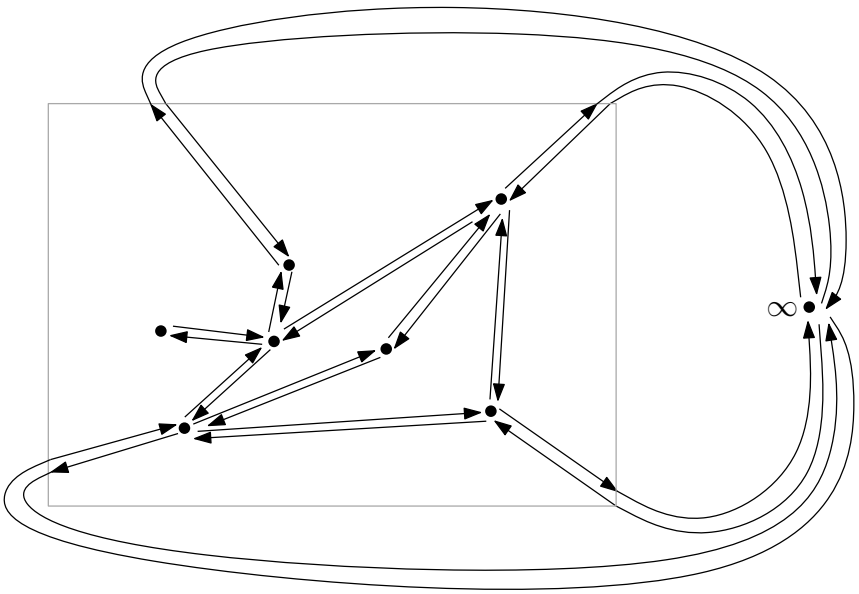


Figure 5.7: *A DCEL with unbounded edges. Usually, we will not show the infinite vertex and draw all edges as straight-line segments. This yields a geometric drawing, like the one within the gray box.*

the data structure they describe is different from what we discussed above and from what people usually consider a DCEL nowadays. Overall, there is a large number of variants of this data structure, which appear under the names *winged edge* data structure [1], *halfedge* data structure [9], or *quad-edge* data structure [5]. Kettner [6] provides a comparison of all these and some additional references.

Questions

17. *What are planar/plane graphs and straight-line embeddings?* Give the definitions and explain the difference between planar and plane.

18. *How many edges can a planar graph have? What is the average vertex degree in a planar graph?* Explain Euler's formula and derive your answers from it (see Exercise 5.2 and 5.3).

19. *How can plane graphs be represented on a computer?* Explain the DCEL data structure and how to work with it.

References

[1] Bruce G. Baumgart, A polyhedron representation for computer vision. In *Proc. AFIPS Natl. Comput. Conf.*, vol. 44, pp. 589–596, AFIPS Press, Alrington, Va., 1975, URL `http://dx.doi.org/10.1145/1499949.1500071`.

[2] John Adrian Bondy and U. S. R. Murty, *Graph Theory*, vol. 244 of *Graduate texts in Mathematics*. Springer-Verlag, New York, 2008, URL http://dx.doi.org/10.1007/978-1-84628-970-5.

[3] Reinhard Diestel, *Graph Theory*. Springer-Verlag, Heidelberg, 4th edn., 2010.

[4] István Fáry, On straight lines representation of planar graphs. *Acta Sci. Math. Szeged*, **11**, (1948), 229–233.

[5] Leonidas J. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, **4**, 2, (1985), 74–123, URL http://dx.doi.org/10.1145/282918.282923.

[6] Lutz Kettner, *Software design in computational geometry and contour-edge based polyhedron visualization*. Ph.D. thesis, ETH Zürich, Zürich, Switzerland, 1999, URL http://dx.doi.org/10.3929/ethz-a-003861002.

[7] David E. Muller and Franco P. Preparata, Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, **7**, (1978), 217–236, URL http://dx.doi.org/10.1016/0304-3975(78)90051-8.

[8] Klaus Wagner, Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, **46**, (1936), 26–32.

[9] Kevin Weiler, Edge-based data structures for solid modeling in a curved surface environment. *IEEE Comput. Graph. Appl.*, **5**, 1, (1985), 21–40, URL http://dx.doi.org/10.1109/MCG.1985.276271.

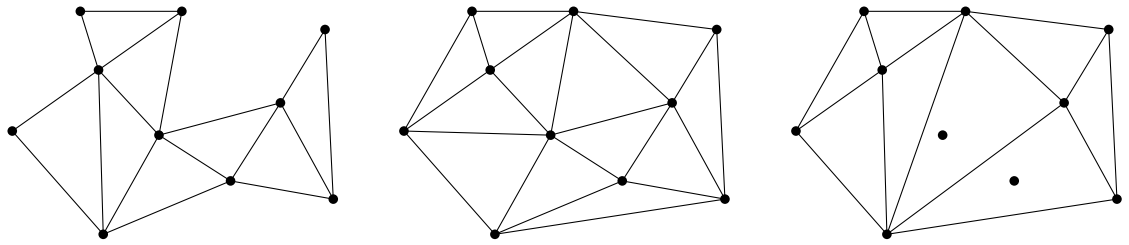
[10] Douglas B. West, *An Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, 2nd edn., 2001.

Chapter 6

Delaunay Triangulations

In Chapter 2 we have discussed triangulations of simple polygons. A triangulation nicely partitions a polygon into triangles, which allows, for instance, to easily compute the area or a guarding of the polygon. Another typical application scenario is to use a triangulation T for interpolation: Suppose a function f is defined on the vertices of the polygon P, and we want to extend it "reasonably" and continuously to P°. Then for a point $p \in P^\circ$ find a triangle t of T that contains p. As p can be written as a convex combination $\sum_{i=1}^{3} \lambda_i v_i$ of the vertices v_1, v_2, v_3 of t, we just use the same coefficients to obtain an interpolation $f(p) := \sum_{i=1}^{3} \lambda_i f(v_i)$ of the function values.

If triangulations are a useful tool when working with polygons, they might also turn out useful to deal with other geometric objects, for instance, point sets. But what could be a triangulation of a point set? Polygons have a clearly defined interior, which naturally lends itself to be covered by smaller polygons such as triangles. A point set does not have an interior, except ... Here the notion of convex hull comes handy, because it allows us to treat a point set as a convex polygon. Actually, not really a convex polygon, because points in the interior of the convex hull should not be ignored completely. But one way to think of a point set is as a convex polygon—its convex hull—possibly with some holes—which are points—in its interior. A triangulation should then partition the convex hull while respecting the points in the interior, as shown in the example in Figure 6.1b.



(a) Simple polygon triangulation. **(b)** Point set triangulation. **(c)** Not a triangulation.

Figure 6.1: *Examples of (non-)triangulations.*

In contrast, the example depicted in Figure 6.1c nicely subdivides the convex hull

but should not be regarded a triangulation: Two points in the interior are not respected but simply swallowed by a large triangle.
This interpretation directly leads to the following adaption of Definition 2.12.

Definition 6.1 *A* **triangulation** *of a finite point set* $P \subset \mathbb{R}^2$ *is a collection* $\mathcal{T}$ *of triangles, such that*

(1) $\operatorname{conv}(P) = \bigcup_{T \in \mathcal{T}} T$*;*

(2) the vertices of all triangles in T *are points from* P*;*

(3) if $p \in t$*, for any* $p \in P$ *and* $t \in \mathcal{T}$*, then* p *is a vertex of* t*; and*

(4) for every distinct pair $T, U \in \mathcal{T}$*, the intersection* $T \cap U$ *is either a common vertex, or a common edge, or empty.*

Note that due to (4) we could also combine (2) and (3) to read "the vertices of the triangles in T are exactly the points from P".

Just as for polygons, triangulations are universally available for point sets, meaning that (almost) every point set admits at least one.

Proposition 6.2 *Every set* $P \subseteq \mathbb{R}^2$ *of* $n \geqslant 3$ *points has a triangulation, unless all points in* P *are collinear.*

Proof. In order to construct a triangulation for P, consider the lexicographically sorted sequence $p_1, \ldots, p_n$ of points in P. Let m be minimal such that $p_1, \ldots, p_m$ are not collinear. We triangulate $p_1, \ldots, p_m$ by connecting p_m to all of $p_1, \ldots, p_{m-1}$ (which *are* on a common line), see Figure 6.2a.

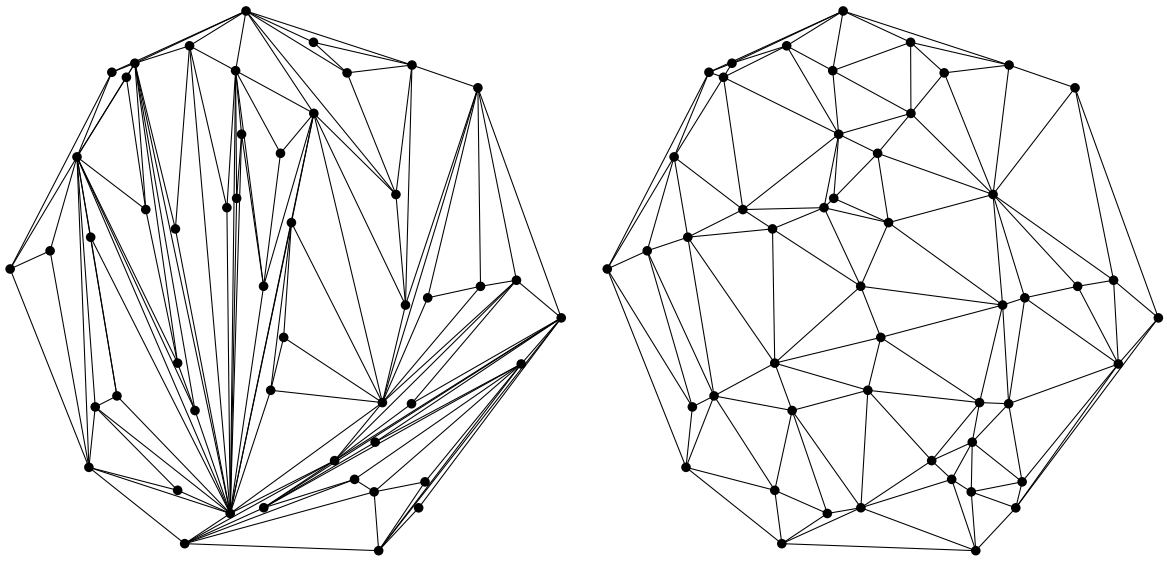
(a) Getting started. **(b)** Adding a point.

Figure 6.2: *Constructing the scan triangulation of* P*.*

Then we add $p_{m+1}, \ldots, p_n$. When adding p_i, for $i > m$, we connect p_i with all vertices of $C_{i-1} := \operatorname{conv}(\{p_1, \ldots, p_{i-1}\})$ that it "sees", that is, every vertex v of C_{i-1} for which $\overline{p_i v} \cap C_{i-1} = \{v\}$. In particular, among these vertices are the two points of tangency from p_i to C_{i-1}, which shows that we always add triangles (Figure 6.2b) whose union after each step covers C_i. □

The triangulation that is constructed in Proposition 6.2 is called a *scan triangulation*. Such a triangulation (Figure 6.3a (left) shows a larger example) is usually "ugly", though,

since it tends to have many long and skinny triangles. This is not just an aesthetic deficit. Having long and skinny triangles means that the vertices of a triangle tend to be spread out far from each other. You can probably imagine that such a behavior is undesirable, for instance, in the context of interpolation. In contrast, the *Delaunay triangulation* of the same point set (Figure 6.3b) looks much nicer, and we will discuss in the next section how to get this triangulation.



(a) Scan triangulation. **(b)** Delaunay triangulation.

Figure 6.3: *Two triangulations of the same set of* 50 *points.*

Exercise 6.3 *Describe an* $O(n \log n)$ *time algorithm to construct a scan triangulation for a set of* n *points in* $\mathbb{R}^2$*.*

On another note, if you look closely into the SLR-algorithm to compute planar convex hull that was discussed in Chapter 3, then you will realize that we also could have used this algorithm in the proof of Proposition 6.2. Whenever a point is discarded during SLR, a triangle is added to the polygon that eventually becomes the convex hull.

In view of the preceding chapter, we may regard a triangulation as a plane graph: the vertices are the points in P and there is an edge between two points $p \neq q$, if and only if there is a triangle with vertices p and q. Therefore we can use Euler's formula to determine the number of edges in a triangulation.

Lemma 6.4 *Any triangulation of a set* $P \subset \mathbb{R}^2$ *of* n *points has exactly* $3n-h-3$ *edges, where* h *is the number of vertices on* $\mathrm{conv}(P)$*.*

Proof. Consider a triangulation T of P and denote by E the set of edges and by F the set of faces of T. We count the number of edge-face incidences in two ways. Denote $\mathcal{I} = \{(e, f) \in E \times F : e \subset \partial f\}$.

On the one hand, every edge is incident to exactly two faces and therefore $|\mathcal{J}| = 2|E|$. On the other hand, every bounded face of T is a triangle and the unbounded face has h edges on its boundary. Therefore, $|\mathcal{J}| = 3(|F| - 1) + h$.

Together we obtain $3|F| = 2|E| - h + 3$. Using Euler's formula $(3n - 3|E| + 3|F| = 6)$ we conclude that $3n - |E| - h + 3 = 6$ and so $|E| = 3n - h - 3$. □

In graph theory, the term "triangulation" is sometimes used as a synonym for "maximal planar". But geometric triangulations are different, they are maximal planar in the sense that no straight-line edge can be added without sacrificing planarity.

Corollary 6.5 *A triangulation of a set* $P \subset \mathbb{R}^2$ *of* n *points is maximal planar, if and only if* $\text{conv}(P)$ *is a triangle.*

Proof. Combine Lemma 5.1 and Lemma 6.4. □

Exercise 6.6 *Find for every* $n \geqslant 3$ *a simple polygon* P *with* n *vertices such that* P *has exactly one triangulation.* P *should be in general position, meaning that no three vertices are collinear.*

Exercise 6.7 *Show that every set of* $n \geqslant 5$ *points in general position (no three points are collinear) has at least two different triangulations.*
Hint: Show first that every set of five points in general position contains a convex 4-hole, that is, a subset of four points that span a convex quadrilateral that does not contain the fifth point.

6.1 The Empty Circle Property

We will now move on to study the ominous and supposedly nice Delaunay triangulations mentioned above. They are defined in terms of an empty circumcircle property for triangles. The *circumcircle* of a triangle is the unique circle passing through the three vertices of the triangle, see Figure 6.4.

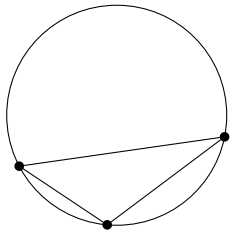


Figure 6.4: *Circumcircle of a triangle.*

Definition 6.8 *A triangulation of a finite point set* $P \subset \mathbb{R}^2$ *is called a* **Delaunay triangulation**, *if the circumcircle of every triangle is empty, that is, there is no point from* P *in its interior.*

Consider the example depicted in Figure 6.5. It shows a Delaunay triangulation of a set of six points: The circumcircles of all five triangles are empty (we also say that the triangles satisfy the empty circle property). The dashed circle is not empty, but that is fine, since it is not a circumcircle of any triangle.

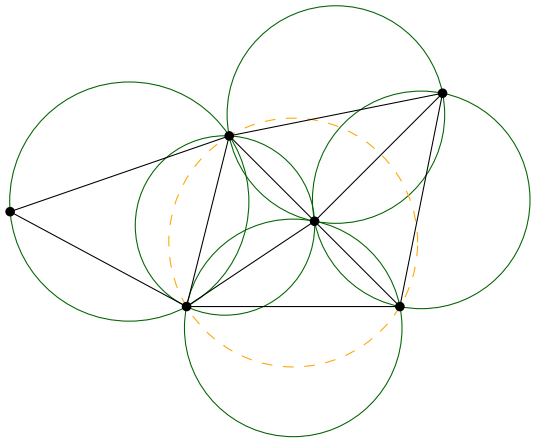
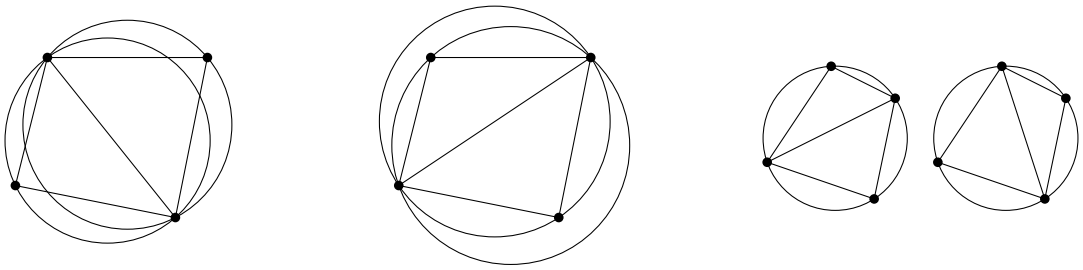


Figure 6.5: *All triangles satisfy the empty circle property.*

It is instructive to look at the case of four points in convex position. Obviously, there are two possible triangulations, but in general, only one of them will be Delaunay, see Figure 6.6a and 6.6b. If the four points are on a common circle, though, this circle is empty; at the same time it is the circumcircle of *all* possible triangles; therefore, both triangulations of the point set are Delaunay, see Figure 6.6c.



(a) Delaunay triangulation. **(b)** Non-Delaunay triangulation. **(c)** Two Delaunay triangulations.

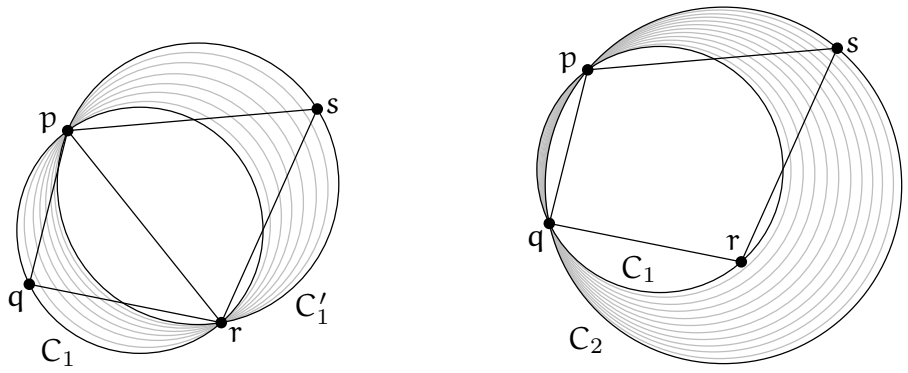
Figure 6.6: *Triangulations of four points in convex position.*

Proposition 6.9 *Given a set* $P \subset \mathbb{R}^2$ *of four points that are in convex position but not cocircular. Then* P *has exactly one Delaunay triangulation.*

Proof. Consider a convex polygon $P = pqrs$. There are two triangulation of P: a triangulation $\mathcal{T}_1$ using the edge pr and a triangulation $\mathcal{T}_2$ using the edge qs.

Consider the family $\mathcal{C}_1$ of circles through pr, which contains the circumcircles $C_1 = pqr$ and $C_1' = rsp$ of the triangles in $\mathcal{T}_1$. By assumption s is not on C_1. If s is outside of

C_1, then q is outside of C_1': Consider the process of continuously moving from C_1 to C_1' in $\mathcal{C}_1$ (Figure 6.7a); the point q is "left behind" immediately when going beyond C_1 and only the final circle C_1' "grabs" the point s.



(a) Going from C_1 to C_1' in $\mathcal{C}_1$. **(b)** Going from C_1 to C_2 in $\mathcal{C}_2$.

Figure 6.7: *Circumcircles and containment for triangulations of four points.*

Similarly, consider the family $\mathcal{C}_2$ of circles through pq, which contains the circumcircles $C_1 = pqr$ and $C_2 = spq$, the latter belonging to a triangle in $\mathcal{T}_2$. As s is outside of C_1, it follows that r is inside C_2: Consider the process of continuously moving from C_1 to C_2 in $\mathcal{C}_2$ (Figure 6.7b); the point r is on C_1 and remains within the circle all the way up to C_2. This shows that $\mathcal{T}_1$ is Delaunay, whereas $\mathcal{T}_2$ is not.

The case that s is located inside C_1 is symmetric: just cyclically shift the roles of $pqrs$ to $qrsp$. □

6.2 The Lawson Flip algorithm

It is not clear yet that every point set actually has a Delaunay triangulation (given that not all points are on a common line). In this and the next two sections, we will prove that this is the case. The proof is algorithmic. Here is the *Lawson flip algorithm* for a set P of n points.

1. Compute some triangulation of P (for example, the scan triangulation).
2. While there exists a subtriangulation of four points in convex position that is not Delaunay (like in Figure 6.6b), replace this subtriangulation by the other triangulation of the four points (Figure 6.6a).

We call the replacement operation in the second step a *(Lawson) flip*.

Theorem 6.10 *Let $P \subseteq \mathbb{R}^2$ be a set of n points, equipped with some triangulation $\mathcal{T}$. The Lawson flip algorithm terminates after at most $\binom{n}{2} = O(n^2)$ flips, and the resulting triangulation $\mathcal{D}$ is a Delaunay triangulation of P.*

We will prove Theorem 6.10 in two steps: First we show that the program described above always terminates and, therefore, is an algorithm, indeed (Section 6.3). Then we show that the algorithm does what it claims to do, namely the result is a Delaunay triangulation (Section 6.4).

6.3 Termination of the Lawson Flip Algorithm: The Lifting Map

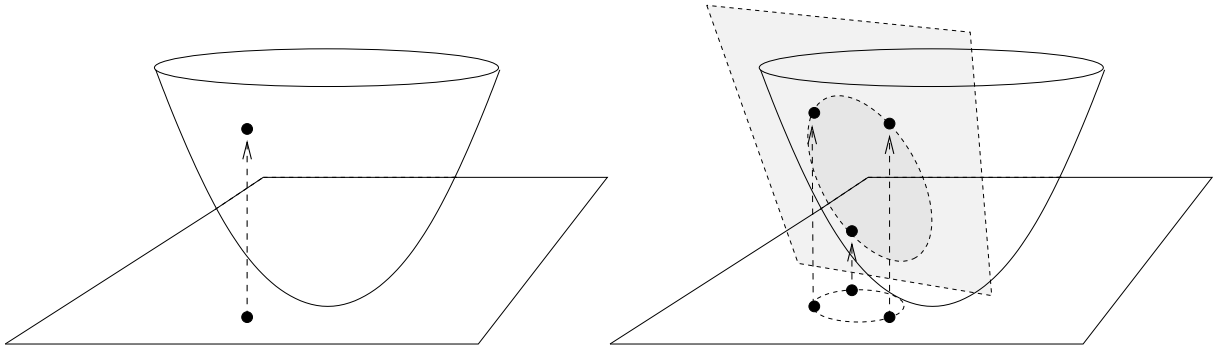
In order to prove Theorem 6.10, we invoke the (parabolic) *lifting map*. This is the following: given a point $p = (x, y) \in \mathbb{R}^2$, its *lifting* $\ell(p)$ is the point

$$\ell(p) = (x, y, x^2 + y^2) \in \mathbb{R}^3.$$

Geometrically, ℓ "lifts" the point vertically up until it lies on the *unit paraboloid*

$$\{(x, y, z) \mid z = x^2 + y^2\} \subseteq \mathbb{R}^3,$$

see Figure 6.8a.



(a) The lifting map.

(b) Points on/inside/outside a circle are lifted to points on/below/above a plane.

Figure 6.8: *The lifting map: circles map to planes.*

The following important property of the lifting map is illustrated in Figure 6.8b.

Lemma 6.11 *Let $C \subseteq \mathbb{R}^2$ be a circle of positive radius. The "lifted circle" $\ell(C) = \{\ell(p) \mid p \in C\}$ is contained in a unique plane $h_C \subseteq \mathbb{R}^3$. Moreover, a point $p \in \mathbb{R}^2$ is strictly inside (outside, respectively) of C if and only if the lifted point $\ell(p)$ is strictly below (above, respectively) h_C.*

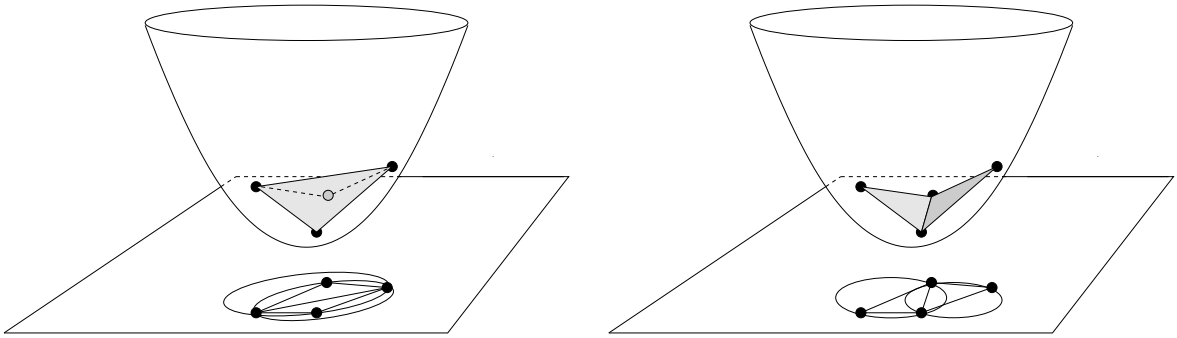
Exercise 6.12 *Prove Lemma 6.11.*

Using the lifting map, we can now prove Theorem 6.10. Let us fix the point set P for this and the next section. First, we need to argue that the algorithm indeed terminates (if you think about it a little, this is not obvious). So let us interpret a flip operation in

the lifted picture. The flip involves four points in convex position in $\mathbb{R}^2$, and their lifted images form a tetrahedron in $\mathbb{R}^3$ (think about why this tetrahedron cannot be "flat").

The tetrahedron is made up of four triangles; when you look at it from the top, you see two of the triangles, and when you look from the bottom, you see the other two. In fact, what you see from the top and the bottom are the lifted images of the two possible triangulations of the four-point set in $\mathbb{R}^2$ that is involved in the flip.

Here is the crucial fact that follows from Lemma 6.11: The two top triangles come from the non-Delaunay triangulation before the flip, see Figure 6.9a. The reason is that both top triangles have the respective fourth point below them, meaning that in $\mathbb{R}^2$, the circumcircles of these triangles contain the respective fourth point—the empty circle property is violated. In contrast, the bottom two triangles come from the Delaunay triangulation of the four points: they both have the respective fourth point above them, meaning that in $\mathbb{R}^2$, the circumcircles of the triangles do not contain the respective fourth point, see Figure 6.9b.



(a) Before the flip: the top two triangles of the tetrahedron and the corresponding non-Delaunay triangulation in the plane.

(b) After the flip: the bottom two triangles of the tetrahedron and the corresponding Delaunay triangulation in the plane.

Figure 6.9: *Lawson flip: the height of the surface of lifted triangles decreases.*

In the lifted picture, a Lawson flip can therefore be interpreted as an operation that replaces the top two triangles of a tetrahedron by the bottom two ones. If we consider the lifted image of the current triangulation, we therefore have a surface in $\mathbb{R}^3$ whose pointwise height can only decrease through Lawson flips. In particular, once an edge has been flipped, this edge will be strictly above the resulting surface and can therefore never be flipped a second time. Since n points can span at most $\binom{n}{2}$ edges, the bound on the number of flips follows.

6.4 Correctness of the Lawson Flip Algorithm

It remains to show that the triangulation of P that we get upon termination of the Lawson flip algorithm is indeed a Delaunay triangulation. Here is a first observation telling us that the triangulation is "locally Delaunay".

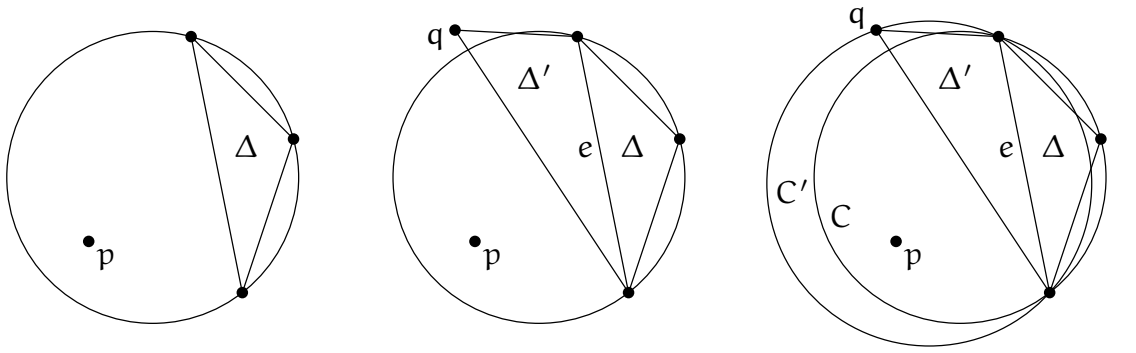
Observation 6.13 *Let Δ, Δ' be two adjacent triangles in the triangulation $\mathcal{D}$ that results from the Lawson flip algorithm. Then the circumcircle of Δ does not have any vertex of Δ' in its interior, and vice versa.*

If the two triangles together form a convex quadrilateral, this follows from the fact that the Lawson flip algorithm did not flip the common edge of Δ and Δ'. If the four vertices are not in convex position, this is basic geometry: given a triangle Δ, its circumcircle C can only contains points of $C \setminus \Delta$ that form a convex quadrilateral with the vertices of Δ.

Now we show that the triangulation is also "globally Delaunay".

Proposition 6.14 *The triangulation $\mathcal{D}$ that results from the Lawson flip algorithm is a Delaunay triangulation.*

Proof. Suppose for contradiction that there is some triangle $\Delta \in \mathcal{D}$ and some point $p \in P$ strictly inside the circumcircle C of Δ. Among all such pairs (Δ, p), we choose one for which we the distance of p to Δ is minimal. Note that this distance is positive since $\mathcal{D}$ is a triangulation of P. The situation is as depicted in Figure 6.10a.



(a) A point p inside the circumcircle C of a triangle Δ.

(b) The edge e of Δ closest to p and the second triangle Δ' incident to e.

(c) The circumcircle C' of Δ' also contains p, and p is closer to Δ' than to Δ.

Figure 6.10: *Correctness of the Lawson flip algorithm.*

Now consider the edge e of Δ that is facing p. There must be another triangle Δ' in $\mathcal{D}$ that is incident to the edge e. By the local Delaunay property of $\mathcal{D}$, the third vertex q of Δ' is on or outside of C, see Figure 6.10b. But then the circumcircle C' of Δ' contains the whole portion of C on p's side of e, hence it also contains p; moreover, p is closer to Δ' than to Δ (Figure 6.10c). But this is a contradiction to our choice of Δ and p. Hence there was no (Δ, p), and $\mathcal{D}$ is a Delaunay triangulation. □

Exercise 6.15 *The Euclidean minimum spanning tree (EMST) of a finite point set $P \subset \mathbb{R}^2$ is a spanning tree for which the sum of the edge lengths is minimum (among all spanning trees of P). Show:*

a) Every EMST of P *is a plane graph.*

b) Every EMST of P *contains a closest pair, i.e., an edge between two points* $p, q \in P$ *that have minimum distance to each other among all point pairs in* $\binom{P}{2}$.

c) Every Delaunay Triangulation of P *contains an EMST of* P.

6.5 The Delaunay Graph

Despite the fact that a point set may have more than one Delaunay triangulation, there are certain edges that are present in every Delaunay triangulation, for instance, the edges of the convex hull.

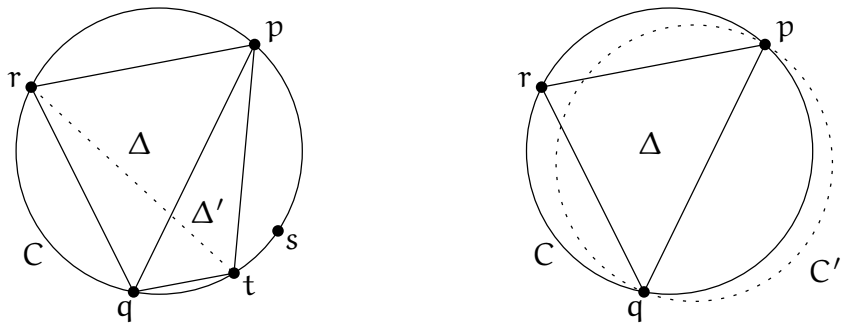
Definition 6.16 *The* **Delaunay graph** *of* $P \subseteq \mathbb{R}^2$ *consists of all line segments* $\overline{pq}$*, for* $p, q \in P$*, that are contained in every Delaunay triangulation of* P.

The following characterizes the edges of the Delaunay graph.

Lemma 6.17 *The segment* $\overline{pq}$*, for* $p, q \in P$*, is in the Delaunay graph of* P *if and only if there exists a circle through* p *and* q *that has* p *and* q *on its boundary and all other points of* P *are strictly outside.*

Proof. "$\Rightarrow$": Let pq be an edge in the Delaunay graph of P, and let $\mathcal{D}$ be a Delaunay triangulation of P. Then there exists a triangle $\Delta = pqr$ in $\mathcal{D}$, whose circumcircle C does not contain any point from P in its interior.

If there is a point s on ∂C such that $\overline{rs}$ intersects $\overline{pq}$, then let $\Delta' = pqt$ denote the other ($\neq \Delta$) triangle in $\mathcal{D}$ that is incident to pq (Figure 6.11a). Flipping the edge pq to rt yields another Delaunay triangulation of P that does not contain the edge pq, in contradiction to pq being an edge in the Delaunay graph of P. Therefore, there is no such point s.



(a) Another point $s \in \partial C$. **(b)** Moving C away from s.

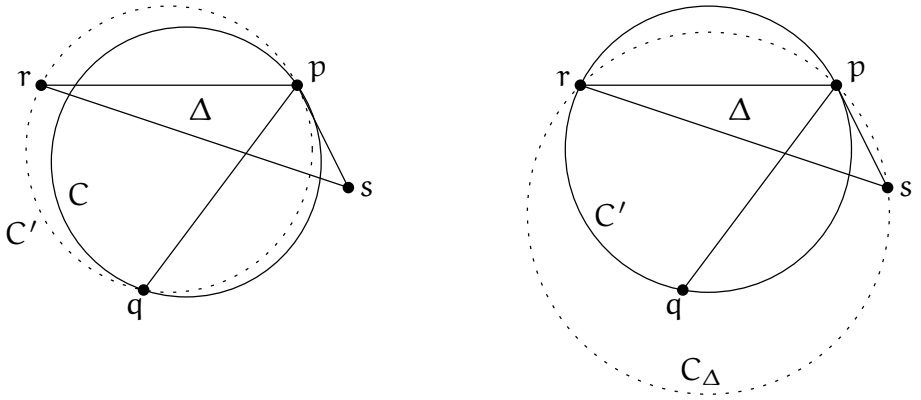
Figure 6.11: *Characterization of edges in the Delaunay graph (I).*

Otherwise we can slightly change the circle C by moving away from r while keeping p and q fix. As P is a finite point set, we can do such a modification without catching another point from P with the circle. In this way we obtain a circle C' through p and q such that all other points from P are strictly outside C' (Figure 6.12b).

"$\Leftarrow$": Let $\mathcal{D}$ be a Delaunay triangulation of P. If $\overline{pq}$ is not an edge of $\mathcal{D}$, there must be another edge of $\mathcal{D}$ that crosses $\overline{pq}$ (otherwise, we could add $\overline{pq}$ to $\mathcal{D}$ and still have a planar graph, a contradiction to $\mathcal{D}$ being a triangulation of P). Let rs denote the first edge of $\mathcal{D}$ that intersects the directed line segment $\overline{pq}$.

Consider the triangle Δ of $\mathcal{D}$ that is incident to rs on the side that faces p (given that $\overline{rs}$ intersects $\overline{pq}$ this is a well defined direction). By the choice of rs neither of the other two edges of Δ intersects $\overline{pq}$, and $p \notin \Delta^\circ$ because Δ is part of a triangulation of P. The only remaining option is that p is a vertex of $\Delta = prs$. As Δ is part of a Delaunay triangulation, its circumcircle C_Δ is empty (i.e., ${C_\Delta}^\circ \cap P = \emptyset$).

Consider now a circle C through p and q, which exists by assumption. Fixing p and q, expand C towards r to eventually obtain the circle C' through p, q, and r (Figure 6.12a). Recall that r and s are on different sides of the line through p and q. Therefore, s lies strictly outside of C'. Next fix p and r and expand C' towards s to eventually obtain the circle C_Δ through p, r, and s (Figure 6.12b). Recall that s and q are on the same side of the line through p and r. Therefore, $q \in C_\Delta$, which is in contradiction to C_Δ being empty. It follows that there is no Delaunay triangulation of P that does not contain the edge pq. □



(a) Expanding C towards r. **(b)** Expanding C' towards s.

Figure 6.12: *Characterization of edges in the Delaunay graph (II).*

The Delaunay graph is useful to prove uniqueness of the Delaunay triangulation in case of general position.

Corollary 6.18 *Let $P \subset \mathbb{R}^2$ be a finite set of points in general position, that is, no four points of P are cocircular. Then P has a unique Delaunay triangulation.* □

6.6 Every Delaunay Triangulation Maximizes the Smallest Angle

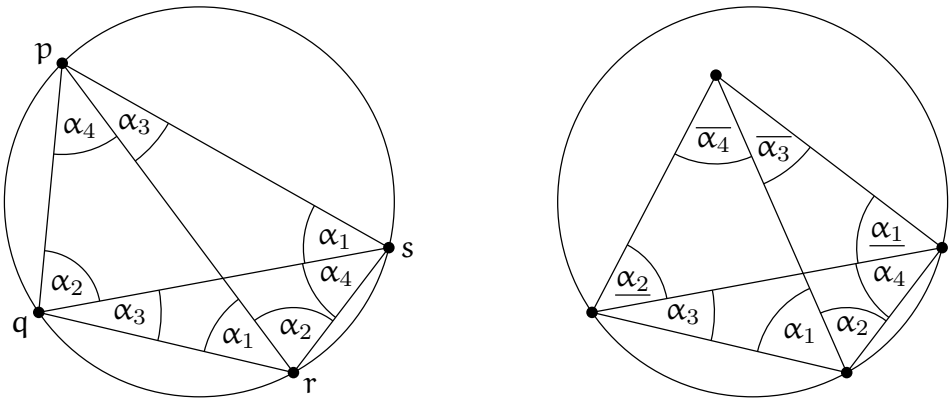
Why are we actually interested in Delaunay triangulations? After all, having empty circumcircles is not a goal in itself. But it turns out that Delaunay triangulations satisfy a number of interesting properties. Here we show just one of them.

Recall that when we compared a scan triangulation with a Delaunay triangulation of the same point set in Figure 6.3, we claimed that the scan triangulation is "ugly" because it contains many long and skinny triangles. The triangles of the Delaunay triangulation, at least in this example, look much nicer, that is, much closer to an equilateral triangle. One way to quantify this "niceness" is to look at the angles that appear in a triangulation: If all angles are large, then all triangles are reasonably close to an equilateral triangle. Indeed, we will show that Delaunay triangulations maximize the smallest angle among all triangulations of a given point set. Note that this does not imply that there are no long and skinny triangles in a Delaunay triangulation. But if there is a long and skinny triangle in a Delaunay triangulation, then there is an at least as long and skinny triangle in *every* triangulation of the point set.

Given a triangulation $\mathcal{T}$ of P, consider the lexicographically sorted sequence $A(\mathcal{T}) = (\alpha_1, \alpha_2, \ldots, \alpha_{3m})$ of interior angles, where m is the number of triangles (we have already remarked earlier that m is a function of P only and does not depend on $\mathcal{T}$). Being sorted lexicographically means that $\alpha_1 \leqslant \alpha_2 \leqslant \cdots \leqslant \alpha_{3m}$. Let $\mathcal{T}, \mathcal{T}'$ be two triangulations of P. We say that $A(\mathcal{T}) < A(\mathcal{T}')$ if there exists some i for which $\alpha_i < \alpha'_i$ and $\alpha_j = \alpha'_j$, for all $j < i$.

Theorem 6.19 *Let $P \subseteq \mathbb{R}^2$ be a finite set of points in general position (not all collinear and no four cocircular). Let $\mathcal{D}^*$ be the unique Delaunay triangulation of P, and let $\mathcal{T}$ be any triangulation of P. Then $A(\mathcal{T}) \leqslant A(\mathcal{D}^*)$.*

In particular, $\mathcal{D}^*$ maximizes the smallest angle among all triangulations of P.



(a) Four cocircular points and the induced eight angles.

(b) The situation before a flip.

Figure 6.13: *Angle-optimality of Delaunay triangulations.*

Proof. We know that $\mathcal{T}$ can be transformed into $\mathcal{D}^*$ through the Lawson flip algorithm, and we are done if we can show that each such flip lexicographically increases the sorted angle sequence. A flip replaces six interior angles by six other interior angles, and we will actually show that the smallest of the six angles *strictly* increases under the flip. This implies that the whole angle sequence increases lexicographically.

Let us first look at the situation of four cocircular points, see Figure 6.13a. In this situation, the *inscribed angle theorem* (a generalization of Thales' Theorem, stated below as Theorem 6.20) tells us that the eight depicted angles come in four equal pairs. For instance, the angles labeled α_1 at s and r are angles on the same side of the chord pq of the circle.

In Figure 6.13b, we have the situation in which we perform a Lawson flip (replacing the solid with the dashed diagonal). By the symbol $\underline{\alpha}$ ($\overline{\alpha}$, respectively) we denote an angle strictly smaller (larger, respectively) than α. Here are the six angles before the flip:

$$\alpha_1 + \alpha_2, \quad \alpha_3, \quad \alpha_4, \quad \underline{\alpha_1}, \quad \underline{\alpha_2}, \quad \overline{\alpha_3} + \overline{\alpha_4}.$$

After the flip, we have

$$\alpha_1, \quad \alpha_2, \quad \overline{\alpha_3}, \quad \overline{\alpha_4}, \quad \underline{\alpha_1} + \alpha_4, \quad \underline{\alpha_2} + \alpha_3.$$

Now, for *every* angle after the flip there is at least one smaller angle before the flip:

$$\begin{aligned}
\alpha_1 &> \underline{\alpha_1},\\
\alpha_2 &> \underline{\alpha_2},\\
\overline{\alpha_3} &> \alpha_3,\\
\overline{\alpha_4} &> \alpha_4,\\
\underline{\alpha_1} + \alpha_4 &> \alpha_4,\\
\underline{\alpha_2} + \alpha_3 &> \alpha_3.
\end{aligned}$$

It follows that the smallest angle strictly increases. □

Theorem 6.20 (Inscribed Angle Theorem) *Let C be a circle with center c and positive radius and $p, q \in C$. Then the angle $\angle prq \bmod \pi = \frac{1}{2}\angle pcq$ is the same, for all $r \in C$.*

Proof. Without loss of generality we may assume that c is located to the left of or on the oriented line pq.

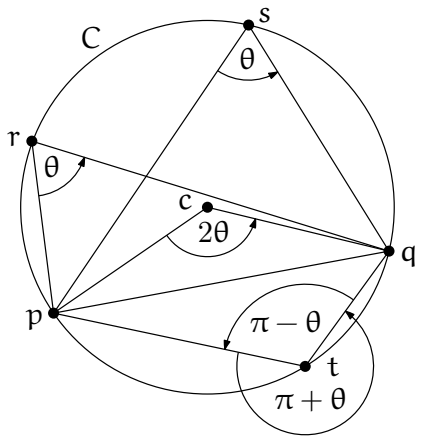
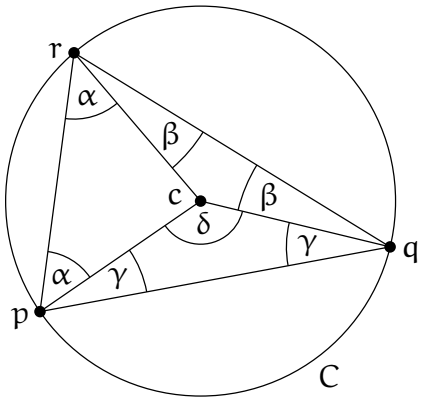


Figure 6.14: *The Inscribed Angle Theorem with* $\theta := \angle \mathrm{prq}$.

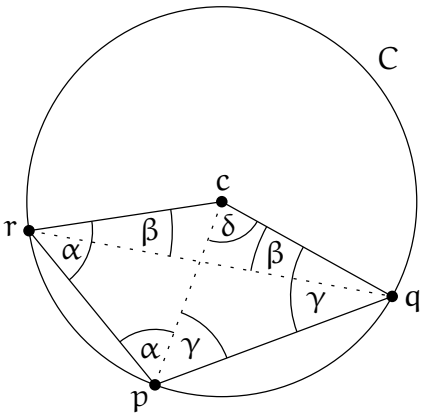
Consider first the case that the triangle $\Delta = \mathrm{pqr}$ contains c. Then Δ can be partitioned into three triangles: pcr, qcr, and cpq. All three triangles are isosceles, because two sides of each form the radius of C. Denote $\alpha = \angle \mathrm{prc}$, $\beta = \angle \mathrm{crq}$, $\gamma = \angle \mathrm{cpq}$, and $\delta = \angle \mathrm{pcq}$ (see the figure shown to the right). The angles we are interested in are $\theta = \angle \mathrm{prq} = \alpha + \beta$ and δ, for which we have to show that $\delta = 2\theta$.

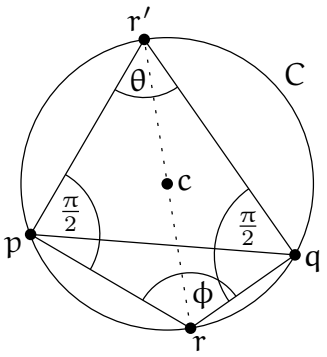
Indeed, the angle sum in Δ is $\pi = 2(\alpha + \beta + \gamma)$ and the angle sum in the triangle cpq is $\pi = \delta + 2\gamma$. Combining both yields $\delta = 2(\alpha + \beta) = 2\theta$.



Next suppose that pqcr are in convex position and r is to the left of or on the oriented line pq. Without loss of generality let r be to the left of or on the oriented line qc. (The case that r lies to the right of or on the oriented line pc is symmetric.) Define α, β, γ, δ as above and observe that $\theta = \alpha - \beta$. Again have to show that $\delta = 2\theta$.

The angle sum in the triangle cpq is $\pi = \delta + 2\gamma$ and the angle sum in the triangle rpq is $\pi = (\alpha - \beta) + \alpha + \gamma + (\gamma - \beta) = 2(\alpha + \gamma - \beta)$. Combining both yields $\delta = \pi - 2\gamma = 2(\alpha - \beta) = 2\theta$.





It remains to consider the case that r is to the right of the oriented line pq.

Consider the point r' that is antipodal to r on C, and the quadrilateral $Q = prqr'$. We are interested in the angle ϕ of Q at r. By Thales' Theorem the inner angles of Q at p and q are both $\pi/2$. Hence the angle sum of Q is $2\pi = \theta + \phi + 2\pi/2$ and so $\phi = \pi - \theta$.

□

What happens in the case where the Delaunay triangulation is not unique? The following still holds.

Theorem 6.21 *Let $P \subseteq \mathbb{R}^2$ be a finite set of points, not all on a line. Every Delaunay triangulation $\mathcal{D}$ of P maximizes the smallest angle among all triangulations $\mathcal{T}$ of P.*

Proof. Let $\mathcal{D}$ be some Delaunay triangulation of P. We infinitesimally perturb the points in P such that no four are on a common circle anymore. Then the Delaunay triangulation becomes unique (Corollary 6.18). Starting from $\mathcal{D}$, we keep applying Lawson flips until we reach the unique Delaunay triangulation $\mathcal{D}^*$ of the perturbed point set. Now we examine this sequence of flips on the original *unperturbed* point set. All these flips must involve four cocircular points (only in the cocircular case, an infinitesimal perturbation can change "good" edges into "bad" edges that still need to be flipped). But as Figure 6.13 (a) easily implies, such a "degenerate" flip does not change the smallest of the six involved angles. It follows that $\mathcal{D}$ and $\mathcal{D}^*$ have the same smallest angle, and since $\mathcal{D}^*$ maximizes the smallest angle among all triangulations $\mathcal{T}$ (Theorem 6.19), so does $\mathcal{D}$. □

6.7 Constrained Triangulations

Sometimes one would like to have a Delaunay triangulation, but certain edges are already prescribed, for example, a Delaunay triangulation of a simple polygon. Of course, one cannot expect to be able to get a proper Delaunay triangulation where all triangles satisfy the empty circle property. But it is possible to obtain some triangulation that comes as close as possible to a proper Delaunay triangulation, given that we are forced to include the edges in E. Such a triangulation is called a *constrained Delaunay triangulation*, a formal definition of which follows.

Let $P \subseteq \mathbb{R}^2$ be a finite point set and $G = (P, E)$ a geometric graph with vertex set P (we consider the edges $e \in E$ as line segments). A triangulation $\mathcal{T}$ of P *respects* G if it contains all segments $e \in E$. A triangulation $\mathcal{T}$ of P that respects G is said to be a *constrained Delaunay triangulation* of P with respect to G if the following holds for every triangle Δ of $\mathcal{T}$:

The circumcircle of Δ contains only points $q \in P$ in its interior that are not visible from the interior of Δ. A point $q \in P$ is *visible* from the interior of Δ if there exists a point p in the interior of Δ such that the line segment $\overline{pq}$ does not intersect any segment $e \in E$. We can thus imagine the line segments of E as "blocking the view".

For illustration, consider the simple polygon and its constrained Delaunay triangulation shown in Figure 6.15. The circumcircle of the shaded triangle Δ contains a whole other triangle in its interior. But these points cannot be seen from Δ°, because all possible connecting line segments intersect the blocking polygon edge e of Δ.

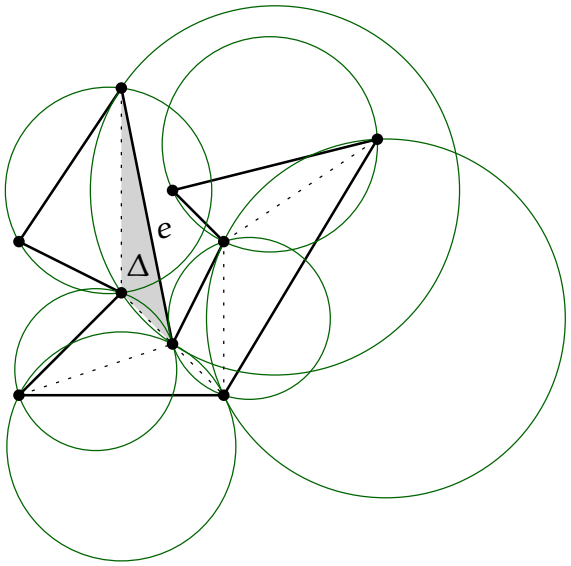


Figure 6.15: *Constrained Delaunay triangulation of a simple polygon.*

Theorem 6.22 *For every finite point set P and every plane graph $G = (P, E)$, there exists a constrained Delaunay triangulation of P with respect to G.*

Exercise 6.23 *Prove Theorem 6.22. Also describe a polynomial algorithm to construct such a triangulation.*

Questions

20. *What is a triangulation?* Provide the definition and prove a basic property: every triangulation with the same set of vertices and the same outer face has the same number of triangles.

21. *What is a triangulation of a point set?* Give a precise definition.

22. *Does every point set (not all points on a common line) have a triangulation?* You may, for example, argue with the scan triangulation.

23. *What is a Delaunay triangulation of a set of points*? Give a precise definition.

24. *What is the Delaunay graph of a point set?* Give a precise definition and a characterization.

25. *How can you prove that every set of points (not all on a common line) has a Delaunay triangulation?* You can for example sketch the Lawson flip algorithm and the Lifting Map, and use these to show the existence.

26. *When is the Delaunay triangulation of a point set unique?* Show that general position is a sufficient condition. Is it also necessary?

27. *What can you say about the "quality" of a Delaunay triangulation?* Prove that every Delaunay triangulation maximizes the smallest interior angle in the triangulation, among the set of all triangulations of the same point set.

Chapter 7

Delaunay Triangulation: Incremental Construction

In the last lecture, we have learned about the Lawson flip algorithm that computes a Delaunay triangulation of a given n-point set $P \subseteq \mathbb{R}^2$ with $O(n^2)$ Lawson flips. One can actually implement this algorithm to run in $O(n^2)$ time, and there are point sets where it may take $\Omega(n^2)$ flips.

In this lecture, we will discuss a different algorithm. The final goal is to show that this algorithm can be implemented to run in $O(n \log n)$ time; this lecture, however, is concerned only with the correctness of the algorithm. Throughout the lecture we assume that P is in general position (no 3 points on a line, no 4 points on a common circle), so that the Delaunay triangulation is unique (Corollary 6.18). There are techniques to deal with non-general position, but we don't discuss them here.

7.1 Incremental construction

The idea is to build the Delaunay triangulation of P by inserting one point after another. We always maintain the Delaunay triangulation of the point set R inserted so far, and when the next point s comes along, we simply update the triangulation to the Delaunay triangulation of $R \cup \{s\}$. Let $\mathcal{DT}(R)$ denote the Delaunay triangulation of $R \subseteq P$.

To avoid special cases, we enhance the point set P with three artificial points "far out". The convex hull of the resulting point set is a triangle; later, we can simply remove the extra points and their incident edges to obtain $\mathcal{DT}(P)$. The incremental algorithm starts off with the Delaunay triangulation of the three artificial points which consists of one big triangle enclosing all other points. (In our figures, we suppress the far-away points, since they are merely a technicality.)

Now assume that we have already built $\mathcal{DT}(R)$, and we next insert $s \in P \setminus R$. Here is the outline of the update step.

1. Find the triangle $\Delta = \Delta(p, q, r)$ of $\mathcal{DT}(R)$ that contains s, and replace it with the three triangles resulting from connecting s with all three vertices p, q, r; see Figure

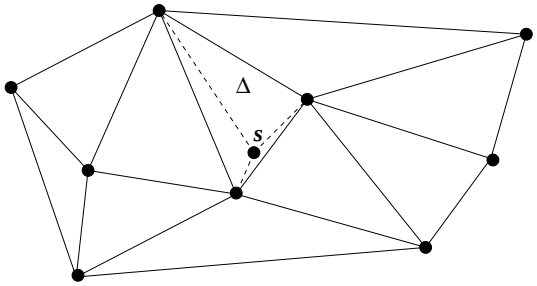


Figure 7.1: *Inserting* s *into* $\mathcal{DT}(R)$*: Step 1*

7.1. We now have a triangulation $\mathcal{T}$ of $R \cup \{s\}$.

2. Perform Lawson flips on $\mathcal{T}$ until $\mathcal{DT}(R \cup \{s\})$ is obtained; see Figure 7.2

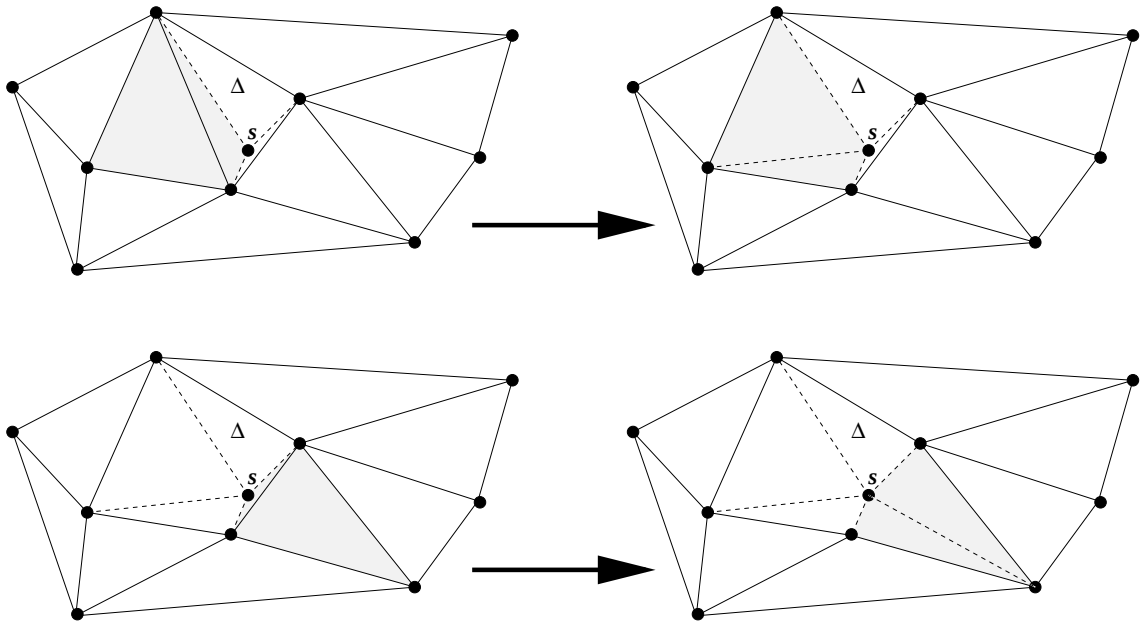


Figure 7.2: *Inserting* s *into* $\mathcal{DT}(R)$*: Step 2*

How to organize the Lawson flips. The Lawson flips can be organized quite systematically, since we always know the candidates for "bad" edges that may still have to be flipped. Initially (after step 1), only the three edges of Δ can be bad, since these are the only edges for which an incident triangle has changed (by inserting s in Step 1). Each of the three new edges is good, since the 4 vertices of its two incident triangles are not in convex position.

Now we have the following invariant (part (a) certainly holds in the first flip):

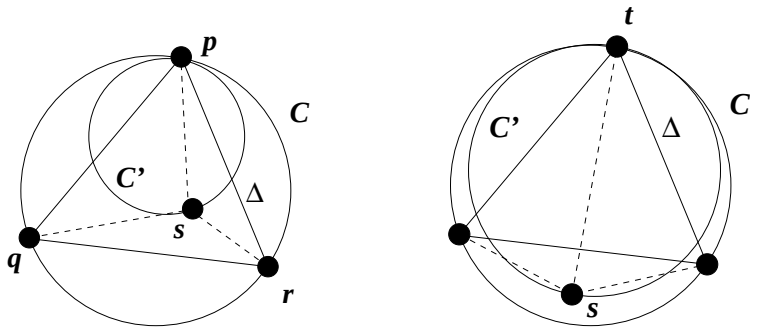
(a) In every flip, the convex quadrilateral Q in which the flip happens has exactly two edges incident to s, and the flip generates a new edge incident to s.

(b) Only the two edges of Q that are *not* incident to s can become bad through the flip.

We will prove part (b) in the next lemma. The invariant then follows since (b) entails (a) in the next flip. This means that we can maintain a queue of potentially bad edges that we process in turn. A good edge will simply be removed from the queue, and a bad edge will be flipped and replaced according to (b) with two new edges in the queue. In this way, we never flip edges incident to s; the next lemma proves that this is correct and at the same time establishes part (b) of the invariant.

Lemma 7.1 *Every edge incident to* s *that is created during the update is an edge of the Delaunay graph of* $R \cup \{s\}$ *and thus an edge that will be in* $\mathcal{DT}(R \cup \{s\})$*. It easily follows that edges incident to* s *will never become bad during the update step.*[1]

Proof. Let us consider one of the first three new edges, $\overline{sp}$, say. Since the triangle Δ has a circumcircle C strictly containing only s (Δ is in $\mathcal{DT}(R)$), we can shrink that circumcircle to a circle C' through s and p with no interior points, see Figure 7.3 (a). This proves that $\overline{sp}$ is in the Delaunay graph. If $\overline{st}$ is an edge created by a flip, a similar argument works. The flip destroys exactly one triangle Δ of $\mathcal{DT}(R)$. Its circumcircle C contains s only, and shrinking it yields an empty circle C' through s and t. Thus, $\overline{st}$ is in the Delaunay graph also in this case. □



(a) New edge $\overline{sp}$ incident to s created in Step 1

(b) New edge $\overline{st}$ incident to s created in Step 2

Figure 7.3: *Newly created edges incident to* s *are in the Delaunay graph*

[1] If such an edge was bad, it could be flipped, but then it would be "gone forever" according to the lifting map interpretation from the previous lecture.

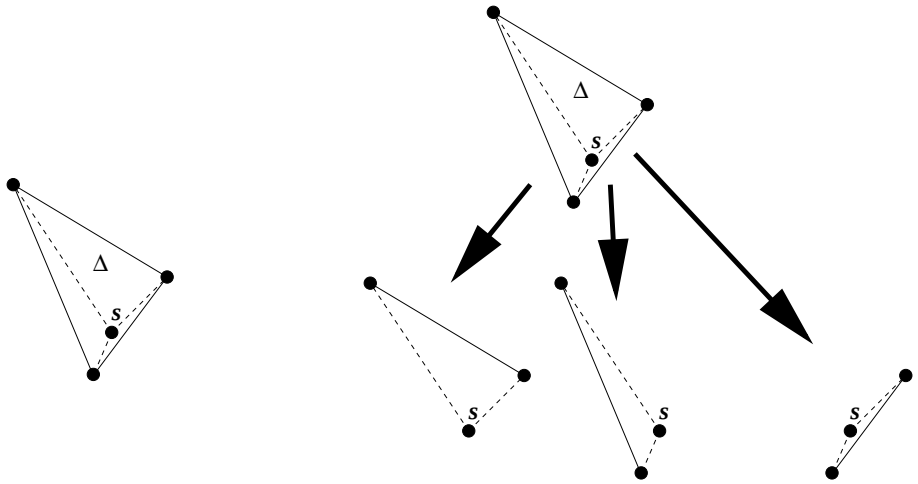


Figure 7.4: *The history graph: one triangle gets replaced by three triangles*

7.2 The History Graph

What can we say about the performance of the incremental construction? Not much yet. First of all, we did not specify how we find the triangle Δ of $\mathcal{DT}(R)$ that contains the point s to be inserted. Doing this in the obvious way (checking all triangles) is not good, since already the find steps would then amount to $O(n^2)$ work throughout the whole algorithm. Here is a smarter method, based on the *history graph*.

Definition 7.2 *Given $R \subseteq P$ (regarded as a sequence that reflects the insertion order), the history graph of R is a directed acyclic graph whose vertices are all triangles that have ever been created during the incremental construction of $\mathcal{DT}(R)$. There is a directed edge from Δ to Δ' whenever Δ has been destroyed during an insertion step, Δ' has been created during the same insertion step, and Δ overlaps with Δ' in its interior.*

It follows that the history graph contains triangles of outdegrees 3, 2 and 0. The ones of outdegree 0 are clearly the triangles of $\mathcal{DT}(R)$.

The triangles of outdegree 3 are the ones that have been destroyed during Step 1 of an insertion. For each such triangle Δ, its three outneighbors are the three new triangles that have replaced it, see Figure 7.4.

The triangles of outdegree 2 are the ones that have been destroyed during Step 2 of an insertion. For each such triangle Δ, its two outneighbors are the two new triangles created during the flip that has destroyed Δ, see Figure 7.5.

The history graph can be built during the incremental construction at asymptotically no extra cost; but it may need extra space since it keeps all triangles ever created. Given the history graph, we can search for the triangle Δ of $\mathcal{DT}(R)$ that contains s, as follows. We start from the big triangle spanned by the three far-away points; this one certainly

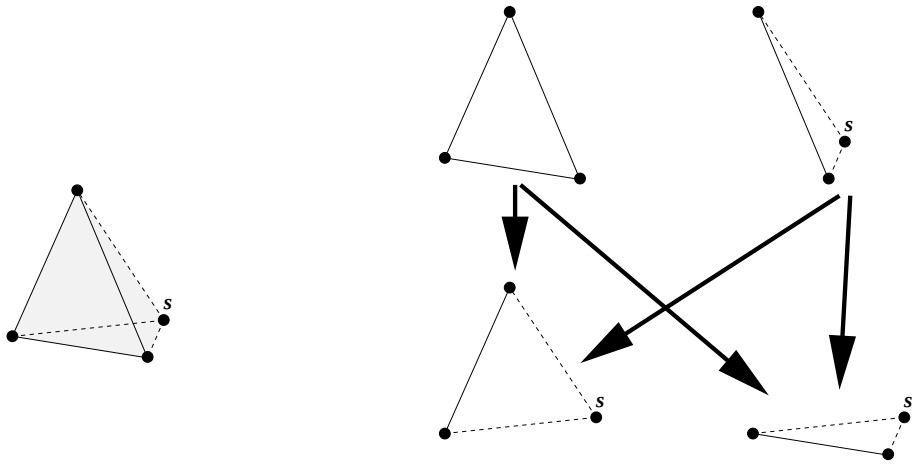


Figure 7.5: *The history graph: two triangles get replaced by two triangles*

contains s. Then we follow a directed path in the history graph. If the current triangle still has outneighbors, we find the unique outneighbor containing s and continue the search with this neighbor. If the current triangle has no outneighbors anymore, it is in $\mathcal{DT}(R)$ and contains s—we are done.

Types of triangles in the history graph. After each insertion of a point s, several triangles are created and added to the history graph. It is important to note that these triangles come in two types: Some of them are valid Delaunay triangles of $R \cup \{s\}$, and they survive to the next stage of the incremental construction. Other triangles are immediately destroyed by subsequent Lawson flips, because they are not Delaunay triangles of $R \cup \{s\}$. These "ephemeral" triangles will give us some headache (though not much) in the algorithm's analysis in the next chapter.

Note that, whenever a Lawson flip is performed, of the two triangles destroyed one of them is always a "valid" triangle from a previous iteration, and the other one is an "ephemeral" triangle that was created at this iteration. The ephemeral triangle is always the one that has s, the newly inserted point, as a vertex.

7.3 The structural change

Concerning the actual update (Steps 1 and 2), we can make the following

Observation 7.3 *Given $\mathcal{DT}(R)$ and the triangle Δ of $\mathcal{DT}(R)$ that contains s, we can build $\mathcal{DT}(R \cup \{s\})$ in time proportional to the degree of s in $\mathcal{DT}(R \cup \{s\})$, which is the number of triangles of $\mathcal{DT}(R \cup \{s\})$ containing s.*

Indeed, since every flip generates exactly one new triangle incident to s, the number of flips is the degree of s minus three. Step 1 of the update takes constant time, and

since also every flip can be implemented in constant time, the observation follows.

In the next lecture, we will show that a clever insertion order guarantees that the search paths traversed in the history graph are short, and that the structural change (the number of new triangles) is small. This will then give us the $O(n \log n)$ algorithm.

Exercise 7.4 *For a sequence of n pairwise distinct numbers $y_1, \ldots, y_n$ consider the sequence of pairs $(\min\{y_1, \ldots, y_i\}, \max\{y_1, \ldots, y_i\})_{i=0,1,\ldots,n}$ ($\min \emptyset := +\infty, \max \emptyset := -\infty$). How often do these pairs change in expectation if the sequence is permuted randomly, each permutation appearing with the same probability? Determine the expected value.*

Questions

28. *Describe the algorithm for the incremental construction of $\mathcal{DT}(P)$*: how do we find the triangle containing the point s to be inserted into $\mathcal{DT}(R)$? How do we transform $\mathcal{DT}(R)$ into $\mathcal{DT}(R \cup \{s\})$? How many steps does the latter transformation take, in terms of $\mathcal{DT}(R \cup \{s\})$?

29. *What are the two types of triangles that the history graph contains?*

Chapter 8

The Configuration Space Framework

In Section 7.1, we have discussed the incremental construction of the Delaunay triangulation of a finite point set. In this lecture, we want to analyze the runtime of this algorithm if the insertion order is chosen *uniformly at random* among all insertion orders. We will do the analysis not directly for the problem of constructing the Delaunay triangulation but in a somewhat more abstract framework, with the goal of reusing the analysis for other problems.

Throughout this lecture, we again assume general position: no three points on a line, no four on a circle.

8.1 The Delaunay triangulation — an abstract view

The incremental construction constructs and destroys triangles. In this section, we want to take a closer look at these triangles, and we want to understand exactly when a triangle is "there".

Lemma 8.1 *Given three points* $p, q, r \in R$*, the triangle* $\Delta(p, q, r)$ *with vertices* p, q, r *is a triangle of* $\mathcal{DT}(R)$ *if and only if the circumcircle of* $\Delta(p, q, r)$ *is empty of points from* R*.*

Proof. The "only if" direction follows from the definition of a Delaunay triangulation (Definition 6.8). The "if" direction is a consequence of general position and Lemma 6.17: if the circumcircle C of $\Delta(p, q, r)$ is empty of points from R, then all the three edges $\overline{pq}, \overline{qr}, \overline{pr}$ are easily seen to be in the Delaunay graph of R. C being empty also implies that the triangle $\Delta(p, q, r)$ is empty, and hence it forms a triangle of $\mathcal{DT}(R)$. □

Next we develop a somewhat more abstract view of $\mathcal{DT}(R)$.

Definition 8.2

(i) For all $p, q, r \in P$*, the triangle* $\Delta = \Delta(p, q, r)$ *is called a* configuration*. The points* p, q *and* r *are called the* defining elements *of* Δ*.*

(ii) A configuration Δ is in conflict *with a point $s \in P$ if s is strictly inside the circumcircle of Δ. In this case, the pair (Δ, s) is called a* conflict*.*

(iii) A configuration Δ is called active *w.r.t. $R \subseteq P$ if (a) the defining elements of Δ are in R, and (b) if Δ is not in conflict with any element of R.*

According to this definition and Lemma 8.1, $\mathcal{DT}(R)$ consists of exactly the configurations that are active w.r.t. R. Moreover, if we consider $\mathcal{DT}(R)$ and $\mathcal{DT}(R \cup \{s\})$ as sets of configurations, we can exactly say how these two sets differ.

There are the configurations in $\mathcal{DT}(R)$ that are not in conflict with s. These configurations are still in $\mathcal{DT}(R \cup \{s\})$. The configurations of $\mathcal{DT}(R)$ that *are* in conflict with s will be removed when going from R to $R \cup \{s\}$. Finally, $\mathcal{DT}(R \cup \{s\})$ contains some new configurations, all of which must have s in their defining set. According to Lemma 8.1, it cannot happen that we get a new configuration without s in its defining set, as such a configuration would have been present in $\mathcal{DT}(R)$ already.

8.2 Configuration Spaces

Here is the abstract framework that generalizes the previous configuration view of the Delaunay triangulation.

Definition 8.3 *Let X (the ground set) and Π (the set of configurations) be finite sets. Furthermore, let*

$$D : \Pi \to 2^X$$

be a function that assigns to every configuration Δ a set of defining elements *$D(\Delta)$. We assume that only a constant number of configurations have the same defining elements. Let*

$$K : \Pi \to 2^X$$

be a function that assigns to every configuration Δ a set of elements in conflict *with Δ (the "killer" elements). We stipulate that $D(\Delta) \cap K(\Delta) = \emptyset$ for all $\Delta \in \Pi$.*

Then the quadruple $\mathcal{S} = (X, \Pi, D, K)$ is called a configuration space*. The number*

$$d = d(\mathcal{S}) := \max_{\Delta \in \Pi} |D(\Delta)|$$

is called the dimension *of $\mathcal{S}$.*

Given $R \subseteq X$, a configuration Δ is called active *w.r.t. R if*

$$D(\Delta) \subseteq R \quad \textit{and} \quad K(\Delta) \cap R = \emptyset,$$

i.e. if all defining elements are in R but no element of R is in conflict with Δ. The set of active configurations w.r.t. R is denoted by $\mathcal{T}_{\mathcal{S}}(R)$, where we drop the subscript if the configuration space is clear from the context.

In case of the Delaunay triangulation, we set $X = P$ (the input point set). Π consists of all triangles $\Delta = \Delta(p, q, r)$ spanned by three points $p, q, r \in X \cup \{a, b, c\}$, where a, b, c are the three artificial far-away points. We set $D(\Delta) := \{p, q, r\} \cap X$. The set $K(\Delta)$ consists of all points strictly inside the circumcircle of Δ. The resulting configuration space has dimension 3, and the technical condition that only a constant number of configurations share the defining set is satisfied as well. In fact, every set of three points defines a *unique* configuration (triangle) in this case. A set of two points or one point defines three triangles (we have to add one or two artificial points which can be done in three ways). The empty set defines one triangle, the initial triangle consisting of just the three artificial points.

Furthermore, in the setting of the Delaunay triangulation, a configuration is active w.r.t. R if it is in $\mathcal{DT}(R \cup \{a, b, c\})$, i.e. we have $\mathcal{T}(R) = \mathcal{DT}(R \cup \{a, b, c\})$.

8.3 Expected structural change

Let us fix a configuration space $\mathcal{S} = (X, \Pi, D, K)$ for the remainder of this lecture. We can also interpret the incremental construction in $\mathcal{S}$. Given $R \subseteq X$ and $s \in X \setminus R$, we want to update $\mathcal{T}(R)$ to $\mathcal{T}(R \cup \{s\})$. What is the number of new configurations that arise during this step? For the case of Delaunay triangulations, this is the relevant question when we want to bound the number of Lawson flips during one update step, since this number is exactly the number of new configurations minus three.

Here is the general picture.

Definition 8.4 *For $Q \subseteq X$ and $s \in Q$, $\deg(s, Q)$ is defined as the number of configurations of $\mathcal{T}(Q)$ that have s in their defining set.*

With this, we can say that the number of new configurations in going from $\mathcal{T}(R)$ to $\mathcal{T}(R \cup \{s\})$ is precisely $\deg(s, R \cup \{s\})$, since the new configurations are by definition exactly the ones that have s in their defining set.

Now the random insertion order comes in for the first time: what is

$$E(\deg(s, R \cup \{s\})),$$

averaged over all insertion orders? In such a random insertion order, R is a random r-element subset of X (when we are about to insert the $(r+1)$-st element), and s is a random element of $X \setminus R$. Let $\mathcal{T}_r$ be the "random variable" for the set of active configurations after r insertion steps.

It seems hard to average over all R, but there is a trick: we make a movie of the randomized incremental construction, and then we watch the movie backwards. What we see is elements of X being deleted one after another, again in random order. This is due to the fact that the reverse of a random order is also random. At the point where the $(r+1)$-st element is being deleted, it is going to be a *random* element s of the currently

present $(r+1)$-element subset Q. For fixed Q, the expected degree of s is simply the average degree of an element in Q which is

$$\frac{1}{r+1}\sum_{s\in Q}\deg(s,Q) \leqslant \frac{d}{r+1}|\mathcal{T}(Q)|,$$

since the sum counts every configuration of $\mathcal{T}(Q)$ at most d times. Since Q is a random $(r+1)$-element subset, we get

$$E(\deg(s,R\cup\{s\})) \leqslant \frac{d}{r+1}t_{r+1},$$

where t_{r+1} is defined as the expected number of active configurations w.r.t. a random $(r+1)$-element set.

Here is a more formal derivation that does not use the backwards movie view. It exploits the bijection

$$(R,s) \mapsto (\underbrace{R\cup\{s\}}_{Q},s)$$

between pairs (R,s) with $|R|=r$ and $s\notin R$ and pairs (Q,s) with $|Q|=r+1$ and $s\in Q$. Let $n=|X|$.

$$\begin{aligned}
E(\deg(s,R\cup\{s\})) &= \frac{1}{\binom{n}{r}}\sum_{R\subseteq X,|R|=r}\frac{1}{n-r}\sum_{s\in X\setminus R}\deg(s,R\cup\{s\})\\
&= \frac{1}{\binom{n}{r}}\sum_{Q\subseteq X,|Q|=r+1}\frac{1}{n-r}\sum_{s\in Q}\deg(s,Q)\\
&= \frac{1}{\binom{n}{r+1}}\sum_{Q\subseteq X,|Q|=r+1}\frac{\binom{n}{r+1}}{\binom{n}{r}}\frac{1}{n-r}\sum_{s\in Q}\deg(s,Q)\\
&= \frac{1}{\binom{n}{r+1}}\sum_{Q\subseteq X,|Q|=r+1}\frac{1}{r+1}\sum_{s\in Q}\deg(s,Q)\\
&\leqslant \frac{1}{\binom{n}{r+1}}\sum_{Q\subseteq X,|Q|=r+1}\frac{d}{r+1}|\mathcal{T}(Q)|\\
&= \frac{d}{r+1}t_{r+1}.
\end{aligned}$$

Thus, the expected number of new configurations in going from $\mathcal{T}_r$ to $\mathcal{T}_{r+1}$ is bounded by

$$\frac{d}{r+1}t_{r+1},$$

where t_{r+1} is the expected size of $\mathcal{T}_{r+1}$.

What do we get for Delaunay triangulations? We have $d = 3$ and $t_{r+1} \leqslant 2(r+4) - 4$ (the maximum number of triangles in a triangulation of $r+4$ points). Hence,

$$E(\deg(s, R \cup \{s\})) \leqslant \frac{6r+12}{r+1} \approx 6.$$

This means that on average, ≈ 3 Lawson flips are done to update $\mathcal{DT}_r$ (the Delaunay triangulation after r insertion steps) to $\mathcal{DT}_{r+1}$. Over the whole algorithm, the expected update cost is thus $O(n)$.

8.4 Bounding location costs by conflict counting

Before we can even update $\mathcal{DT}_r$ to $\mathcal{DT}_{r+1}$ during the incremental construction of the Delaunay triangulation, we need to locate the new point s in $\mathcal{DT}_r$, meaning that we need to find the triangle that contains s. We have done this with the history graph: During the insertion of s we "visit" a sequence of triangles from the history graph, each of which contains s and was created at some previous iteration $k < r$.

However, some of these visited triangles are "ephemeral" triangles (recall the discussion at the end of Section 7.2), and they present a problem to the generic analysis we want to perform. Therefore, we will do a charging scheme, so that all triangles charged are valid Delaunay triangles.

The charging scheme is as follows: If the visited triangle Δ is a valid Delaunay triangle (from some previous iteration), then we simply charge the visit of Δ during the insertion of s to the triangle-point pair (Δ, s).

If, on the other hand, Δ is an "ephemeral" triangle, then Δ was destroyed, together with some neighbor Δ', by a Lawson flip into another pair Δ'', Δ'''. Note that this neighbor Δ' was a valid triangle. Thus, in this case we charge the visit of Δ during the insertion of s to the pair (Δ', s). Observe that s is contained in the circumcircle of Δ', so s is in conflict with Δ'.

This way, we have charged each visit to a triangle in the history graph to a triangle-point pair of the form (Δ, s), such that Δ is in conflict with s. Furthermore, it is easy to see that no such pair gets charged more than once.

We define the notion of a *conflict* in general:

Definition 8.5 *A* conflict *is a configuration-element pair* (Δ, s) *where* $\Delta \in \mathcal{T}_r$ *for some* r *and* $s \in K(\Delta)$.

Thus, the running time of the Delaunay algorithm is proportional to the number of conflicts. We now proceed to derive a bound on the expected number of conflicts in the generic configuration-space framework.

8.5 Expected number of conflicts

Since every configuration involved in a conflict has been created in some step r (we include step 0), the total number of conflicts is

$$\sum_{r=0}^{n} \sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} |K(\Delta)|,$$

where $\mathcal{T}_{-1} := \emptyset$. $\mathcal{T}_0$ consists of constantly many configurations only (namely those where the set of defining elements is the empty set), each of which is in conflict with at most all elements; moreover, no conflict is created in step n. Hence,

$$\sum_{r=0}^{n} \sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} |K(\Delta)| = O(n) + \sum_{r=1}^{n-1} \sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} |K(\Delta)|,$$

and we will bound the latter quantity. Let

$$K(r) := \sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} |K(\Delta)|, \quad r = 1, \ldots, n-1.$$

and $k(r) := E(K(r))$ the expected number of conflicts created in step r.

Bounding $k(r)$. We know that $\mathcal{T}_r$ arises from a random r-element set R. Fixing R, the backwards movie view tells us that $\mathcal{T}_{r-1}$ arises from $\mathcal{T}_r$ by deleting a random element s of R. Thus,

$$\begin{aligned} k(r) &= \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{1}{r} \sum_{s \in R} \sum_{\Delta \in \mathcal{T}(R) \setminus \mathcal{T}(R \setminus \{s\})} |K(\Delta)| \\ &= \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{1}{r} \sum_{s \in R} \sum_{\Delta \in \mathcal{T}(R), s \in D(\Delta)} |K(\Delta)| \\ &\leqslant \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r} \sum_{\Delta \in \mathcal{T}(R)} |K(\Delta)|, \end{aligned}$$

since in the sum over $s \in R$, every configuration is counted at most d times. Since we can rewrite

$$\sum_{\Delta \in \mathcal{T}(R)} |K(\Delta)| = \sum_{y \in X \setminus R} |\{\Delta \in \mathcal{T}(R) : y \in K(\Delta)\}|,$$

we thus have

$$k(r) \leqslant \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r} \sum_{y \in X \setminus R} |\{\Delta \in \mathcal{T}(R) : y \in K(\Delta)\}|.$$

To estimate this further, here is a simple but crucial

Lemma 8.6 *The configurations in $\mathcal{T}(R)$ that are not in conflict with y are the configurations in $\mathcal{T}(R \cup \{y\})$ that do not have y in their defining set; in formulas:*

$$|\mathcal{T}(R)| - |\{\Delta \in \mathcal{T}(R) : y \in K(\Delta)\}| = |\mathcal{T}(R \cup \{y\})| - \deg(y, R \cup \{y\}).$$

The proof is a direct consequence of the definitions: every configuration in $\mathcal{T}(R)$ not in conflict with y is by definition still present in $\mathcal{T}(R \cup \{y\})$ and still does not have y in its defining set. And a configuration in $\mathcal{T}(R \cup \{y\})$ with y not in its defining set is by definition already present in $\mathcal{T}(R)$ and already there not in conflict with y.

The lemma implies that

$$k(r) \leqslant k_1(r) - k_2(r) + k_3(r),$$

where

$$k_1(r) = \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r} \sum_{y \in X \setminus R} |\mathcal{T}(R)|,$$

$$k_2(r) = \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r} \sum_{y \in X \setminus R} |\mathcal{T}(R \cup \{y\})|,$$

$$k_3(r) = \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r} \sum_{y \in X \setminus R} \deg(y, R \cup \{y\}).$$

Estimating $k_1(r)$. This is really simple.

$$\begin{aligned} k_1(r) &= \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r} \sum_{y \in X \setminus R} |\mathcal{T}(R)| \\ &= \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r}(n-r)|\mathcal{T}(R)| \\ &= \frac{d}{r}(n-r)t_r. \end{aligned}$$

Estimating $k_2(r)$. For this, we need to employ our earlier $(R, y) \mapsto (R \cup \{y\}, y)$ bijection again.

$$\begin{aligned}
k_2(r) &= \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r} \sum_{y \in X \setminus R} |\mathcal{T}(R \cup \{y\})| \\
&= \frac{1}{\binom{n}{r}} \sum_{Q \subseteq X, |Q|=r+1} \frac{d}{r} \sum_{y \in Q} |\mathcal{T}(Q)| \\
&= \frac{1}{\binom{n}{r+1}} \sum_{Q \subseteq X, |Q|=r+1} \frac{\binom{n}{r+1}}{\binom{n}{r}} \frac{d}{r}(r+1)|\mathcal{T}(Q)| \\
&= \frac{1}{\binom{n}{r+1}} \sum_{Q \subseteq X, |Q|=r+1} \frac{d}{r}(n-r)|\mathcal{T}(Q)| \\
&= \frac{d}{r}(n-r)t_{r+1} \\
&= \frac{d}{r+1}(n-(r+1))t_{r+1} + \frac{dn}{r(r+1)}t_{r+1} \\
&= k_1(r+1) + \frac{dn}{r(r+1)}t_{r+1}.
\end{aligned}$$

Estimating $k_3(r)$. This is similar to $k_2(r)$ and in addition uses a fact that we have employed before: $\sum_{y \in Q} \deg(y, Q) \leqslant d|\mathcal{T}(Q)|$.

$$\begin{aligned}
k_3(r) &= \frac{1}{\binom{n}{r}} \sum_{R \subseteq X, |R|=r} \frac{d}{r} \sum_{y \in X \setminus R} \deg(y, R \cup \{y\}) \\
&= \frac{1}{\binom{n}{r}} \sum_{Q \subseteq X, |Q|=r+1} \frac{d}{r} \sum_{y \in Q} \deg(y, Q) \\
&\leqslant \frac{1}{\binom{n}{r}} \sum_{Q \subseteq X, |Q|=r+1} \frac{d^2}{r}|\mathcal{T}(Q)| \\
&= \frac{1}{\binom{n}{r+1}} \sum_{Q \subseteq X, |Q|=r+1} \frac{\binom{n}{r+1}}{\binom{n}{r}} \frac{d^2}{r}|\mathcal{T}(Q)| \\
&= \frac{1}{\binom{n}{r+1}} \sum_{Q \subseteq X, |Q|=r+1} \frac{n-r}{r+1} \cdot \frac{d^2}{r}|\mathcal{T}(Q)| \\
&= \frac{d^2}{r(r+1)}(n-r)t_{r+1} \\
&= \frac{d^2 n}{r(r+1)}t_{r+1} - \frac{d^2}{r+1}t_{r+1}.
\end{aligned}$$

Summing up. Let us recapitulate: the overall expected number of conflicts is $O(n)$ plus

$$\sum_{r=1}^{n-1} k(r) = \sum_{r=1}^{n-1} (k_1(r) - k_2(r) + k_3(r)).$$

Using our previous estimates, $k_1(2), \ldots, k_1(n-1)$ are canceled by the first terms of $k_2(1), \ldots, k_2(n-2)$. The second term of $k_2(r)$ can be combined with the first term of $k_3(r)$, so that we get

$$\begin{aligned}
\sum_{r=1}^{n-1} (k_1(r) - k_2(r) + k_3(r)) &\leqslant k_1(1) - \underbrace{k_1(n)}_{=0} + n \sum_{r=1}^{n-1} \frac{d(d-1)}{r(r+1)} t_{r+1} - \sum_{r=1}^{n-1} \frac{d^2}{r+1} t_{r+1} \\
&\leqslant d(n-1)t_1 + d(d-1)n \sum_{r=1}^{n-1} \frac{t_{r+1}}{r(r+1)} \\
&= O\left(d^2 n \sum_{r=1}^{n} \frac{t_r}{r^2} \right).
\end{aligned}$$

The Delaunay case. We have argued that the expected number of conflicts asymptotically bounds the expected total location cost over all insertion steps. The previous equation tells us that this cost is proportional to $O(n)$ plus

$$O\left(9n \sum_{r=1}^{n} \frac{2(r+4) - 4}{r^2} \right) = O\left(n \sum_{r=1}^{n} \frac{1}{r} \right) = O(n \log n).$$

Here,

$$\sum_{r=1}^{n} \frac{1}{r} =: H_n$$

is the n-th Harmonic Number which is known to be approximately $\ln n$.

By going through the abstract framework of configuration spaces, we have thus analyzed the randomized incremental construction of the Delaunay triangulation of n points. According to Section 8.3, the expected update cost itself is only $O(n)$. The steps dominating the runtime are the location steps via the history graph. According to Section 8.5, all history graph searches (whose number is proportional to the number of conflicts) can be performed in expected time $O(n \log n)$, and this then also bounds the space requirements of the algorithm.

Exercise 8.7 *Design and analyze a sorting algorithm based on randomized incremental construction in configuration spaces. The input is a set S of numbers, and the output should be the sorted sequence (in increasing order).*

a) *Define an appropriate configuration space for the problem! In particular, the set of active configurations w.r.t.* S *should represent the desired sorted sequence.*

b) *Provide an efficient implementation of the incremental construction algorithm. "Efficient" means that the runtime of the algorithm is asymptotically dominated by the number of conflicts.*

c) *What is the expected number of conflicts (and thus the asymptotic runtime of your sorting algorithm) for a set* S *of* n *numbers?*

Questions

30. *What is a configuration space?* Give a precise definition! What is an active configuration?

31. *How do we get a configuration space from the problem of computing the Delaunay triangulation of a finite point set?*

32. *How many new active configurations do we get on average when inserting the* r*-th element?* Provide an answer for configuration spaces in general, and for the special case of the Delaunay triangulation.

33. *What is a conflict?* Provide an answer for configuration spaces in general, and for the special case of the Delaunay triangulation.

34. *Explain why counting the expected number of conflicts asymptotically bounds the cost for the history searches during the randomized incremental construction of the Delaunay triangulation!*

Chapter 9

Trapezoidal Maps

In this section, we will see another application of randomized incremental construction in the abstract configuration space framework. At the same time, this will give us an efficient algorithm for solving the general problem of point location, as well as a faster algorithm for computing all intersections between a given set of line segments.

9.1 The Trapezoidal Map

To start with, let us introduce the concept of a *trapezoidal map*.

We are given a set $S = \{s_1, \ldots, s_n\}$ of line segments in the plane (not necessarily disjoint). We make several general position assumptions.

We assume that no two segment endpoints and intersection points have the same x-coordinate. As an exception, we do allow several segments to share an endpoint. We also assume that no line segment is vertical, that any two line segments intersect in at most one point (which is a common endpoint, or a proper crossing), and that no three line segments have a common point. Finally, we assume that $s_i \subseteq [0,1]^2$ for all i (which can be achieved by scaling the coordinates of the segments accordingly).

Definition 9.1 *The* trapezoidal map *of S is the partition of $[0,1]^2$ into vertices, edges, and faces (called* trapezoids*), obtained as follows. Every segment endpoint and point of intersection between segments gets connected by two vertical extensions with the next feature below and above, where a feature is either another line segment or an edge of the bounding box $[0,1]^2$.*

Figure 9.1 gives an example.

(The general-position assumptions are made only for convenience and simplicity of the presentation. The various degeneracies can be handled without too much trouble, though we will not get into the details.)

The trapezoids of the trapezoidal map are "true" trapezoids (quadrangles with two parallel vertical sides), and triangles (which may be considered as degenerate trapezoids).

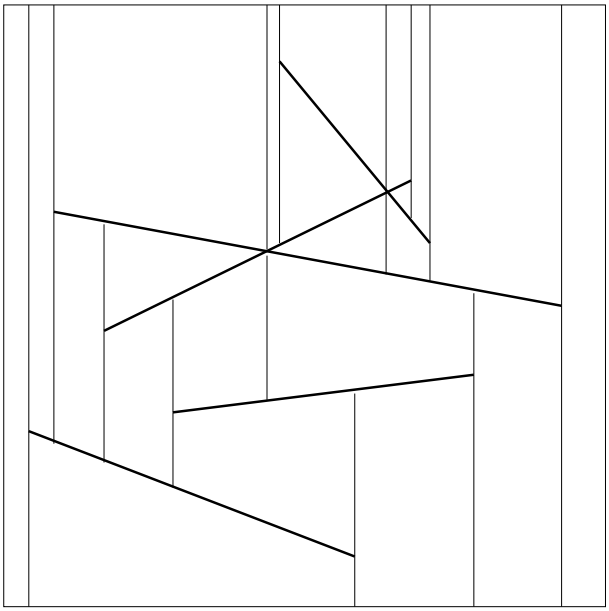


Figure 9.1: *The trapezoidal map of five line segments (depicted in bold)*

9.2 Applications of trapezoidal maps

In this chapter we will see two applications of trapezoidal maps (there are others):

1. *Point location:* Given a set of n segments in the plane, we want to preprocess them in order to answer point-location queries: given a point p, return the cell (connected component of the complement of the segments) that contains p (see Figure 9.2). This is a more powerful alternative to Kirkpatrick's algorithm that handles only triangulations, and which is treated in Section 10.5. The preprocessing constructs the trapezoidal map of the segments (Figure 9.3) in expected time $O(n \log n + K)$, where K is the number of intersections between the input segments; the query time will be $O(\log n)$ in expectation.

2. *Line segment intersection:* Given a set of n segments, we will report all segment intersections in expected time $O(n \log n + K)$. This is a faster alternative to the line-sweep algorithm we saw in Section 4.2, which takes time $O((n + K) \log n)$.

9.3 Incremental Construction of the Trapezoidal Map

We can construct the trapezoidal map by inserting the segments one by one, in random order, always maintaining the trapezoidal map of the segments inserted so far. In order to perform manipulations efficiently, we can represent the current trapezoidal map as a doubly-connected edge list (see Section 5.2), for example.

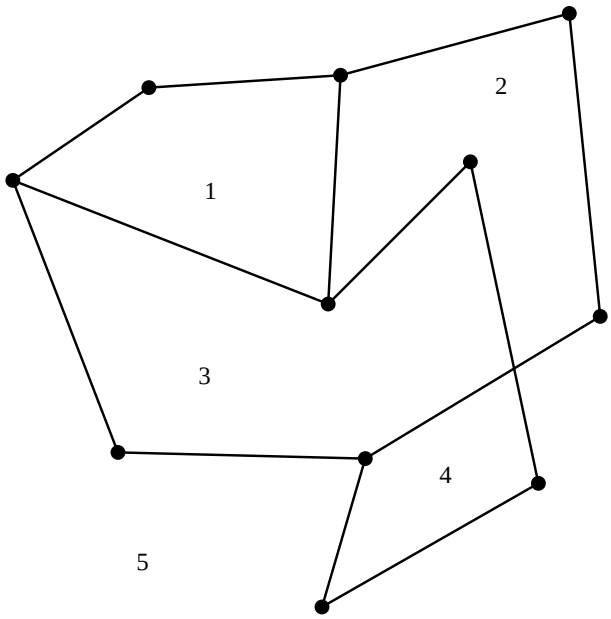


Figure 9.2: *The general point location problem defined by a set of (possibly intersecting) line segments. In this example, the segments partition the plane into 5 cells.*

Suppose that we have already inserted segments $s_1, \ldots, s_{r-1}$, and that the resulting trapezoidal map $\mathcal{T}_{r-1}$ looks like in Figure 9.1. Now we insert segment s_r (see Figure 9.4).

Here are the four steps that we need to do in order to construct $\mathcal{T}_r$.

1. **Find** the trapezoid $\square_0$ of $\mathcal{T}_{r-1}$ that contains the left endpoint of s_r.

2. **Trace** s_r through $\mathcal{T}_{r-1}$ until the trapezoid containing the right endpoint of s_r is found. To get from the current trapezoid $\square$ to the next one, traverse the boundary of $\square$ until the edge is found through which s_r leaves $\square$.

3. **Split** the trapezoids intersected by s_r. A trapezoid $\square$ may get replaced by

   - two new trapezoids (if s_r intersects two vertical extensions of $\square$);
   - three new trapezoids (if s_r intersects one vertical extension of $\square$);
   - four new trapezoids (if s_r intersects no vertical extension of $\square$).

4. **Merge** trapezoids by removing parts of vertical extensions that do not belong to $\mathcal{T}_r$ anymore.

Figure 9.5 illustrates the **Trace** and **Split** steps. s_6 intersects 5 trapezoids, and they are being split into $3, 3, 4, 3$, and 3 trapezoids, respectively.

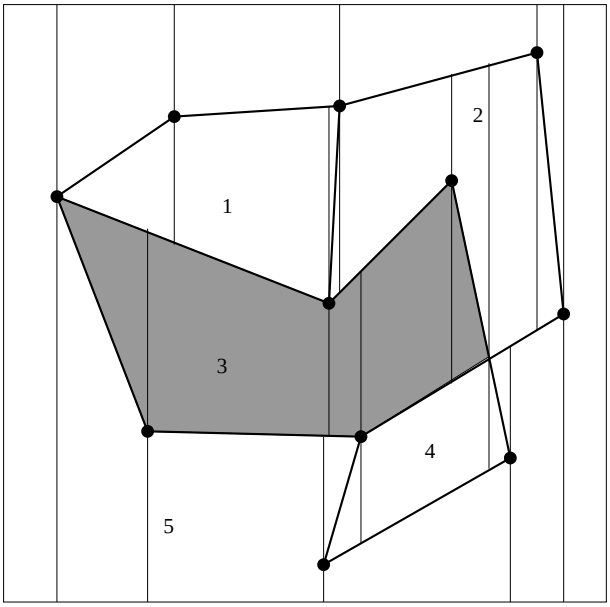


Figure 9.3: *The trapezoidal map is a refinement of the partition of the plane into cells. For example, cell 3 is a union of five trapezoids.*

The **Merge** step is shown in Figure 9.6. In the example, there are two vertical edges that need to be removed (indicated with a cross) because they come from vertical extensions that are cut off by the new segment. In both cases, the two trapezoids to the left and right of the removed edge are being merged into one trapezoid (drawn shaded).

9.4 Using trapezoidal maps for point location

Recall that in the point location problem we want to preprocess a given set S of segments in order to answer subsequent point-location queries: S partitions the plane into connected *cells* and we want to know, given a query point q, to which cell q belongs, see Figure 9.2.

Note that the trapezoidal map of S is a *refinement* of the partition of the plane into cells, in the sense that a cell might be partitioned into several trapezoids, but every trapezoid belongs to a single cell, see Figure 9.3. Thus, once the trapezoidal map of S is constructed, we can easily "glue together" trapezoids that touch along their vertical sides, obtaining the original cells. Then we can answer point-location queries using the same routine that performs the **Find** step (whose implementation will be described below).

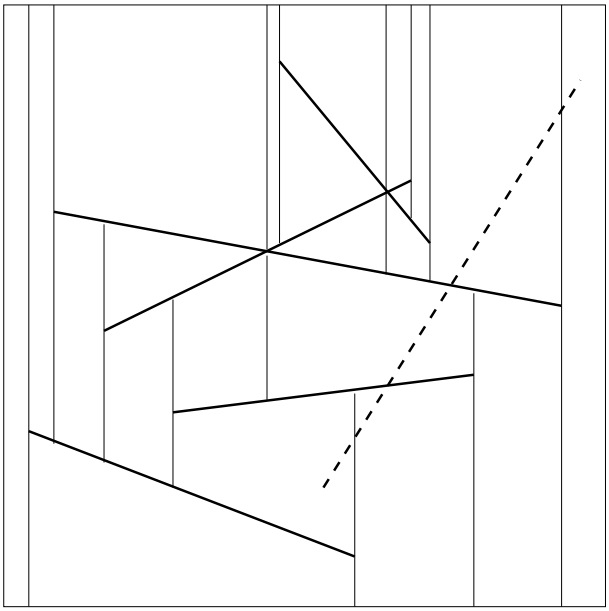


Figure 9.4: *A new segment (dashed) is to be inserted*

9.5 Analysis of the incremental construction

In order to analyze the runtime of the incremental construction, we insert the segments in random order, and we employ the configuration space framework. We also implement the **Find** step in such a way that the analysis boils down to conflict counting, just as for the Delaunay triangulation.

9.5.1 Defining The Right Configurations

Recall that a configuration space is a quadruple $\mathcal{S} = (X, \Pi, D, K)$, where X is the ground set, Π is the set of configurations, D is a mapping that assigns to each configuration its defining elements ("generators"), and K is a mapping that assigns to each configuration its conflict elements ("killers").

It seems natural to choose $X = S$, the set of segments, and to define Π as the set of all possible trapezoids that could appear in the trapezoidal map of some subset of segments. Indeed, this satisfies one important property of configuration spaces: for each configuration, the number of generators is constant.

Lemma 9.2 *For every trapezoid $\square$ in the trapezoidal map of $R \subseteq S$, there exists a set $D \subseteq R$ of at most four segments, such that $\square$ is in the trapezoidal map of D.*

Proof. By our general position assumption, each non-vertical side of $\square$ is a subset of a unique segment in R, and each vertical side of $\square$ is induced by a unique (left or right) endpoint, or by the intersection of two unique segments. In the latter case, one of these

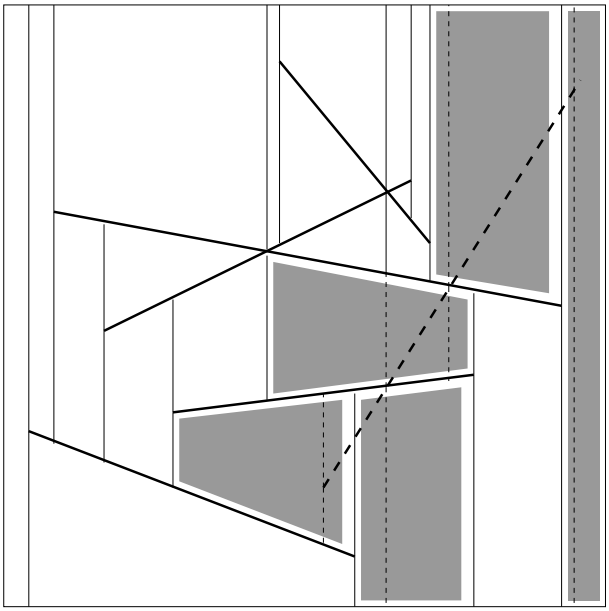


Figure 9.5: *The* **Trace** *and* **Split** *steps*

segments also contributes a non-vertical side, and in the former case, we attribute the endpoint to the "topmost" segment with that (left or right) endpoint. It follows that there is a set of at most four segments whose trapezoidal map already contains □. □

But there is a problem with this definition of configurations. Recall that we can apply the general configuration space analysis only if

(i) the cost of updating $\mathcal{T}_{r-1}$ to $\mathcal{T}_r$ is proportional to the *structural change*, the number of configurations in $\mathcal{T}_r \setminus \mathcal{T}_{r-1}$; and

(ii) the expected cost of all **Find** operations during the randomized incremental construction is proportional to the expected number of conflicts. (This is the "conflict counting" part.)

Here we see that already (i) fails. During the **Trace** step, we traverse the boundary of each trapezoid intersected by s_r in order to find the next trapezoid. Even if s_r intersects only a small number of trapezoids (so that the structural change is small), the traversals may take very long. This is due to the fact that a trapezoid can be incident to a large number of edges. Consider the trapezoid labeled □ in Figure 9.7. It has many incident vertical extensions from above. Tracing a segment through such a trapezoid takes time that we cannot charge to the structural change.

To deal with this, we slightly adapt our notion of configuration.

Definition 9.3 *Let* Π *be the set of all trapezoids together with at most one incident vertical edge ("trapezoids with tail") that appear in the trapezoidal map of some*

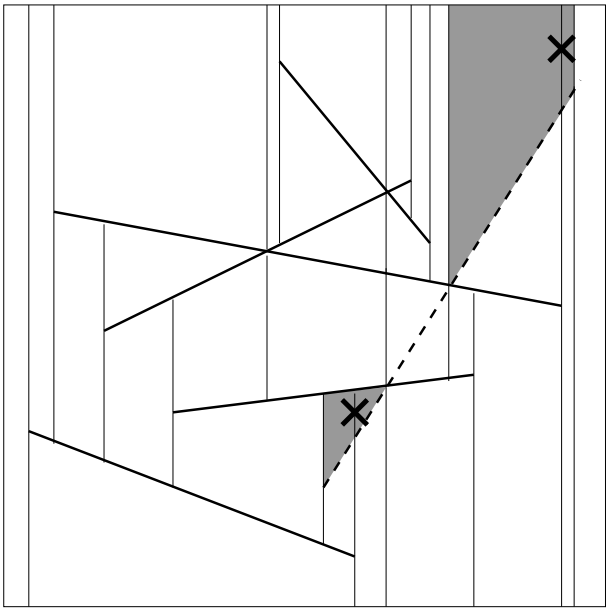


Figure 9.6: *The* **Merge** *steps*

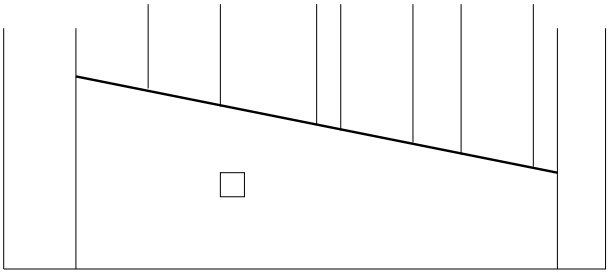


Figure 9.7: *Trapezoids may have arbitrarily large complexity*

subset of $X = S$*, see Figure 9.8. A trapezoid without any incident vertical edge is also considered a trapezoid with tail.*

As it turns out, we still have constantly many generators.

Lemma 9.4 *For every trapezoid with tail* $\square$ *in the trapezoidal map of* $R \subseteq S$*, there exists a set* $D \subseteq R$ *of at most six segments, such that* $\square$ *is in the trapezoidal map of* D*.*

Proof. We already know from Lemma 9.2 that the trapezoid without tail has at most 4 generators. And since the tail is induced by a unique segment endpoint or by the intersection of a unique pair of segments, the claim follows. $\square$

Here is the complete specification of our configuration space $\mathcal{S} = (X, \Pi, D, K)$.

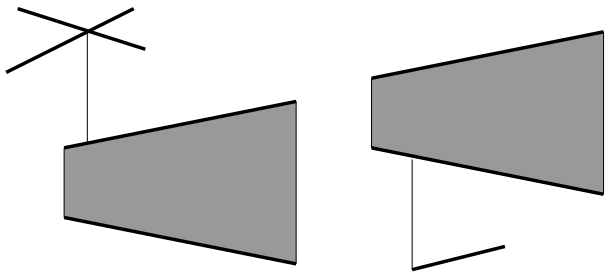


Figure 9.8: *A trapezoid with tail is a trapezoid together with at most one vertical edge attached to its upper or its lower segment.*

Definition 9.5 *Let $X = S$ be the set of segments, and Π the set of all trapezoids with tail. For each trapezoid with tail $\square$, $D(\square)$ is the set of at most 6 generators. $K(\square)$ is the set of all segments that intersect $\square$ in the interior of the trapezoid, or cut off some part of the tail, or replace the topmost generator of the left or right side, see Figure 9.9.*

Then $\mathcal{S} = (X, \Pi, D, K)$ is a configuration space of dimension at most 6, by Lemma 9.4. The only additional property that we need to check is that $D(\square) \cap K(\square) = \emptyset$ for all trapezoids with tail, but this is clear since no generator of $\square$ properly intersects the trapezoid of $\square$ or cuts off part of its tail.

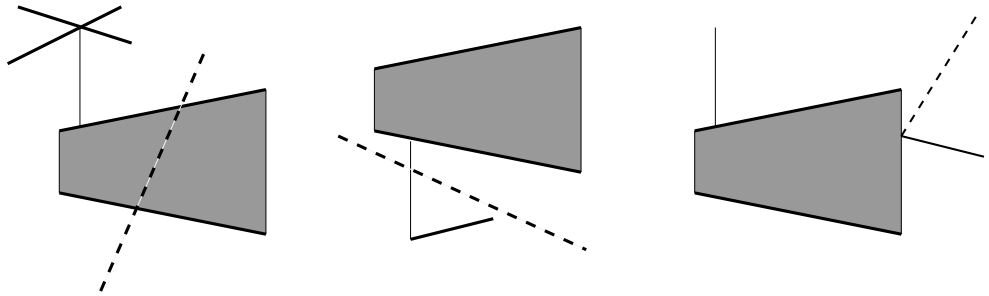


Figure 9.9: *A trapezoid with tail $\square$ is in conflict with a segment s (dashed) if s intersects $\square$ in the interior of the trapezoid (left), or cuts off part of the tail (middle), or is a new topmost segment generating a vertical side.*

9.5.2 Update Cost

Now we can argue that the update cost can be bounded by the structural change. We employ the same trick as for Delaunay triangulations. We prove that the update cost is in each step $r - 1 \to r$ proportional to the number of configurations that are being *destroyed*. Over the whole algorithm, we cannot destroy more configurations than we create, so the bound that we get is also a bound in terms of the overall structural change.

Lemma 9.6 *In updating $\mathcal{T}_{r-1}$ to $\mathcal{T}_r$, the steps* **Trace**, **Split**, *and* **Merge** *can be performed in time proportional to the number of trapezoids with tail in $\mathcal{T}_{r-1} \setminus \mathcal{T}_r$.*

Proof. By definition, the complexity (number of edges) of a trapezoid is proportional to the number of trapezoids with tail that share this trapezoid. This means, the cost of traversing the trapezoid can be charged to the trapezoids with tail containing it, and all of them will be destroyed (this includes the trapezoids with tail that just change their left or right generator; in the configuration space, this is also a destruction). This takes care of the **Trace** step. The **Split** and **Merge** steps can be done within the same asymptotic time bounds since they can be performed by traversing the boundaries of all intersected trapezoids a constant number of times each. For efficiently doing the manipulations on the trapezoidal map, we can for example represent it using a doubly-connected edge list. □

We can now employ the general configuration space analysis to bound the *expected* structural change throughout the randomized incremental construction; as previously shown, this asymptotically bounds the expected cost of the **Trace**, **Split**, and **Merge** steps throughout the algorithm. Let us recall the general bound.

Theorem 9.7 *Let $\mathcal{S} = (X, \Pi, D, K)$ be a configuration space of fixed dimension with $|X| = n$. The expected number of configurations that are created throughout the algorithm is bounded by*

$$O\left(\sum_{r=1}^{n} \frac{t_r}{r}\right),$$

where t_r is the expected size of $\mathcal{T}_r$, the expected number of active configurations after r insertion steps.

9.5.3 The History Graph

Here is how we realize the **Find** step (as well as the point-location queries for our point-location application). It is a straightforward history graph approach as for Delaunay triangulations. Every trapezoid that we ever create is a node in the history graph; whenever a trapezoid is destroyed, we add outgoing edges to its (at most four) successor trapezoids. Note that trapezoids are destroyed during the steps **Split** and **Merge**. In the latter step, every destroyed trapezoid has only one successor trapezoid, namely the one it is merged into. It follows that we can prune the nodes of the "ephemeral" trapezoids that exist only between the **Split** and **Merge** steps. What we get is a history graph of degree at most 4, such that every non-leaf node corresponds to a trapezoid in $\mathcal{T}_{r-1} \setminus \mathcal{T}_r$, for some r.

9.5.4 Cost of the Find step

We can use the history graph for point location during the **Find** step. Given a segment endpoint p, we start from the bounding box (the unique trapezoid with no generators) that is certain to contain p. Since for every trapezoid in the history graph, its area is covered by the at most four successor trapezoids, we can simply traverse the history graph along directed edges until we reach a leaf that contains p. This leaf corresponds to the trapezoid of the current trapezoidal map containing p. By the outdegree-4-property, the cost of the traversal is proportional to the length of the path that we traverse.

Here is the crucial observation that allows us to reduce the analysis of the **Find** step to "conflict counting". Note that this is is precisely what we also did for Delaunay triangulations, except that there, we had to deal explicitly with "ephemeral" triangles.

Recall Definition 8.5, according to which a *conflict* is a pair $(\square, s)$ where $\square$ is a trapezoid with tail, contained in some intermediate trapezoidal map, and $s \in K(\square)$.

Lemma 9.8 *During a run of the incremental construction algorithm for the trapezoidal map, the total number of history graph nodes traversed during all* **Find** *steps is bounded by the number of conflicts during the run.*

Proof. Whenever we traverse a node (during insertion of segment s_r, say), the node corresponds to a trapezoid $\square$ (which we also consider as a trapezoid with tail) in some set $\mathcal{T}_s, s < r$, such that $p \in \square$, where p is the left endpoint of the segment s_r. We can therefore uniquely identify this edge with the conflict $(\square, s_r)$. The statement follows. $\square$

Now we can use the configuration space analysis that precisely bounds the *expected* number of conflicts, and therefore the expected cost of the **Find** steps over the whole algorithm. Let us recapitulate the bound.

Theorem 9.9 *Let $\mathcal{S} = (X, \Pi, D, K)$ be a configuration space of fixed dimension d with $|X| = n$. The expected number of conflicts during randomized incremental construction of $\mathcal{T}_n$ is bounded by*

$$O\left(n \sum_{r=1}^{n} \frac{t_r}{r^2}\right),$$

where t_r is as before the expected size of $\mathcal{T}_r$.

9.5.5 Applying the General Bounds

Let us now apply Theorem 9.7 and Theorem 9.9 to our concrete situation of trapezoidal maps. What we obviously need to determine for that is the quantity t_r, the expected number of active configurations after r insertion steps.

Recall that the configurations are the trapezoids with tail that exist at this point. The first step is easy.

Observation 9.10 *In every trapezoidal map, the number of trapezoids with tail is proportional to the number vertices.*

Proof. Every trapezoid with tail that actually has a tail can be charged to the vertex of the trapezoidal map on the "trapezoid side" of the tail. No vertex can be charged twice in this way. The trapezoids with no tail are exactly the faces of the trapezoidal map, and since the trapezoidal map is a planar graph, their number is also proportional to the number vertices. □

Using this observation, we have therefore reduced the problem of computing t_r to the problem of computing the expected number of vertices in $\mathcal{T}_r$. To count the latter, we note that every segment endpoint and every segment intersection generates 3 vertices: one at the point itself, and two where the vertical extensions hit another feature. Here, we are sweeping the 4 bounding box vertices under the rug.

Observation 9.11 *In every trapezoidal map of* r *segments, the number of vertices is*

$$6r + 3k,$$

where k *is the number of pairwise intersections between the* r *segments.*

So far, we have not used the fact that we have a random insertion order, but this comes next.

Lemma 9.12 *Let* K *be the total number of pairwise intersections between segments in* S*, and let* k_r *be the random variable for the expected number of pairwise intersections between the first* r *segments inserted during randomized incremental construction. Then*

$$k_r = K\frac{\binom{n-2}{r-2}}{\binom{n}{r}} = K\frac{r(r-1)}{n(n-1)}.$$

Proof. Let us consider the intersection point of two fixed segments s and s'. This intersection point appears in $\mathcal{T}_r$ if and only both s and s' are among the first r segments. There are $\binom{n}{r}$ ways of choosing the set of r segments (and all choices have the same probability); since the number of r-element sets containing s and s' is $\binom{n-2}{r-2}$, the probability for the intersection point to appear is

$$\frac{\binom{n-2}{r-2}}{\binom{n}{r}} = \frac{r(r-1)}{n(n-1)}.$$

Summing this up over all K intersection points, and using linearity of expectation, the statement follows. □

From Observation 9.10, Observation 9.11 and Lemma 9.12, we obtain the following

Corollary 9.13 *The expected number* t_r *of active configurations after* r *insertion steps is*

$$t_r = O\left(r + K\frac{r(r-1)}{n^2}\right).$$

Plugging this into Theorem 9.7 and Theorem 9.9, we obtain the following final

Theorem 9.14 *Let* S *be a set of* n *segments in the plane, with a total of* K *pairwise intersections. The randomized incremental construction computes the trapezoidal map of* S *in time*

$$O(n \log n + K).$$

Proof. We already know that the expected update cost (subsuming steps **Trace**, **Split**, and **Merge**) is proportional to the expected overall structural change, which by Theorem 9.7 is

$$O\left(\sum_{r=1}^{n} \frac{t_r}{r}\right) = O(n) + O\left(\frac{K}{n^2}\sum_{r=1}^{n} r\right) = O(n + K).$$

We further know that the expected point location cost (subsuming step **Find**) is proportional to the overall expected number of conflicts which by Theorem 9.9 is

$$O\left(n\sum_{r=1}^{n} \frac{t_r}{r^2}\right) = O(n \log n) + O\left(\frac{K}{n}\sum_{r=1}^{n} 1\right) = O(n \log n + K).$$

□

9.6 Analysis of the point location

Finally, we return to the application of trapezoidal maps for point location. We make precise what we mean by saying that "point-location queries are handled in $O(\log n)$ expected time", and we prove our claim.

Lemma 9.15 *Let* $S = \{s_1, \ldots, s_n\}$ *be any set of* n *segments. Then there exists a constant* $c > 0$ *such that, with high probability (meaning, with probability tending to* 1 *as* $n \to \infty$*), the history graph produced by the random incremental construction answers* every possible *point-location query in time at most* $c \log n$.

Note that our only randomness assumption is over the random permutation of S chosen at the beginning of the incremental construction. We do not make any randomness assumption on the given set of segments.

The proof of Lemma 9.15 is by a typical application of Chernoff's bound followed by the union bound.

Recall (or please meet) Chernoff's bound:

Lemma 9.16 *Let $X_1, X_2, \ldots, X_n$ be independent 0/1 random variables, and let $X = X_1 + \cdots + X_n$. Let $p_i = \Pr[X_i = 1]$, and let $\mu = \mathrm{E}[X] = p_1 + \cdots + p_n$. Then,*

$$\Pr[X < (1-\delta)\mu] < \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu} \qquad \text{for every } 0 < \delta < 1;$$

$$\Pr[X > (1+\delta)\mu] < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} \qquad \text{for every } \delta > 0.$$

The important thing to note is that $e^{-\delta}/(1-\delta)^{1-\delta}$ as well as $e^{\delta}/(1+\delta)^{1+\delta}$ are strictly less than 1 for every fixed $\delta > 0$, and decrease with increasing δ.

Now back to the proof of Lemma 9.15:

Proof. First note that, even though there are infinitely many possible query points, there is only a finite number of combinatorially distinct possible *queries*: If two query points lie together in every trapezoid (either both inside or both outside), among all possible trapezoids defined by segments of S, then there is no difference in querying one point versus querying the other, as far as the algorithm is concerned. Since there are $O(n^4)$ possible trapezoids (recall that each trapezoid is defined by at most four segments), there are only $O(n^4)$ queries we have to consider.

Fix a query point q. We will show that there exists a large enough constant $c > 0$, such that only with probability at most $O(n^{-5})$ does the query on q take more than $c \log n$ steps.

Let $s_1, s_2, \ldots, s_n$ be the random order of the segments chosen by the algorithm, and for $1 \leqslant r \leqslant n$ let $\mathcal{T}_r$ be the trapezoidal map generated by the first r segments. Note that for every r, the point q belongs to exactly one trapezoid of $\mathcal{T}_r$. The question is how many times the trapezoid containing q changes during the insertion of the segments, since these are exactly the trapezoids of the history graph that will be visited when we do a point-location query on q.

For $1 \leqslant r \leqslant n$, let A_r be the event that the trapezoid containing q changes from $\mathcal{T}_{r-1}$ to $\mathcal{T}_r$. What is the probability of A_r? As in Section 8.3, we "run the movie backwards": To obtain $\mathcal{T}_{r-1}$ from $\mathcal{T}_r$, we delete a random segment from among $s_1, \ldots, s_r$; the probability that the trapezoid containing q in $\mathcal{T}_r$ is destroyed is at most $4/r$, since this trapezoid is defined by at most four segments. Thus, $\Pr[A_r] \leqslant 4/r$, independently of every other A_s, $s \neq r$.

For each r let X_r be a random variable equal to 1 if A_r occurs, and equal to 0 otherwise. We are interested in the quantity $X = X_1 + \cdots + X_r$. Then $\mu = \mathrm{E}[X] = \sum_{r=1}^{n} \Pr[A_r] = 4 \ln n + O(1)$. Applying Chernoff's bound with $\delta = 2$ (it is just a matter of choosing δ large enough), we get

$$\Pr[X > (1+\delta)\mu] < 0.273^{\mu} = O(0.273^{4 \ln n}) = O(n^{-5.19}),$$

so we can take our c to be anything larger than $4(1+\delta) = 12$.

Thus, for every fixed query q, the probability of a "bad event" (a permutation that results in a long query time) is $O(n^{-5})$. Since there are only $O(n^4)$ possible choices for

q, by the union bound the probability of *some* q having a bad event is $O(1/n)$, which tends to zero with n. □

9.7 The trapezoidal map of a simple polygon

An important special case of the trapezoidal map is obtained when the input segments form a simple polygon; seee Figure 9.10. In this case, we are mostly interested in the part of the trapezoidal map inside the polygon, since that parts allows us to obtain a triangulation of the polygon in linear time.

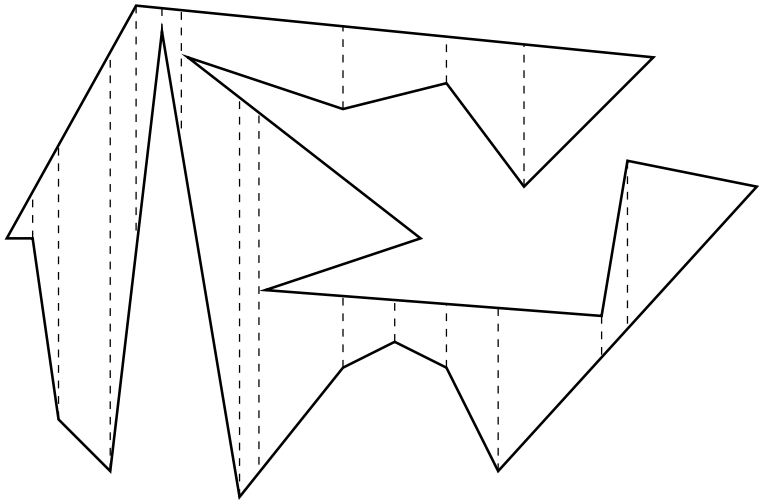


Figure 9.10: *The trapezoidal map inside a simple polygon*

To get a triangulation, we first go through all trapezoids; we know that each trapezoid must have one polygon vertex on its left and one on its right boundary (due to general position, there is actually *exactly* one vertex on each of these two boundaries). Whenever the segment connecting these two vertices is not an edge of the polygon, we have a diagonal, and we insert this diagonal. Once we have done this for all trapezoids, it is easily seen that we have obtained a subdivision of the polygon into x-monotone polygons, each of which can be triangualated in linear time; see Exercise 9.24. This immediately allows us to improve over the statement of Exercise 2.19.

Corollary 9.17 *A simple polygon with n vertices can be triangulated in expected time $O(n \log n)$.*

Proof. By Theorem 9.14, the trapezoidal decomposition induced by the segments of a simple polygon can be combuted in expected time $O(n \log n)$, since there are no intersections between the segments. Using the above linear-time triangulation algorithm from the trapezoidal map, the result follows. □

The goal of this section is to further improve this bound and show the following result.

Theorem 9.18 *Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of edges of an n-vertex simple polygon, in counterclockwise order around the polygon. The trapezoidal map induced by S (and thus also a triangulation of S) can be computed in exptected time $O(n \log^* n)$.*

Informally speaking, the function $\log^* n$ is the number of times we have to iterate the operation of taking (binary) logarithms, before we get from n down to 1. Formally, we define

$$\log^{(h)}(n) = \begin{cases} n, & \text{if } h = 0 \\ \log^{(h-1)}(\log n), & \text{otherwise} \end{cases}$$

as the h-times iterated logarithm, and for $n \geqslant 1$ we set

$$\log^* n = \max\{h : \log^{(h)} n \geqslant 1\}.$$

For example, we have

$$\log^*(2^{65536}) = 5,$$

meaning that for all practical purposes, $\log^* n \leqslant 5$; a bound of $O(n \log^* n)$ is therefore very close to a linear bound.

History flattening. Recall that the bottleneck in the randomized incremental construction is the **Find** step. Using the special structure we have (the segments form a simple polygon in this order), we can speed up this step. Suppose that at some point during incremental construction, we have built the trapezoidal map of a subset of r segments, along with the history graph. We now flatten the history by removing all trapezoids that are not in $\mathcal{T}_r$, the current trapezoidal map. To allow for point location also in the future, we need an "entry point" into the flattened history, for every segment not inserted so far (the old entry point for all segments was the bounding unit square $[0, 1]^2$).

Lemma 9.19 *Let S be the set of edges of an n-vertex simple polygon, in counterclockwise order around the polygon. For $R \subseteq S$, let $\mathcal{T}(R)$ be the trapezoidal map induced by R. In time proportional to n plus the number of conflicts between trapezoids in $\mathcal{T}(R)$ and segments in $S \setminus R$, we can find for all segment $s \in S$ a trapezoid of $\mathcal{T}(R)$ that contains an endpoint of s.*

Proof. In a trivial manner (and in time $O(n)$), we do this for the first segment s_1 and its first endpoint p_1. Knowing the trapezoid $\square_i$ containing p_i, the first endpoint of s_i, we trace the segment s_i through $\mathcal{T}(R)$ until p_{i+1}, its other endpoint and first endpoint of s_{i+1} is found, along with the trapezoid $\square_{i+1}$ containing it. This is precisely what we also did in the **Trace** Step 2 of the incremental construction in Section 9.3.

By Lemma 9.6 (pretending that we are about to insert s_i), the cost of tracing s_i through $\mathcal{T}(R)$ is proportional to the size of $\mathcal{T}(R) \setminus \mathcal{T}(R \cup \{s_i\})$, equivalently, the number of conflicts between trapezoids in $\mathcal{T}(R)$ and s_i. The statement follows by adding up the costs for all i. □

This is exactly where the special structure of our segments forming a polygon helps. After locating p_i, we can locate the next endpoint p_{i+1} in time proportional to the structural change that the insertion of s_i would cause. We completely avoid the traversal of old history trapezoids that would be necessary for an efficient location of p_{i+1} from scratch.

Next we show what Lemma 9.19 gives us in expectation.

Lemma 9.20 *Let S be the set of edges of an n-vertex simple polygon, in counterclockwise order around the polygon, and let $\mathcal{T}_r$ be the trapezoidal map obtained after inserting r segments in random order. In expected time $O(n)$, we can find for each segment s not inserted so far a trapezoid of $\mathcal{T}_r$ containing an endpoint of s.*

Proof. According to Lemma 9.19, the expected time is bounded by $O(n + \ell(r))$, where

$$\ell(r) = \frac{1}{\binom{n}{r}} \sum_{R \subseteq S, |R|=r} \sum_{y \in S \setminus R} |\{\square \in \mathcal{T}(R) : y \in K(\square)\}|.$$

This looks very similar to the bound on the quantity $k(r)$ that we have derived in Section 8.5 to count the expected number of conflicts in general configuration spaces:

$$k(r) \leqslant \frac{1}{\binom{n}{r}} \sum_{R \subseteq S, |R|=r} \frac{d}{r} \sum_{y \in S \setminus R} |\{\Delta \in \mathcal{T}(R) : y \in K(\Delta)\}|.$$

Indeed, the difference is only an additional factor of $\frac{d}{r}$ in the latter. For $k(r)$, we have then computed the bound

$$k(r) \leqslant k_1(r) - k_2(r) + k_3(r) \leqslant \frac{d}{r}(n-r)t_r - \frac{d}{r}(n-r)t_{r+1} + \frac{d^2}{r(r+1)}(n-r)t_{r+1}, \quad (9.21)$$

where t_r is the expected size of $\mathcal{T}_r$. Hence, to get a bound for $\ell(r)$, we simply have to cancel $\frac{d}{r}$ from all terms to obtain

$$\ell(r) \leqslant (n-r)t_r - (n-r)t_{r+1} + \frac{d}{r+1}(n-r)t_{r+1} = O(n),$$

since $t_r \leqslant t_{r+1} = O(r)$ in the case of nonintersecting line segments. □

The faster algorithm. We proceed as in the regular randomized incremental construction, except that we frequently and at well-chosen points flatten the history. Let us define

$$N(h) = \left\lceil \frac{n}{\log^{(h)} n} \right\rceil, \quad 0 \leqslant h \leqslant \log^* n.$$

We have $N(0) = 1, N(\log^* n) \leqslant n$ and $N(\log^* n + 1) > n$. We insert the segments in random order, but proceed in $\log^* n + 1$ *rounds*. In round $h = 1, \ldots, \log^* n + 1$, we do the following.

(i) Flatten the history graph by finding for each segment s not inserted so far a trapezoid of the current trapezoidal map containing an endpoint of s.

(ii) Insert the segments $N(h-1)$ up to $N(h) - 1$ in the random order, as usual, but starting from the flat history established in (i).

In the last round, we have $N(h) - 1 \geqslant n$, so we stop with segment n in the random order.

From Lemma 9.20, we know that the expected cost of step (i) over all rounds is bounded by $O(n \log^* n)$ which is our desired overall bound. It remains to prove that the same bound also deals with step (ii). We do not have to worry about the overall expected cost of performing the structural changes in the trapezoidal map: this will be bounded by $O(n)$, using $t_r = O(r)$ and Theorem 9.7. It remains to analyze the **Find** step, and this is where the history flattening leads to a speedup. Adapting Lemma 9.8 and its proof accordingly, we obtain the following.

Lemma 9.22 *During round h of the fast incremental construction algorithm for the trapezoidal map, the total number of history graph nodes traversed during all* **Find** *steps is bounded by $N(h) - N(h-1)$[1], plus the number of conflicts between trapezoids that are created during round h, and segments inserted during round h.*

Proof. The history in round h only contains trapezoids that are active at some point in round h. On the one hand, we have the "root nodes" present immediately after flattening the history, on the other hand the trapezoids that are created during the round. The term $N(h) - N(h-1)$ accounts for the traversals of the root nodes during round h. As in the proof of Lemma 9.8, the traversals of other history graph nodes can be charged to the number of conflicts between trapezoids that are created during round h and segments inserted during round h. □

To count the expected number of conflicts involving trapezoids created in round h, we go back to the general configuration space framework once more. With $k_h(r)$ being the expected number of conflicts created in step r, *restricted* to the ones involving segments inserted in round h, we need to bound

$$\kappa(h) = \sum_{r=N(h-1)}^{N(h)-1} k_h(r).$$

[1] this amounts to $O(n)$ throughout the algorithm and can therefore be neglected

As in (9.21), we get

$$k_h(r) \leqslant \frac{d}{r}(N(h)-r)t_r - \frac{d}{r}(N(h)-r)t_{r+1} + \frac{d^2}{r(r+1)}(N(h)-r)t_{r+1},$$

where we have replaced n with $N(h)$, due to the fact that we do not need to consider segments in later rounds. Then $t_r = O(r)$ yields

$$\begin{aligned}
\kappa(h) &\leqslant O\left(N(h)\sum_{r=N(h-1)}^{N(h)-1}\frac{1}{r}\right)\\
&= O\left(N(h)\log\frac{N(h)}{N(h-1)}\right)\\
&= O\left(N(h)\log\frac{\log^{(h-1)} n}{\log^{(h)} n}\right)\\
&= O\left(N(h)\log^{(h)} n\right) = O(n).
\end{aligned}$$

It follows that step (ii) of the fast algorithm also requires expected linear time per round, and Theorem 9.18 follows.

Exercise 9.23 *a) You are given a set of n pairwise disjoint line segments. Find an algorithm to answer* vertical ray shooting *queries in $O(\log n)$ time. That is, preprocess the data such that given a query point q you can report in $O(\log n)$ time which segment is the first above q (or if there are none). Analyze the running time and the space consumption of the preprocessing.*

b) What happens if we allow intersections of the line segments? Explain in a few words how you have to adapt your solution and how the time and space complexity would change.

Exercise 9.24 *Show that an n-vertex x-monotone polygon can be triangulated in time $O(n)$. (As usual a polygon is given as a list of its vertices in counterclockwise order. A polygon P is called x-monotone if for all vertical lines ℓ, the intersection $\ell \cap P$ has at most one component.)*

Questions

35. *What is the definition of a trapezoidal map?*

36. *How does the random incremental construction of a trapezoidal map proceed?* What are the main steps to be executed at each iteration?

37. *How can trapezoidal maps be used for the point location problem?*

38. *What is the configuration space framework?* Recall Section 8.3.

39. *What is a naive way of defining a configuration in the case of trapezoids, and why does it fail?*

40. *What is a more successful way of defining a configuration?* Why do things work in this case?

41. *What is the history graph, and how is it used to answer point location queries?*

42. *What is the performance of the random incremental construction of the trapezoidal map when used for the point-location problem?* Be precise!

43. *What probabilistic techniques are used in proving this performance bound?*

44. *How can you speed up the randomized incremental construction in the case where the input segments from a simple polygon? Sketch the necessary changes to the algorithm, and how they affect the analysis.*

Chapter 10

Voronoi Diagrams

10.1 Post Office Problem

Suppose there are n post offices $p_1, \ldots p_n$ in a city. Someone who is located at a position q within the city would like to know which post office is closest to him. Modeling the city as a planar region, we think of $p_1, \ldots p_n$ and q as points in the plane. Denote the set of post offices by $P = \{p_1, \ldots p_n\}$.

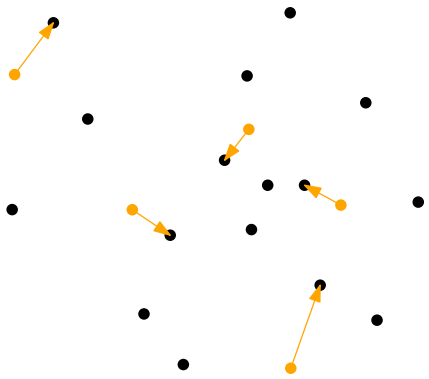


Figure 10.1: *Closest post offices for various query points.*

While the locations of post offices are known and do not change so frequently, we do not know in advance for which—possibly many—query locations the closest post office is to be found. Therefore, our long term goal is to come up with a data structure on top of P that allows to answer any possible query efficiently. The basic idea is to apply a so-called *locus approach*: we partition the query space into regions on which is the answer is the same. In our case, this amounts to partition the plane into regions such that for all points within a region the same point from P is closest (among all points from P).

As a warmup, consider the problem for two post offices $p_i, p_j \in P$. For which query locations is the answer p_i rather than p_j? This region is bounded by the bisector of p_i and p_j, that is, the set of points which have the same distance to both points.

Proposition 10.1 *For any two distinct points in $\mathbb{R}^d$ the bisector is a hyperplane, that is, in $\mathbb{R}^2$ it is a line.*

Proof. Let $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ be two points in $\mathbb{R}^d$. The bisector of p and q consists of those points $x = (x_1, \ldots, x_d)$ for which

$$\|p - x\| = \|q - x\| \iff \|p - x\|^2 = \|q - x\|^2 \iff \|p\|^2 - \|q\|^2 = 2(p - q)^\top x \,.$$

As p and q are distinct, this is the equation of a hyperplane. □

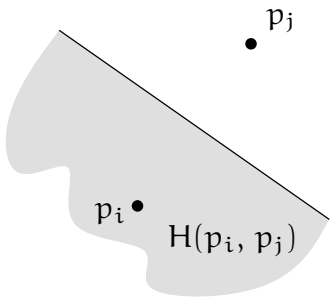


Figure 10.2: *The bisector of two points.*

Denote by $H(p_i, p_j)$ the closed halfspace bounded by the bisector of p_i and p_j that contains p_i. In $\mathbb{R}^2$, $H(p_i, p_j)$ is a halfplane; see Figure 10.2.

Exercise 10.2

a) What is the bisector of a line ℓ and a point $p \in \mathbb{R}^2 \setminus \ell$, that is, the set of all points $x \in \mathbb{R}^2$ with $\|x - p\| = \|x - \ell\|$ $(= \min_{q \in \ell} \|x - q\|)$?

b) For two points $p \neq q \in \mathbb{R}^2$, what is the region that contains all points whose distance to p is exactly twice their distance to q?

10.2 Voronoi Diagram

In the following we work with a set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^2$.

Definition 10.3 (Voronoi cell) *For $p_i \in P$ denote the* **Voronoi cell** $V_P(i)$ *of p_i by*

$$V_P(i) := \left\{ q \in \mathbb{R}^2 \;\middle|\; \|q - p_i\| \leqslant \|q - p\| \text{ for all } p \in P \right\}.$$

Proposition 10.4

$$V_P(i) = \bigcap_{j \neq i} H(p_i, p_j) \,.$$

Proof. For $j \neq i$ we have $\|q - p_i\| \leqslant \|q - p_j\| \iff q \in H(p_i, p_j)$. □

Corollary 10.5 *$V_P(i)$ is non-empty and convex.*

Proof. According to Proposition 10.4, $V_P(i)$ is the intersection of a finite number of halfplanes and hence convex. $V_P(i) \neq \emptyset$ because $p_i \in V_P(i)$. □

Observe that every point of the plane lies in some Voronoi cell but no point lies in the interior of two Voronoi cells. Therefore these cells form a subdivision of the plane. See Figure 10.3 for an example.

Definition 10.6 (Voronoi Diagram) *The* **Voronoi Diagram** $VD(P)$ *of a set* $P = \{p_1, \ldots, p_n\}$ *of points in* $\mathbb{R}^2$ *is the subdivision of the plane induced by the Voronoi cells* $V_P(i)$*, for* $i = 1, \ldots, n$*.*
Denote by $VV(P)$ *the set of vertices, by* $VE(P)$ *the set of edges, and by* $VR(P)$ *the set of regions (faces) of* $VD(P)$*.*

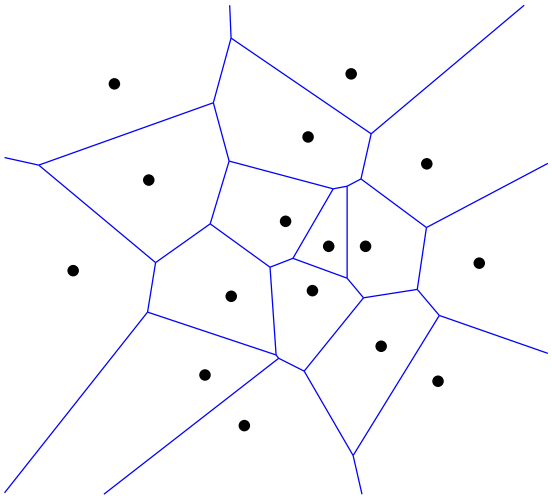


Figure 10.3: *Example: The Voronoi diagram of a point set.*

Lemma 10.7 *For every vertex* $v \in VV(P)$ *the following statements hold.*

a) v *is the common intersection of at least three edges from* $VE(P)$*;*

b) v *is incident to at least three regions from* $VR(P)$*;*

c) v *is the center of a circle* $C(v)$ *through at least three points from* P *such that*

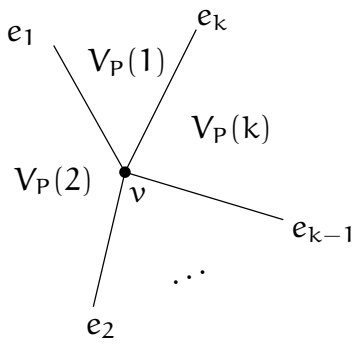
d) $C(v)^\circ \cap P = \emptyset$*.*

Proof. Consider a vertex $v \in VV(P)$. As all Voronoi cells are convex, $k \geqslant 3$ of them must be incident to v. This proves Part a) and b).

Without loss of generality let these cells be $V_P(i)$, for $1 \leqslant i \leqslant k$. Denote by e_i, $1 \leqslant i \leqslant k$, the edge incident to v that bounds $V_P(i)$ and $V_P((i \bmod k) + 1)$.

For any $i = 1, \ldots, k$ we have $v \in e_i \Rightarrow \|v - p_i\| = \|v - p_{(i \bmod k)+1}\|$. In other words, p_1, p_2, $\ldots$, p_k are cocircular, which proves Part c).

Part d): Suppose there exists a point $p_\ell \in C(v)^\circ$. Then the vertex v is closer to p_ℓ than it is to any of $p_1, \ldots, p_k$, in contradiction to the fact that v is contained in all of $V_P(1), \ldots, V_P(k)$. □



Corollary 10.8 *If P is in general position (no four points from P are cocircular), then for every vertex $v \in VV(P)$ the following statements hold.*

a) v is the common intersection of exactly three edges from $VE(P)$;

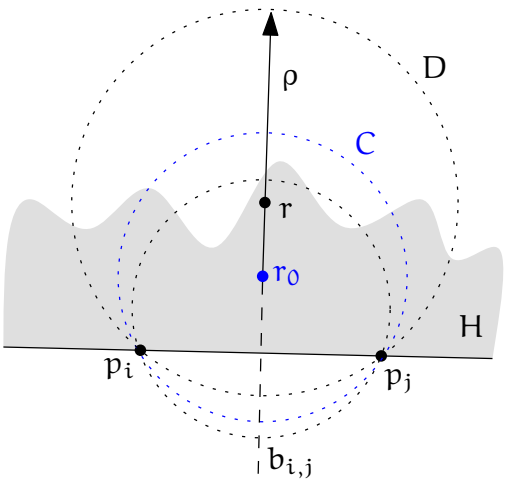
b) v is incident to exactly three regions from $VR(P)$;

c) v is the center of a circle $C(v)$ through exactly three points from P such that

d) $C(v)^\circ \cap P = \emptyset$. □

Lemma 10.9 *There is an unbounded Voronoi edge bounding $V_P(i)$ and $V_P(j)$ $\iff$ $\overline{p_ip_j} \cap P = \{p_i, p_j\}$ and $\overline{p_ip_j} \subseteq \partial\mathrm{conv}(P)$, where the latter denotes the boundary of the convex hull of P.*

Proof.
Denote by $b_{i,j}$ the bisector of p_i and p_j, and let $\mathcal{C}$ denote the family of circles centered at some point on $b_{i,j}$ and passing through p_i (and p_j). There is an unbounded Voronoi edge bounding $V_P(i)$ and $V_P(j)$ $\iff$ there is a ray $\rho \subset b_{i,j}$ such that $\|r - p_k\| > \|r - p_i\|$ $(= \|r - p_j\|)$, for every $r \in \rho$ and every $p_k \in P$ with $k \notin \{i, j\}$. Equivalently, there is a ray $\rho \subset b_{i,j}$ such that for every point $r \in \rho$ the circle $C \in \mathcal{C}$ centered at r does not contain any point from P in its interior.



The latter statement implies that the open halfplane H, whose bounding line passes through p_i and p_j and such that H contains the infinite part of ρ, contains no point from P in its interior. Therefore, $\overline{p_ip_j}$ appears on $\partial\mathrm{conv}(P)$ and $\overline{p_ip_j}$ does not contain any $p_k \in P$, for $k \neq i, j$.

Conversely, suppose that $\overline{p_ip_j}$ appears on $\partial\mathrm{conv}(P)$ and $\overline{p_ip_j} \cap P = \{p_i, p_j\}$. Then some halfplane H whose bounding line passes through p_i and p_j contains no point from

P in its interior. In particular, the existence of H together with $\overline{p_ip_j} \cap P = \{p_i, p_j\}$ implies that there is some circle $C \in \mathcal{C}$ such that $C \cap P = \{p_i, p_j\}$. Denote by r_0 the center of C and let ρ denote the ray starting from r_0 along $b_{i,j}$ such that the infinite part of ρ is contained in H. Consider any circle $D \in \mathcal{C}$ centered at a point $r \in \rho$ and observe that $D \setminus H \subseteq C \setminus H$. As neither H nor C contain any point from P in their respective interior, neither does D. This holds for every D, and we have seen above that this statement is equivalent to the existence of an unbounded Voronoi edge bounding $V_P(i)$ and $V_P(j)$. □

10.3 Duality

A *straight-line dual* of a plane graph G is a graph G' defined as follows: Choose a point for each face of G and connect any two such points by a straight edge, if the corresponding faces share an edge of G. Observe that this notion depends on the embedding; that is why the straight-line dual is defined for a plane graph rather than for an abstract graph. In general, G' may have edge crossings, which may also depend on the choice of representative points within the faces. However, for Voronoi diagrams is a particularly natural choice of representative points such that G' is plane: the points from P.

Theorem 10.10 (Delaunay [2]) *The straight-line dual of* $\mathrm{VD}(P)$ *for a set* $P \subset \mathbb{R}^2$ *of* $n \geqslant 3$ *points in general position (no three points from* P *are collinear and no four points from* P *are cocircular) is a triangulation: the unique Delaunay triangulation of* P*.*

Proof. By Lemma 10.9, the convex hull edges appear in the straight-line dual T of $\mathrm{VD}(P)$ and they correspond exactly to the unbounded edges of $\mathrm{VD}(P)$. All remaining edges of $\mathrm{VD}(P)$ are bounded, that is, both endpoints are Voronoi vertices. Consider some $v \in \mathrm{VV}(P)$. According to Corollary 10.8(b), v is incident to exactly three Voronoi regions, which, therefore, form a triangle $\triangle(v)$ in T. By Corollary 10.8(d), the circumcircle of $\triangle(v)$ does not contain any point from P in its interior. Hence $\triangle(v)$ appears in the (unique by Corollary 6.18) Delaunay triangulation of P.

Conversely, for any triangle $p_ip_jp_k$ in the Delaunay triangulation of P, by the empty circle property the circumcenter c of $p_ip_jp_k$ has p_i, p_j, and p_k as its closest points from P. Therefore, $c \in \mathrm{VV}(P)$ and—as above—the triangle $p_ip_jp_k$ appears in T. □

It is not hard to generalize Theorem 10.10 to general point sets. In this case, a Voronoi vertex of degree k is mapped to a convex polygon with k cocircular vertices. Any triangulation of such a polygon yields a Delaunay triangulation of the point set.

Corollary 10.11 $|\mathrm{VE}(P)| \leqslant 3n - 6$ *and* $|\mathrm{VV}(P)| \leqslant 2n - 5$*.*

Proof. Every edge in $\mathrm{VE}(P)$ corresponds to an edge in the dual Delaunay triangulation. The latter is a plane graph on n vertices and thus has at most $3n - 6$ edges and at most $2n - 4$ faces by Lemma 5.1. Only the bounded faces correspond to a vertex in $\mathrm{VD}(P)$. □

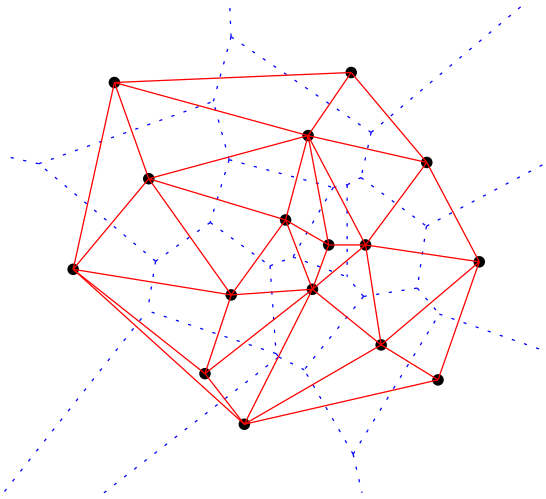


Figure 10.4: *The Voronoi diagram of a point set and its dual Delaunay triangulation.*

Corollary 10.12 *For a set* $P \subset \mathbb{R}^2$ *of* n *points, the Voronoi diagram of* P *can be constructed in expected* $O(n \log n)$ *time and space.*

Proof. We have seen that a Delaunay triangulation T for P can be obtained using randomized incremental construction in the given time and space bounds. As T is a plane graph, its number of vertices, edges, and faces all are linear in n. Therefore, the straight-line dual of T—which by Theorem 10.10 is the desired Voronoi diagram—can be computed in $O(n)$ additional time and space. □

Exercise 10.13 *Consider the Delaunay triangulation* T *for a set* $P \subset \mathbb{R}^2$ *of* $n \geqslant 3$ *points in general position. Prove or disprove:*

a) Every edge of T *intersects its dual Voronoi edge.*

b) Every vertex of $VD(P)$ *is contained in its dual Delaunay triangle.*

10.4 Lifting Map

Recall the lifting map that we used in Section 6.3 to prove that the Lawson Flip Algorithm terminates. Denote by $\mathcal{U} : z = x^2 + y^2$ the unit paraboloid in $\mathbb{R}^3$. The lifting map $\ell : \mathbb{R}^2 \to \mathcal{U}$ with $\ell : p = (p_x, p_y) \mapsto (p_x, p_y, {p_x}^2 + {p_y}^2)$ is the projection of the x/y-plane onto $\mathcal{U}$ in direction of the z-axis.

For $p \in \mathbb{R}^2$ let H_p denote the plane of tangency to $\mathcal{U}$ in $\ell(p)$. Denote by $h_p : \mathbb{R}^2 \to H_p$ the projection of the x/y-plane onto H_p in direction of the z-axis (see Figure 10.5).

Lemma 10.14 $\|\ell(q) - h_p(q)\| = \|p - q\|^2$, *for any points* $p, q \in \mathbb{R}^2$.

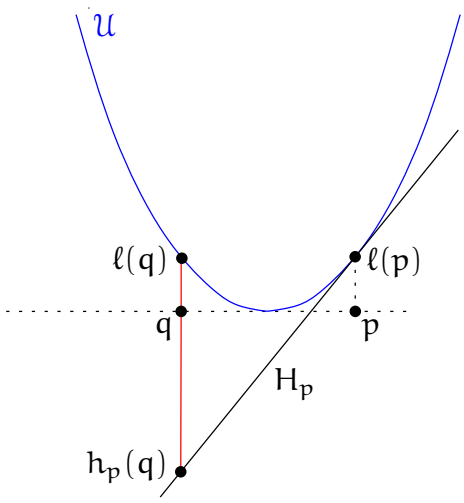


Figure 10.5: *Lifting map interpretation of the Voronoi diagram in a two-dimensional projection.*

Exercise 10.15 *Prove Lemma 10.14.* Hint: *First determine the equation of the tangent plane* H_p *to* $\mathcal{U}$ *in* $\ell(p)$.

Theorem 10.16 *For* $p = (p_x, p_y) \in \mathbb{R}^2$ *denote by* H_p *the plane of tangency to the unit paraboloid* $\mathcal{U} = \{(x, y, z) : z = x^2 + y^2\} \subset \mathbb{R}^3$ *in* $\ell(p) = (p_x, p_y, {p_x}^2 + {p_y}^2)$. *Let* $\mathcal{H}(P) := \bigcap_{p \in P} H_p^+$ *the intersection of all halfspaces above the planes* H_p, *for* $p \in P$. *Then the vertical projection of* $\partial\mathcal{H}(P)$ *onto the* x/y*-plane forms the Voronoi Diagram of* P *(the faces of* $\partial\mathcal{H}(P)$ *correspond to Voronoi regions, the edges to Voronoi edges, and the vertices to Voronoi vertices).*

Proof. For any point $q \in \mathbb{R}^2$, the vertical line through q intersects every plane H_p, $p \in P$. By Lemma 10.14 the topmost plane intersected belongs to the point from P that is closest to q. □

10.5 Point location in a Voronoi Diagram

One last bit is still missing in order to solve the post office problem optimally.

Theorem 10.17 *Given a triangulation* T *for a set* $P \subset \mathbb{R}^2$ *of* n *points, one can build in* $O(n)$ *time an* $O(n)$ *size data structure that allows for any query point* $q \in \mathrm{conv}(P)$ *to find in* $O(\log n)$ *time a triangle from* T *containing* q.

The data structure we will employ is known as *Kirkpatrick's hierarchy*. But before discussing it in detail, let us put things together in terms of the post office problem.

Corollary 10.18 (Nearest Neighbor Search) *Given a set* $P \subset \mathbb{R}^2$ *of* n *points, one can build in expected* $O(n \log n)$ *time an* $O(n)$ *size data structure that allows for any query*

point $q \in \mathrm{conv}(P)$ *to find in* $O(\log n)$ *time a nearest neighbor of* q *among the points from* P*.*

Proof. First construct the Voronoi Diagram V of P in expected $O(n \log n)$ time. It has exactly n convex faces. Every unbounded face can be cut by the convex hull boundary into a bounded and an unbounded part. As we are concerned with query points within $\mathrm{conv}(P)$ only, we can restrict our attention to the bounded parts.[1] Any convex polygon can easily be triangulated in time linear in its number of edges (= number of vertices). As V has at most $3n - 6$ edges and every edge appears in exactly two faces, V can be triangulated in $O(n)$ time overall. Label each of the resulting triangles with the point from p, whose Voronoi region contains it, and apply the data structure from Theorem 10.17. □

10.5.1 Kirkpatrick's Hierarchy

We will now the develop the data structure for point location in a triangulation, as described in Theorem 10.17. The main idea is to construct a hierarchy $T_0, \ldots, T_h$ of triangulations, such that

- $T_0 = T$,
- the vertices of T_i are a subset of the vertices of T_{i-1}, for $i = 1, \ldots, h$, and
- T_h comprises a single triangle only.

Search. For a query point x the triangle from T containing x can be found as follows.

Search$(x \in \mathbb{R}^2)$

1. For $i = h, h-1, \ldots, 0$: Find a triangle t_i from T_i that contains x.
2. return t_0.

This search is efficient under the following conditions.

(C1) Every triangle from T_i intersects only few ($\leqslant c$) triangles from T_{i-1}. (These will then be connected via the data structure.)

(C2) h is small ($\leqslant d \log n$).

Proposition 10.19 *The search procedure described above needs* $\leqslant 3cd \log n = O(\log n)$ *orientation tests.*

Proof. For every T_i, $0 \leqslant i < h$, at most c triangles are tested as to whether or not they contain x. Using three orientation tests one can determine whether or not a triangle contains a given point. □

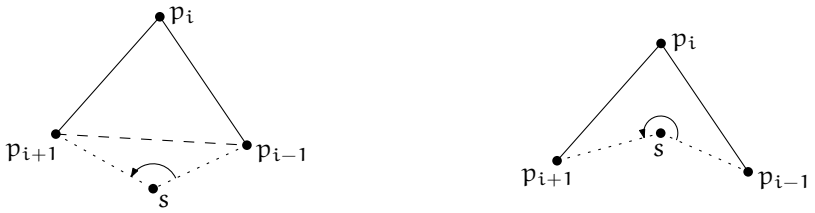
[1] We even know how to decide in $O(\log n)$ time whether or not a given point lies within $\mathrm{conv}(P)$, see Exercise 3.20.

Thinning. Removing a vertex v and all its incident edges from a triangulation creates a non-triangulated hole that forms a star-shaped polygon since all points are visible from v (the star-point).

Lemma 10.20 *A star-shaped polygon, given as a sequence of $n \geqslant 3$ vertices and a star-point, can be triangulated in $O(n)$ time.*

Proof. For $n = 3$ there is nothing to do. For $n > 3$, consider a star-shaped polygon $P = (p_0, \ldots, p_{n-1})$ and a star-point s of P. Consider some convex vertex p_i of P, that is, $\angle p_{i+1}p_ip_{i-1} \leqslant \pi$, all indices taken $\bmod n$. (Every simple polygon on $n \geqslant 3$ vertices has at least three convex vertices, on the convex hull.)

As P is star-shaped, the quadrilateral $Q = p_{i-1}p_ip_{i+1}s$ is completely contained in P. Therefore, if $\angle p_{i-1}sp_{i+1} \leqslant \pi$ and hence Q is convex (Figure 10.6a), then we can add $p_{i-1}p_{i+1}$ as a diagonal. In this way one triangle is cut off from P, leaving a star-shaped polygon P' (with respect to s) on $n-1$ vertices. The polygon P' can then be triangulated recursively. If, on the other hand, $\angle p_{i-1}sp_{i+1} > \pi$ (Figure 10.6b), we cannot safely add



(a) $\angle p_{i-1}sp_{i+1} \leqslant \pi$. (b) $\angle p_{i-1}sp_{i+1} > \pi$.

Figure 10.6: *The quadrilateral $p_{i-1}p_ip_{i+1}s$ is contained in P.*

the edge $p_{i-1}p_{i+1}$ because it might intersect other edges of P or even lie outside of P. But we claim that in this case there exists another convex vertex p_j of P, for which $\angle p_{j-1}sp_{j+1} \leqslant \pi$ and therefore we can add the edge $p_{j-1}p_{j+1}$ instead.

In fact, it is enough to choose p_j to be some convex vertex of P that is not a neighbor of p_i: As $\sum_{k=0}^{n-1} \angle p_{k-1}sp_k = 2\pi$ and $\angle p_{i-1}sp_i + \angle p_isp_{i+1} = \angle p_{i-1}sp_{i+1} > \pi$, we have $\angle p_{j-1}sp_j + \angle p_jsp_{j+1} = \angle p_{j-1}sp_{j+1} < \pi$.

It remains to show that such a vertex p_j exists. P has at least three convex vertices. One of them is p_i. If the only other two convex vertices are p_{i-1} and p_{i+1}, then we make the whole argument with p_{i-1} instead of p_i and find $j = i + 1$. Note that p_{i-1} and p_{i+1} are not neighbors because P is not a triangle.

As for the linear time bound, we simply scan the sequence of vertices as in Graham's Scan. For every triple $p_{i-1}p_ip_{i+1}$ of successive vertices it is tested (in constant time) whether $p_{i-1}p_ip_{i+1}s$ forms a convex quadrilateral. If so, then the edge $p_{i-1}p_{i+1}$ is added, effectively removing p_i from further consideration. Therefore, p_i can be charged for the (potential, if there are enough vertices left) additional test of the new triple $p_xp_{i-1}p_{i+1}$ formed with the predecessor p_x of p_{i-1} in the current sequence. As shown for Graham's

Scan in Theorem 3.23, this results in a linear time algorithm overall.[2] □

As a side remark, the *kernel* of a simple polygon, that is, the set of all star-points, can be constructed in linear time as well. We will get back to this question later in the course...

Our working plan is to obtain T_i from T_{i-1} by removing several *independent* (pairwise non-adjacent) vertices and re-triangulating. These vertices should

a) have small degree (otherwise the degree within the hierarchy gets too large, that is, we need to test too many triangles on the next level) and

b) be many (otherwise the height h of the hierarchy gets too large).

The following lemma asserts the existence of a sufficiently large set of independent small-degree vertices in every triangulation.

Lemma 10.21 *In every triangulation of n points in $\mathbb{R}^2$ there exists an independent set of at least $\lceil n/18 \rceil$ vertices of maximum degree 8. Moreover, such a set can be found in $O(n)$ time.*

Proof. Let $T = (V, E)$ denote the graph of the triangulation, which we consider as an abstract graph in the following. We may suppose that T is maximally planar, that is, all faces are triangles. (Otherwise triangulate the exterior face arbitrarily. An independent set in the resulting graph is also independent in T.) For $n = 3$ the statement is true. Let $n \geqslant 4$.

By the Euler formula we have $|E| = 3n - 6$, that is,

$$\sum_{v \in V} \deg_T(v) = 2|E| = 6n - 12 < 6n.$$

Let $W \subseteq V$ denote the set of vertices of degree at most 8. Claim: $|W| > n/2$. Suppose $|W| \leqslant n/2$. As every vertex has degree at least three, we have

$$\begin{aligned} \sum_{v \in V} \deg_T(v) &= \sum_{v \in W} \deg_T(v) + \sum_{v \in V \setminus W} \deg_T(v) \geqslant 3|W| + 9|V \setminus W| \\ &= 3|W| + 9(n - |W|) = 9n - 6|W| \geqslant 9n - 3n = 6n, \end{aligned}$$

in contradiction to the above.

Construct an independent set U in T as follows (greedily): As long as $W \neq \emptyset$, add an arbitrary vertex $v \in W$ to U and remove v and all its neighbors from W.

Obviously U is independent and all vertices in U have degree at most 8. At each selection step at most 9 vertices are removed from W. Therefore $|U| \geqslant \lceil (n/2)/9 \rceil = \lceil n/18 \rceil$. □

Proof. (of Theorem 10.17)
Construct the hierarchy $T_0, \ldots T_h$ with $T_0 = T$ as follows. Obtain T_i from T_{i-1} by removing

[2] Recall that the $O(n \log n)$ time bound for Graham's Scan was caused by the initial sorting only.

an independent set U as in Lemma 10.21 and re-triangulating the resulting holes. By Lemma 10.20 and Lemma 10.21 every step is linear in the number $|T_i|$ of vertices in T_i. The total cost for building the data structure is thus

$$\sum_{i=0}^{h} \alpha|T_i| \leqslant \sum_{i=0}^{h} \alpha n(17/18)^i < \alpha n \sum_{i=0}^{\infty} (17/18)^i = 18\alpha n \in O(n),$$

for some constant α. Similarly the space consumption is linear.

The number of levels amounts to $h = \log_{18/17} n < 12.2 \log n$. Thus by Proposition 10.19 the search needs at most $3 \cdot 8 \cdot \log_{18/17} n < 292 \log n$ orientation tests. □

Improvements. As the name suggests, the hierarchical approach discussed above is due to David Kirkpatrick [5]. The constant 292 that appears in the search time is somewhat large. There has been a whole line of research trying to improve it using different techniques.

- Sarnak and Tarjan [6]: $4 \log n$.
- Edelsbrunner, Guibas, and Stolfi [3]: $3 \log n$.
- Goodrich, Orletsky, and Ramaiyer [4]: $2 \log n$.
- Adamy and Seidel [1]: $1 \log n + 2\sqrt{\log n} + O(\sqrt[4]{\log n})$.

Exercise 10.22 *Let $\{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane, which we call* obstacles*. Imagine there is a disk of radius r centered at the origin which can be moved around the obstacles but is not allowed to intersect them (touching the boundary is ok). Is it possible to move the disk out of these obstacles? See the example depicted in Figure 10.7 below.*

More formally, the question is whether there is a (continuous) path $\gamma : [0, 1] \longrightarrow \mathbb{R}^2$ with $\gamma(0) = (0, 0)$ and $\|\gamma(1)\| \geqslant \max\{\|p_1\|, \ldots, \|p_n\|\}$, such that at any time $t \in [0, 1]$ and $\|\gamma(t) - p_i\| \geqslant r$, for any $1 \leqslant i \leqslant n$. Describe an algorithm to decide this question and to construct such a path—if one exists—given arbitrary points $\{p_1, p_2, \ldots, p_n\}$ and a radius $r > 0$. Argue why your algorithm is correct and analyze its running time.

Exercise 10.23 *This exercise is about an application from* Computational Biology*: You are given a set of disks $P = \{a_1, .., a_n\}$ in $\mathbb{R}^2$, all with the same radius $r_a > 0$. Each of these disks represents an atom of a protein. A water molecule is represented by a disc with radius $r_w > r_a$. A water molecule cannot intersect the interior of any protein atom, but it can be tangent to one. We say that an atom $a_i \in P$ is* accessible *if there exists a placement of a water molecule such that it is tangent to a_i and does not intersect the interior of any other atom in P. Given P, find an $O(n \log n)$ time algorithm which determines all atoms of P that are inaccessible.*

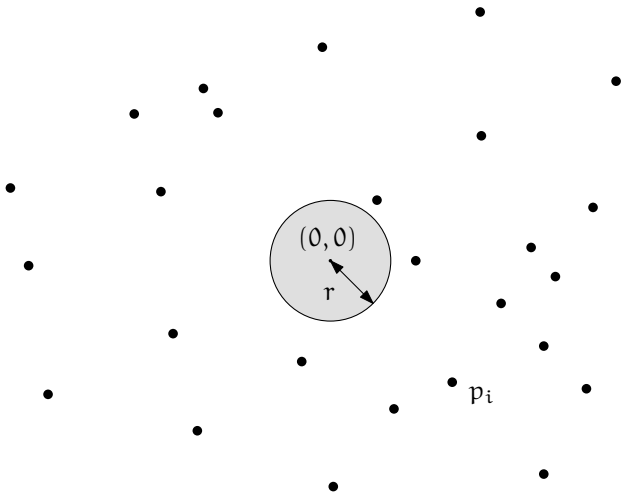


Figure 10.7: *Motion planning: Illustration for Exercise 10.22.*

Exercise 10.24 *Let* $P \subset \mathbb{R}^2$ *be a set of* n *points. Describe a data structure to find in* $O(\log n)$ *time a point in* P *that is furthest from a given query point* q *among all points in* P*.*

Exercise 10.25 *Show that the bounds given in Theorem 10.17 are optimal in the algebraic computation tree model.*

Questions

45. *What is the Voronoi diagram of a set of points in* $\mathbb{R}^2$*?* Give a precise definition and explain/prove the basic properties: convexity of cells, why is it a subdivision of the plane?, Lemma 10.7, Lemma 10.9.

46. *What is the correspondence between the Voronoi diagram and the Delaunay triangulation for a set of points in* $\mathbb{R}^2$*?* Prove duality (Theorem 10.10) and explain where general position is needed.

47. *How to construct the Voronoi diagram of a set of points in* $\mathbb{R}^2$*?* Describe an $O(n \log n)$ time algorithm, for instance, via Delaunay triangulation.

48. *How can the Voronoi diagram be interpreted in context of the lifting map?* Describe the transformation and prove its properties to obtain a formulation of the Voronoi diagram as an intersection of halfspaces one dimension higher.

49. *What is the Post-Office Problem and how can it be solved optimally?* Describe the problem and a solution using linear space, $O(n \log n)$ preprocessing, and $O(\log n)$ query time.

50. *How does Kirkpatrick's hierarchical data structure for planar point location work exactly?* Describe how to build it and how the search works, and prove the runtime bounds. In particular, you should be able to state and prove Lemma 10.20, Lemma 10.21, and Theorem 10.17.

References

[1] Udo Adamy and Raimund Seidel, On the exaxt worst case query complexity of planar point location. *J. Algorithms*, **37**, (2000), 189–217, URL http://dx.doi.org/10.1006/jagm.2000.1101.

[2] Boris Delaunay, Sur la sphère vide. A la memoire de Georges Voronoi. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, **7**, (1934), 793–800.

[3] Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi, Optimal point location in a monotone subdivision. *SIAM J. Comput.*, **15**, 2, (1986), 317–340, URL http://dx.doi.org/10.1137/0215023.

[4] Michael T. Goodrich, Mark W. Orletsky, and Kumar Ramaiyer, Methods for achieving fast query times in point location data structures. In *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, pp. 757–766, 1997, URL http://doi.acm.org/10.1145/314161.314438.

[5] David G. Kirkpatrick, Optimal search in planar subdivisions. *SIAM J. Comput.*, **12**, 1, (1983), 28–35, URL http://dx.doi.org/10.1137/0212002.

[6] Neil Sarnak and Robert E. Tarjan, Planar point location using persistent search trees. *Commun. ACM*, **29**, 7, (1986), 669–679, URL http://dx.doi.org/10.1145/6138.6151.

Chapter 11

Linear Programming

This lecture is about a special type of optimization problems, namely *linear programs*. We start with a geometric problem that can directly be formulated as a linear program.

11.1 Linear Separability of Point Sets

Let $P \subseteq \mathbb{R}^d$ and $Q \subseteq \mathbb{R}^d$ be two finite point sets in d-dimensional space. We want to know whether there exists a hyperplane that separates P from Q (we allow non-strict separation, i.e. some points are allowed to be on the hyperplane). Figure 11.1 illustrates the 2-dimensional case.

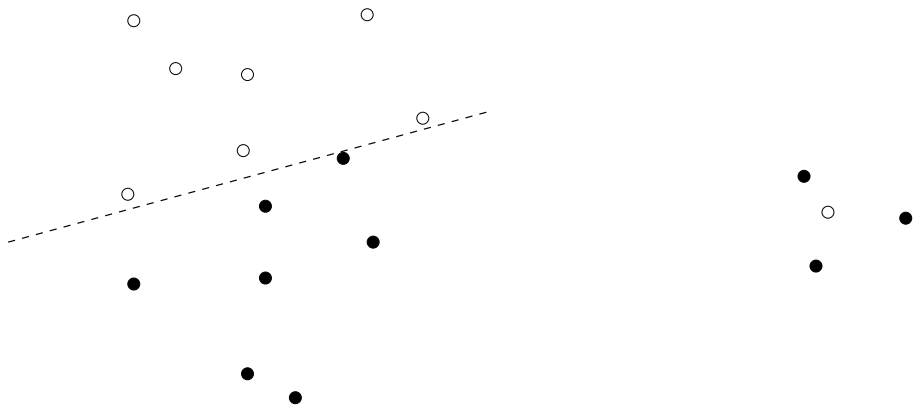


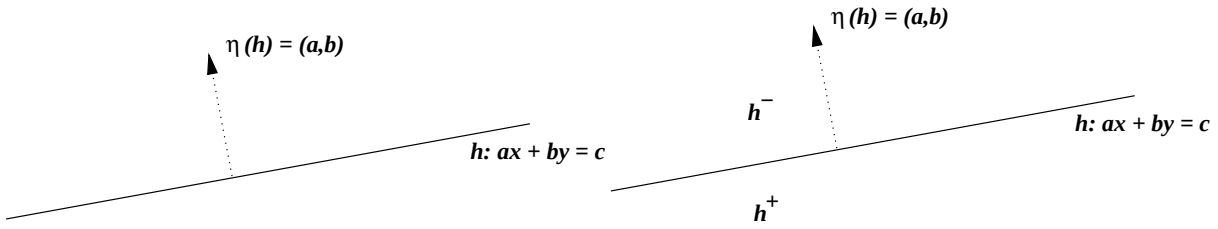
Figure 11.1: *Left: there is a separating hyperplane; Right: there is no separating hyperplane*

How can we formalize this problem? A hyperplane is a set of the form

$$h = \{x \in \mathbb{R}^d : h_1 x_1 + h_2 x_2 + \cdots + h_d x_d = h_0\},$$

where $h_i \in \mathbb{R}, i = 0, \ldots, d$. For example, a line in the plane has an equation of the form $ax + by = c$.

The vector $\eta(h) = (h_1, h_2, \ldots, h_d) \in \mathbb{R}^d$ is called the *normal vector* of h. It is orthogonal to the hyperplane and usually visualized as in Figure 11.2(a).



(a) The normal vector of a hyperplane **(b)** The two halfspaces of a hyperplane

Figure 11.2: *The concepts of hyperplane, normal vector, and halfspace*

Every hyperplane h defines two closed *halfspaces*

$$\begin{aligned} h^+ &= \{x \in \mathbb{R}^d : h_1x_1 + h_2x_2 + \cdots + h_dx_d \leqslant h_0\}, \\ h^- &= \{x \in \mathbb{R}^d : h_1x_1 + h_2x_2 + \cdots + h_dx_d \geqslant h_0\}. \end{aligned}$$

Each of the two halfpsaces is the region of space "on one side" of h (including h itself). The normal vector $\eta(h)$ points into h^-, see Figure 11.2(b). Now we can formally define linear separability.

Definition 11.1 *Two point sets $P \subseteq \mathbb{R}^d$ and $Q \subseteq \mathbb{R}^d$ are called* linearly separable *if there exists a hyperplane h such that $P \subseteq h^+$ and $Q \subseteq h^-$. In formulas, if there exist real numbers $h_0, h_1, \ldots, h_d$ such that*

$$\begin{aligned} h_1p_1 + h_2p_2 + \cdots + h_dp_d &\leqslant h_0, \quad p \in P, \\ h_1q_1 + h_2q_2 + \cdots + h_dq_d &\geqslant h_0, \quad q \in Q. \end{aligned}$$

As we see from Figure 11.1, such $h_0, h_1, \ldots, h_d$ may or may not exist. How can we find out?

11.2 Linear Programming

The problem of testing for linear separability of point sets is a special case of the general linear programming problem.

Definition 11.2 *Given $n, d \in \mathbb{N}$ and real numbers*

$$\begin{aligned} b_i &, \quad i = 1, \ldots, n, \\ c_j &, \quad j = 1, \ldots, d, \\ a_{ij} &, \quad i = 1, \ldots, n, \quad j = 1, \ldots, d, \end{aligned}$$

the linear program *defined by these numbers is the problem of finding real numbers* $x_1, x_2, \ldots, x_d$ *such that*

(i) $\sum_{j=1}^{d} a_{ij} x_j \leqslant b_i, \quad i = 1, \ldots, n$, *and*

(ii) $\sum_{j=1}^{d} c_j x_j$ *is as large as possible subject to (i).*

Let us get a geometric intuition: each of the n *constraints* in (i) requires $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ to be in the positive halfspace of some hyperplane. The intersection of all these halfspaces is the *feasible region* of the linear program. If it is empty, there is no solution—the linear program is called *infeasible*.

Otherwise—and now (ii) enters the picture—we are looking for a *feasible solution* x (a point inside the feasible region) that maximizes $\sum_{j=1}^{d} c_j x_j$. For every possible value γ of this sum, the feasible solutions for which the sum attains this value are contained in the hyperplane

$$\{x \in \mathbb{R}^d : \sum_{j=1}^{d} c_j x_j = \gamma\}$$

with normal vector $c = (c_1, \ldots, c_d)$. Increasing γ means to shift the hyperplane into direction c. The highest γ is thus obtained from the hyperplane that is most extreme in direction c among all hyperplanes that intersect the feasible region, see Figure 11.3.

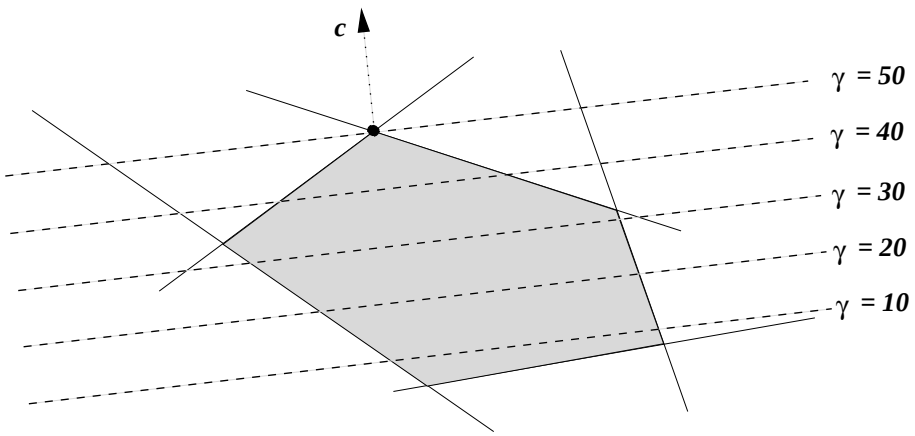


Figure 11.3: *A linear program: finding the feasible solution in the intersection of five positive halfspaces that is most extreme in direction* c *(has highest value* $\gamma = \sum_{j=1}^{d} c_j x_j$*)*

In Figure 11.3, we do have an *optimal solution* (a feasible solution x of highest value $\sum_{j=1}^{d} c_j x_j$), but in general, there might be feasible solutions of arbitrarily high γ-value. In this case, the linear program is called *unbounded*, see Figure 11.4.

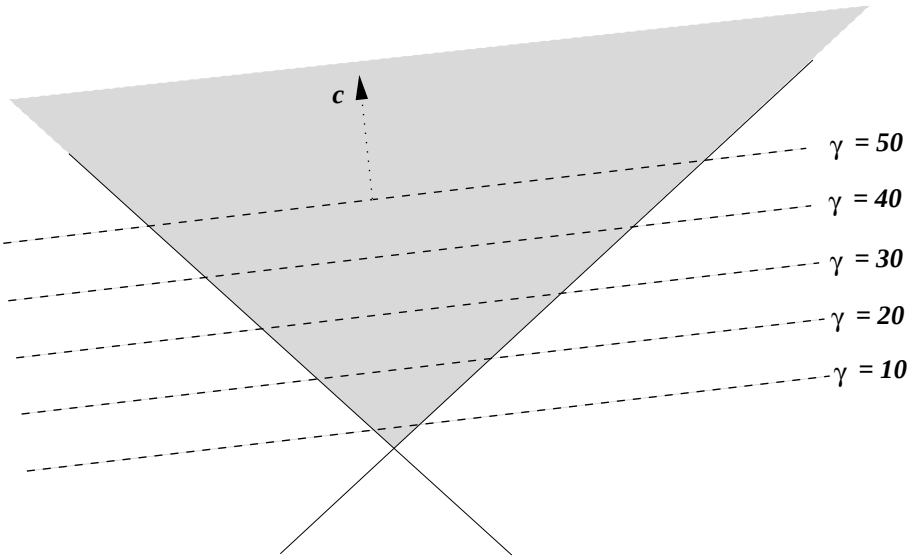


Figure 11.4: *An unbounded linear program*

It can be shown that infeasibility and unboundedness are the only obstacles for the existence of an optimal solution. If the linear program is feasible and bounded, there exists an optimal solution.

This is not entirely trivial, though. To appreciate the statement, consider the problem of finding a point (x, y) that (i) satisfies $y \geqslant e^x$ and (ii) has smallest value of y among all (x, y) that satisfy (i). This is not a linear program, but in the above sense it is feasible (there are (x, y) that satisfy (i)) and bounded (y is bounded below from 0 over the set of feasible solutions). Still, there is no optimal solution, since values of y arbitrarily close to 0 can be attained but not 0 itself.

Even if a linear program has an optimal solution, it is in general not unique. For example, if you rotate c in Figure 11.3 such that it becomes orthogonal to the top-right edge of the feasible region, then every point of this edge is an optimal solution. Why is this called a *linear* program? Because all constraints are linear inequalities, and the objective function is a linear function. There is also a reason why it is called a linear *program*, but we won't get into this here (see [3] for more background).

Using vector and matrix notation, a linear program can succinctly be written as follows.

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & c^T x \\ & \text{subject to} & Ax \leqslant b \end{array}$$

Here, $c, x \in \mathbb{R}^d$, $b \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times d}$, and $\cdot^T$ denotes the transpose operation. The inequality "$\leqslant$" is interpreted componentwise. The vector x represents the *variables*, c is called the *objective function vector*, b the *right-hand side*, and A the *constraint matrix*.

To *solve* a linear programs means to either report that the problem is infeasible or unbounded, or to compute an optimal solution x^*. If we can solve linear programs, we can also decide linear separability of point sets. For this, we check whether the linear program

$$\begin{array}{ll} \text{maximize} & 0 \\ \text{subject to} & h_1 p_1 + h_2 p_2 + \cdots + h_d p_d - h_0 \leqslant 0, \quad p \in P, \\ & h_1 q_1 + h_2 q_2 + \cdots + h_d q_d - h_0 \geqslant 0, \quad q \in Q. \end{array}$$

in the $d+1$ variables $h_0, h_1, h_2, \ldots, h_d$ and objective function vector $c = 0$ is feasible or not. The fact that some inequalities are of the form "$\geqslant$" is no problem, of course, since we can multiply an inequality by -1 to turn "$\geqslant$" into "$\leqslant$".

11.3 Minimum-area Enclosing Annulus

Here is another geometric problem that we can write as a linear program, although this is less obvious. Given a point set $P \subseteq \mathbb{R}^2$, find a *minimum-area annulus* (region between two concentric circles) that contains P; see Figure 11.5 for an illustration.

The optimal annulus can be used to test whether the point set P is (approximately) on a common circle which is the case if the annulus has zero (or small) area.

Let us write this as an optimization problem in the variables $c = (c_1, c_2) \in \mathbb{R}^2$ (the center) and $r, R \in \mathbb{R}$ (the small and the large radius).

$$\begin{array}{ll} \text{minimize} & \pi(R^2 - r^2) \\ \text{subject to} & r^2 \leqslant \|p - c\|^2 \leqslant R^2, \quad p \in P. \end{array}$$

This neither has a linear objective function nor are the constraints linear inequalities. But a variable substitution will take care of this. We define new variables

$$u := r^2 - \|c\|^2, \tag{11.3}$$

$$v := R^2 - \|c\|^2. \tag{11.4}$$

Omitting the factor π in the objective function does not affect the optimal solution (only its value), hence we can equivalently work with the objective function $v - u = R^2 - r^2$. The constraint $r^2 \leqslant \|p - c\|^2$ is equivalent to $r^2 \leqslant \|p\|^2 - 2p^T c + \|c\|^2$, or

$$u + 2p^T c \leqslant \|p\|^2.$$

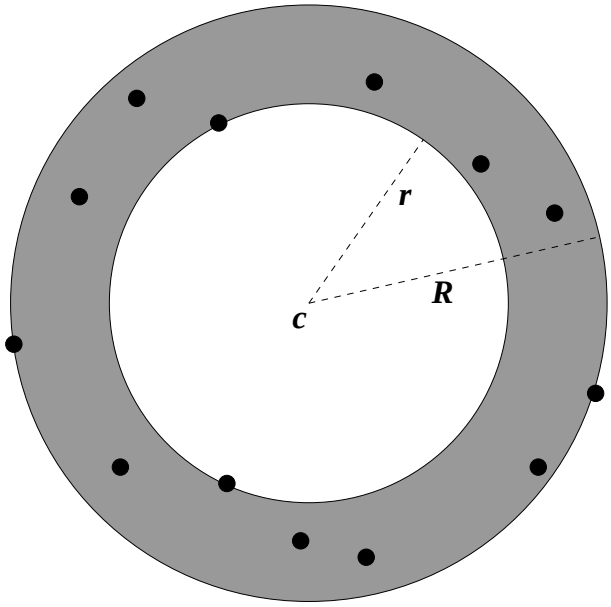


Figure 11.5: *A minimum-area annulus containing* P

In the same way, $\|p - c\| \leqslant R$ turns out to be equivalent to

$$v + 2p^T c \geqslant \|p\|^2.$$

This means, we now have a *linear* program in the variables u, v, c_1, c_2:

$$\begin{array}{ll} \text{maximize} & u - v \\ \text{subject to} & u + 2p^T c \leqslant \|p\|^2, \quad p \in P \\ & v + 2p^T c \geqslant \|p\|^2, \quad p \in P. \end{array}$$

From optimal values for u, v and c, we can also reconstruct r^2 and R^2 via (11.3) and (11.4). It cannot happen that r^2 obtained in this way is negative: since we have $r^2 \leqslant \|p - c\|^2$ for all p, we could still increase u (and hence r^2 to at least 0), which is a contradicition to $u - v$ being maximal.

Exercise 11.5 *a) Prove that the problem of finding a largest disk inside a convex polygon can be formulated as a linear program! What is the number of variables in your linear program?*

b) Prove that the problem of testing whether a simple polygon is starshaped can be formulated as a linear program.

Exercise 11.6 *Given a simple polygon* P *as a list of vertices along its boundary. Describe a linear time algorithm to decide whether* P *is star-shaped and—if so—to construct the so-called* kernel *of* P*, that is, the set of all star-points.*

11.4 Solving a Linear Program

Linear programming was first studied in the 1930's -1950's, and some of its original applications were of a military nature. In the 1950's, Dantzig invented the *simplex method* for solving linear programs, a method that is fast in practice but is not known to come with any theoretical guarantees [1].

The computational complexity of solving linear programs was unresolved until 1979 when Leonid Khachiyan discovered a polynomial-time algorithm known as the *ellipsoid method* [2]. This result even made it into the New York times.

From a computational geometry point of view, linear programs with a *fixed* number of variables are of particular interest (see our two applications above, with d and 4 variables, respectively, where d may be 2 or 3 in some relavant cases). As was first shown by Megiddo, such linear programs can be solved in time $O(n)$, where n is the number of constraints [4]. In the next lecture, we will describe a much simpler randomized $O(n)$ algorithm due to Seidel [5].

Questions

51. *What is a linear program?* Give a precise definition! How can you visualize a linear program? What does it mean that the linear program is infeasible / unbounded?

52. *Show an application of linear programming!* Describe a geometric problem that can be formulated as a linear program, and give that formulation!

References

[1] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[2] L. G. Khachiyan, Polynomial algorithms in linear programming, *U.S.S.R. Comput. Math. and Math. Phys* **20** (1980), 53–72.

[3] J. Matoušek and B. Gärtner, *Understanding and Using Linear Programming*, Universitext, Springer-Verlag, 2007.

[4] N. Megiddo, Linear programming in linear time when the dimension is fixed, *J. ACM* **31** (1984), 114–127.

[5] R. Seidel, Small-dimensional linear programming and convex hulls made easy, *Discrete Comput. Geom.* **6** (1991), 423–434.

Chapter 12

A randomized Algorithm for Linear Programming

Let us recall the setup from last lecture: we have a linear program of the form

$$\begin{array}{lll} \text{(LP)} & \text{maximize} & c^T x \\ & \text{subject to} & Ax \leqslant b, \end{array} \tag{12.1}$$

where $c, x \in \mathbb{R}^d$ (there are d variables), $b \in \mathbb{R}^n$ (there are n constraints), and $A \in \mathbb{R}^{n \times d}$. The scenario that we are interested in here is that d is a (small) constant, while n tends to infinity.

The goal of this lecture is to present a randomized algorithm (due to Seidel [2]) for solving a linear program whose expected runtime is $O(n)$. The constant behind the big-O will depend exponentially on d, meaning that this algorithm is practical only for small values of d.

To prepare the ground, let us first get rid of the unboundedness issue. We add to our linear program a set of $2d$ constraints

$$-M \leqslant x_i \leqslant M, \quad i = 1, 2, \ldots d, \tag{12.2}$$

where M is a symbolic constant assumed to be larger than any real number that it is compared with. Formally, this can be done by computing with rational functions in M (quotients of polynomials of degree d in the "variable "M), rather than real numbers. The original problem is bounded if and only if the solution of the new (and bounded) problem does not depend on M. This is called the *big-M method*.

Now let H, $|H| = n$, denote the set of original constraints. For $h \in H$, we write the corresponding constraint as $a_h x \leqslant b_h$.

Definition 12.3 *For $Q, R \subseteq H$, $Q \cap R = \emptyset$, let $x^*(Q, R)$ denote the lexicographically largest optimal solution of the linear program*

$$\begin{array}{lll} \textit{LP(Q,R)} & \textit{maximize} & c^T x \\ & \textit{subject to} & a_h x \leqslant b_h, \qquad h \in Q \\ & & a_h x = b_h, \qquad h \in R \\ & & -M \leqslant x_i \leqslant M, \quad i = 1, 2, \ldots, d. \end{array}$$

If this linear program has no feasible solution, we set $x^*(Q, R) = \infty$.

What does this mean? We delete some of the original constraints (the ones not in $Q \cup R$, and we require some of the constraints to hold with equality (the ones in R). Since every linear equation $a_h x = b_h$ can be simulated by two linear inequalities $a_h x \leqslant b_h$ and $a_h x \geqslant b_h$, this again assumes the form of a linear program. By the big-M method, this linear program is bounded, but it may be infeasible. If it is feasible, there may be several optimal solutions, but choosing the lexicographically largest one leads to a unique solution $x^*(Q, R)$.

Our algorithm will compute $x^*(H, \emptyset)$, the lexicographically largest optimal solution of (12.1) subject to the additional bounding-box constraint (12.2). We also assume that $x^*(H, \emptyset) \neq \infty$, meaning that (12.1) is feasible. At the expense of solving an auxiliary problem with one extra variable, this may be assumed w.l.o.g. (Exercise).

Exercise 12.4 *Suppose that you have an algorithm $\mathcal{A}$ for solving* feasible *linear programs of the form*

$$\begin{array}{lll} \textit{(LP)} & \textit{maximize} & c^T x \\ & \textit{subject to} & Ax \leqslant b, \end{array}$$

where feasible means that there exists $\tilde{x} \in \mathbb{R}^d$ such that $A\tilde{x} \leqslant b$. Extend algorithm $\mathcal{A}$ such that it can deal with arbitrary (not necessarily feasible) linear programs of the above form.

12.1 Helly's Theorem

A crucial ingredient of the algorithm's analysis is that the optimal solution $x^*(H, \emptyset)$ is already determined by a constant number (at most d) of constraints. More generally, the following holds.

Lemma 12.5 *Let $Q, R \subseteq H, Q \cap R = \emptyset$, such that the constraints in R are* independent. *This means that the set $\{x \in \mathbb{R}^d : a_h x = b_h, h \in R\}$ has dimension $d - |R|$.*

If $x^*(Q, R) \neq \infty$, then there exists $S \subseteq Q, |S| \leqslant d - |R|$ such that

$$x^*(S, R) = x^*(Q, R).$$

The proof uses *Helly's Theorem*, a classic result in convexity theory.

Theorem 12.6 (Helly's Theorem [1]) *Let $C_1, \ldots C_n$ be $n \geqslant d + 1$ convex subsets of $\mathbb{R}^d$. If any $d + 1$ of the sets have a nonempty common intersection, then the common intersection of* all n *sets is nonempty.*

Even in $\mathbb{R}^1$, this is not entirely obvious. There it says that for every set of intervals with pairwise nonempty overlap there is one point contained in all the intervals. We will not prove Helly's Theorem here but just use it to prove Lemma 12.5.

Proof. (**Lemma 12.5**) The statement is trivial for $|Q| \leqslant d - |R|$, so assume $|Q| > d - |R|$. Let

$$L(R) := \{x \in \mathbb{R}^d : a_h x = b_h, h \in R\}$$

and

$$B := \{x \in \mathbb{R}^d : -M \leqslant x_i \leqslant M, i = 1, \ldots, d\}.$$

For a vector $x = (x_1, \ldots, x_d)$, we define $x_0 = c^T x$, and we write $x > x'$ if $(x_0, x_1, \ldots, x_d)$ is lexicographically larger than $(x'_0, x'_1, \ldots, x'_d)$.

Let $x^* = x^*(Q, R)$ and consider the $|Q| + 1$ sets

$$C_h = \{x \in \mathbb{R}^d : a_h x \leqslant b_h\} \cap B \cap L(R), \quad h \in Q$$

and

$$C_0 = \{x \in \mathbb{R}^d : x > x^*\} \cap B \cap L(R).$$

The first observation (that may require a little thinking in case of C_0) is that all these sets are convex. The second observation is that their common intersection is empty. Indeed, any point in the common intersection would be a feasible solution $\tilde{x}$ of $LP(Q, R)$ with $\tilde{x} > x^* = x^*(Q, R)$, a contradiction to $x^*(Q, R)$ being the lexicographically largest optimal solution of $LP(Q, R)$. The third observation is that since $L(R)$ has dimension $d - |R|$, we can after an affine transformation assume that all our $|Q| + 1$ convex sets are actually convex subsets of $\mathbb{R}^{d-|R|}$.

Then, applying Helly's Theorem yields a subset of $d - |R| + 1$ constraints with an empty common intersection. Since all the C_h do have $x^*(Q, R)$ in common, this set of constraints must contain C_0. This means, there is $S \subseteq Q, |S| = d - |R|$ such that

$$x \in C_h \ \forall h \in S \quad \Rightarrow \quad x \notin C_0.$$

In particular, $x^*(S, R) \in C_h$ for all $h \in S$, and so it follows that $x^*(S, R) \leqslant x^* = x^*(Q, R)$. But since $S \subseteq Q$, we also have $x^*(S, R) \geqslant x^*(Q, R)$, and $x^*(S, R) = x^*(Q, R)$ follows. □

12.2 Convexity, once more

We need a second property of linear programs on top of Lemma 12.5; it is also a consequence of convexity of the constraints, but a much simpler one.

Lemma 12.7 *Let* $Q, R \subseteq H, Q \cap R \neq \emptyset$ *and* $x^*(Q, R) \neq \infty$. *Let* $h \in Q$. *If*

$$a_h x^*(Q \setminus \{h\}, R) > b_h,$$

then

$$x^*(Q, R) = x^*(Q \setminus \{h\}, R \cup \{h\}).$$

Before we prove this, let us get an intuition. The vector $x^*(Q \setminus \{h\}, R)$ is the optimal solution of $LP(Q \setminus \{h\}, R)$. And the inequality $a_h x^*(Q \setminus \{h\}, R) > b_h$ means that the constraint h is violated by this solution. The implication of the lemma is that at the optimal solution of $LP(Q, R)$, the constraint h must be satisfied with equality in which case this optimal solution is at the same time the optimal solution of the more restricted problem $LP(Q \setminus \{h\}, R \cup \{h\})$.

Proof. Let us suppose for a contradition that

$$a_h x^*(Q, R) < b_h$$

and consider the line segment s that connects $x^*(Q, R)$ with $x^*(Q \setminus \{h\}, R)$. By the previous strict inequality, we can make a small step (starting from $x^*(Q, R)$) along this line segment without violating the constraint h (Figure 12.1). And since both $x^*(Q, R)$ as well as $x^*(Q \setminus \{h\}, R)$ satisfy all other constraints in $(Q \setminus \{h\}, R)$, convexity of the constraints implies that this small step takes us to a feasible solution of $LP(Q, R)$ again. But this solution is lexicographically larger than $x^*(Q, R)$, since we move towards the lexicographically larger vector $x^*(Q \setminus \{h\}, R)$; this is a contradiction. □

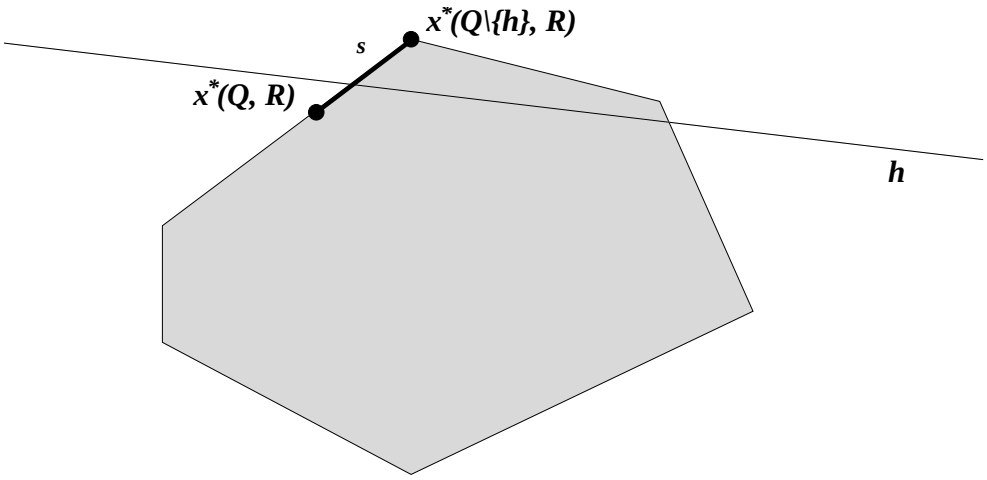


Figure 12.1: *Proof of Lemma 12.7*

12.3 The Algorithm

The algorithm reduces the computation of $x^*(H, \emptyset)$ to the computation of $x^*(Q, R)$ for various sets Q, R, where R is an *independent* set of constraints. Suppose you want to compute $x^*(Q, R)$ (assuming that $x^*(Q, R) \neq \infty$). If $Q = \emptyset$, this is "easy", since we have a constant-size problem, defined by R with $|R| \leqslant d$ and $2d$ bounding-box constraints $-M \leqslant x_i \leqslant M$.

Otherwise, we choose $h \in Q$ and recursively compute $x^*(Q \setminus \{h\}, R) \neq \infty$. We then check whether constraint h is violated by this solution. If not, we are done, since then $x^*(Q \setminus \{h\}, R) = x^*(Q, R)$ (Think about why!). But if h is violated, we can use Lemma 12.7 to conclude that $x^*(Q, R) = x^*(Q \setminus \{h\}, R \cup \{h\})$, and we recursively compute the latter solution. Here is the complete pseudocode.

```
LP(Q, R):
    IF Q = ∅ THEN
        RETURN x*(∅, R)
    ELSE
        choose h ∈ Q uniformly at random
        x* := LP(Q \ {h}, R)
        IF a_h x* ⩽ b_h THEN
            RETURN x*
        ELSE
            RETURN LP(Q \ {h}, R ∪ {h})
        END
    END
```

To solve the original problem, we call this algorithm with $\mathcal{LP}(H, \emptyset)$. It is clear that the algorithm terminates since the first argument Q becomes smaller in every recursive call. It is also true (Exercise) that every set R that comes up during this algorithm is indeed an independent set of constraints and in particular has at most d elements. The correctness of the algorithm then follows from Lemma 12.7.

Exercise 12.8 *Prove that all sets R of constraints that arise during a call to algorithm $\mathcal{LP}(H, \emptyset)$ are independent, meaning that the set*

$$\{x \in \mathbb{R}^d : a_h x = b_h, h \in R\}$$

of points that satisfy all constraints in R with equality has dimension $d - |R|$.

12.4 Runtime Analysis

Now we get to the analysis of algorithm LP, and this will also reveal why the random choice is important.

We will analyze the algorithm in terms of the expected number of *violation tests* $a_h x^* \leqslant b_h$, and in terms of the expected number of *basis computations* $x^*(\emptyset, R)$ that it performs. This is a good measure, since these are the dominating operations of the algorithm. Moreover, both violation test and basis computation are "cheap" operations in the sense that they can be performed in time $f(d)$ for some f.

More specifically, a violation test can be performed in time $O(d)$; the time needed for a basis computation is less clear, since it amounts to solving a small linear program itself. Let us suppose that it is done by brute-force enumeration of all vertices of the bounded polyhedron defined by the at most $3d$ (in)equalities

$$a_h x = b_h, h \in R$$

and

$$-M \leqslant x_i \leqslant M, i = 1, \ldots, d.$$

12.4.1 Violation Tests

Lemma 12.9 *Let $T(n, j)$ be the maximum expected number of violation tests performed in a call to $\mathcal{LP}(Q, R)$ with $|Q| = n$ and $|R| = d - j$. For all $j = 0, \ldots, d$,*

$$\begin{aligned} T(0, j) &= 0 \\ T(n, j) &\leqslant T(n-1, j) + 1 + \frac{j}{n} T(n-1, j-1), \quad n > 0. \end{aligned}$$

Note that in case of $j = 0$, we may get a negative argument on the right-hand side, but due to the factor $0/n$, this does not matter.

Proof. If $|Q| = \emptyset$, there is no violation test. Otherwise, we recursively call $\mathcal{LP}(Q \setminus \{h\}, R)$ for some h which requires at most $T(n-1, j)$ violation tests in expectation. Then there is one violation test within the call to $\mathcal{LP}(Q, R)$ itself, and depending on the outcome, we perform a second recursive call $\mathcal{LP}(Q \setminus \{h\}, R \cup \{h\})$ which requires an expected number of at most $T(n-1, j-1)$ violation tests. The crucial fact is that the probability of performing a second recursive call is at most j/n.

To see this, fix some $S \subseteq Q, |S| \leqslant d - |R| = j$ such that $x^*(Q, R) = x^*(S, R)$. Such a set S exists by Lemma 12.5. This means, we can remove from Q all constraints except the ones in S, without changing the solution.

If $h \notin S$, we thus have

$$x^*(Q, R) = x^*(Q \setminus \{h\}, R),$$

meaning that we have already found $x^*(Q, R)$ after the first recursive call; in particular, we will then have $a_h x^* \leqslant b_h$, and there is no second recursive call. Only if $h \in S$ (and this happens with probability $|S|/n \leqslant j/n$), there can be a second recursive call. □

The following can easily be verified by induction.

Theorem 12.10

$$T(n, j) \leqslant \sum_{i=0}^{j} \frac{1}{i!} j! n.$$

Since $\sum_{i=0}^{j} \frac{1}{i!} \leqslant \sum_{i=0}^{\infty} \frac{1}{i!} = e$, we have $T(n, j) = O(j!n)$. If $d \geqslant j$ is constant, this is $O(n)$.

12.4.2 Basis Computations

To count the basis computations, we proceed as in Lemma 12.9, except that the "1" now moves to a different place.

Lemma 12.11 *Let* $B(n, j)$ *be the maximum expected number of basis computations performed in a call to* $\mathcal{LP}(Q, R)$ *with* $|Q| = n$ *and* $|R| = d - j$*. For all* $j = 0, \ldots, d$*,*

$$\begin{aligned} B(0, j) &= 1 \\ B(n, j) &\leqslant B(n-1, j) + \frac{j}{n} B(n-1, j-1), \quad n > 0. \end{aligned}$$

Interestingly, $B(n, j)$ turns out to be much smaller than $T(n, j)$ which is good since a basic computation is much more expensive than a violation test. Here is the bound that we get.

Theorem 12.12

$$B(n, j) \leqslant (1 + H_n)^j = O(\log^j n),$$

where H_n *is the* n*-th Harmonic number.*

Proof. This is also a simple induction, but let's do this one since it is not entirely obvious. The statement holds for $n = 0$ with the convention that $H_0 = 0$. It also holds for $j = 0$, since Lemma 12.11 implies $B(n, 0) = 1$ for all n. For $n, j > 0$, we inductively obtain

$$\begin{aligned} B(n, j) &\leqslant (1 + H_{n-1})^j + \frac{j}{n}(1 + H_{n-1})^{j-1} \\ &\leqslant \sum_{k=0}^{j} \binom{j}{k} (1 + H_{n-1})^{j-k} (\frac{1}{n})^k \\ &= (1 + H_{n-1} + \frac{1}{n})^j = (1 + H_n)^j. \end{aligned}$$

The second inequality uses the fact that the terms $(1 + H_{n-1})^j$ and $\frac{j}{n}(1 + H_{n-1})^{j-1}$ are the first two terms of the sum in the second line. □

12.4.3 The Overall Bound

Putting together Theorems 12.10 and 12.12 (for $j = d$, corresponding to the case $R = \emptyset$), we obtain the following

Theorem 12.13 *A linear program with* n *constraints in* d *variables (*d *a constant) can be solved in time* $O(n)$*.*

Questions

53. *What is Helly's Theorem?* Give a precise statement and outline the application of the theorem for linear programming (Lemma 12.5).

54. *Outline an algorithm for solving linear programs*! Sketch the main steps of the algorithm and the correctness proof! Also explain how one may achieve the preconditions of feasibility and boundedness.

55. *Sketch the analysis of the algorithm!* Explain on an intuitive level how randomization helps, and how the recurrence relations for the expected number of violation tests and basis computations are derived. What is the expected runtime of the algorithm?

References

[1] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, Heidelberg, West Germany, 1987.

[2] R. Seidel, Small-dimensional linear programming and convex hulls made easy, *Discrete Comput. Geom.* **6** (1991), 423–434.

Chapter 13

Line Arrangements

During the course of this lecture we encountered several situations where it was convenient to assume that a point set is "in general position". In the plane, general position usually amounts to no three points being collinear and/or no four of them being cocircular. This raises an algorithmic question: How can we test for n given points whether or not three of them are collinear? Obviously, we can test all triples in $O(n^3)$ time. Can we do better? In order to answer this question, we will take a detour through the dual plane.

Observe that points and hyperplanes in $\mathbb{R}^d$ are very similar objects, given that both can be described using d coordinates/parameters. It is thus tempting to match these parameters to each other and so create a mapping between points and hyperplanes. In $\mathbb{R}^2$ hyperplanes are lines and the standard *projective duality* transform maps a point $p = (p_x, p_y)$ to the line $p^* : y = p_x x - p_y$ and a non-vertical line $g : y = mx + b$ to the point $g^* = (m, -b)$.

Proposition 13.1 *The standard projective duality transform is*

- *incidence preserving:* $p \in g \iff g^* \in p^*$ *and*
- *order preserving:* p *is above* $g \iff g^*$ *is above* p^*.

Exercise 13.2 *Prove Proposition 13.1.*

Exercise 13.3 *Describe the image of the following point sets under this mapping*

a) a halfplane

b) $k \geqslant 3$ *collinear points*

c) a line segment

d) the boundary points of the upper convex hull of a finite point set.

Another way to think of duality is in terms of the parabola $\mathcal{P} : y = \frac{1}{2}x^2$. For a point p on $\mathcal{P}$, the dual line p^* is the tangent to $\mathcal{P}$ at p. For a point p not on $\mathcal{P}$, consider the vertical projection p' of p onto $\mathcal{P}$: the slopes of p^* and p'^* are the same, just p^* is shifted by the difference in y-coordinates.

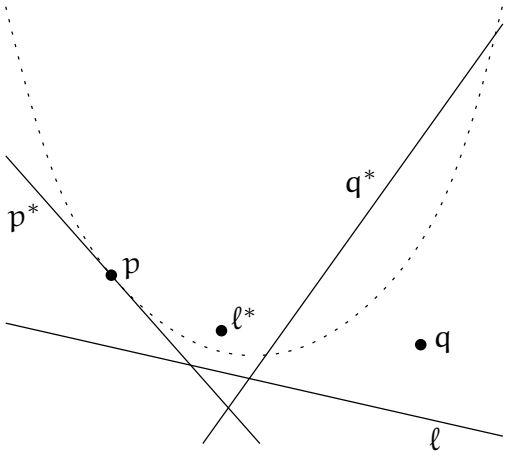


Figure 13.1: *Point $\leftrightarrow$ line duality with respect to the parabola $y = \frac{1}{2}x^2$.*

The question of whether or not three points in the primal plane are collinear transforms to whether or not three lines in the dual plane meet in a point. This question in turn we will answer with the help of *line arrangements*, as defined below.

13.1 Arrangements

The subdivision of the plane induced by a finite set L of lines is called the **arrangement** $\mathcal{A}(L)$. Observe that all cells of the subdivision are intersections of halfplanes and thus convex. A line arrangement is **simple** if no two lines are parallel and no three lines meet in a point. Although lines are unbounded, we can regard a line arrangement a bounded object by (conceptually) putting a sufficiently large box around that contains all vertices. Such a box can be constructed in $O(n \log n)$ time for n lines.

Exercise 13.4 *How?*

Moreover, we can view a line arrangement as a planar graph by adding an additional vertex at "infinity", that is incident to all rays which leave this bounding box. For algorithmic purposes, we will mostly think of an arrangement as being represented by a doubly connected edge list (DCEL), cf. Section 5.2.

Theorem 13.5 *A simple arrangement $\mathcal{A}(L)$ of n lines in $\mathbb{R}^2$ has $\binom{n}{2}$ vertices, n^2 edges, and $\binom{n}{2} + n + 1$ faces/cells.*

Proof. Since all lines intersect and all intersection points are pairwise distinct, there are $\binom{n}{2}$ vertices.

The number of edges we count using induction on n. For $n = 1$ we have $1^2 = 1$ edge. By adding one line to an arrangement of $n-1$ lines we split $n-1$ existing edges into two and introduce n new edges along the newly inserted line. Thus, there are in total $(n-1)^2 + 2n - 1 = n^2 - 2n + 1 + 2n - 1 = n^2$ edges.

The number f of faces can now be obtained from Euler's formula $v - e + f = 2$, where v and e denote the number of vertices and edges, respectively. However, in order to apply Euler's formula we need to consider $\mathcal{A}(L)$ as a planar graph and take the symbolic "infinite" vertex into account. Therefore,

$$f = 2 - \left(\binom{n}{2} + 1\right) + n^2 = 1 + \frac{1}{2}(2n^2 - n(n-1)) = 1 + \frac{1}{2}(n^2 + n) = 1 + \binom{n}{2} + n\,.$$

□

The **complexity** of an arrangement is simply the total number of vertices, edges, and faces (in general, cells of any dimension).

Exercise 13.6 *Consider a set of lines in the plane with no three intersecting in a common point. Form a graph* G *whose vertices are the intersection points of the lines and such that two vertices are adjacent if and only if they appear consecutively along one of the lines. Prove that* $\chi(G) \leqslant 3$*, where* $\chi(G)$ *denotes the chromatic number of the graph* G*. In other words, show how to color the vertices of* G *using at most three colors such that no two adjacent vertices have the same color.*

13.2 Construction

As the complexity of a line arrangement is quadratic, there is no need to look for a subquadratic algorithm to construct it. We will simply construct it incrementally, inserting the lines one by one. Let $\ell_1, \ldots, \ell_n$ be the order of insertion.

At Step i of the construction, locate ℓ_i in the leftmost cell of $\mathcal{A}(\{\ell_1, \ldots, \ell_{i-1}\})$ it intersects. (The halfedges leaving the infinite vertex are ordered by slope.) This takes $O(i)$ time. Then traverse the boundary of the face F found until the halfedge h is found where ℓ_i leaves F (see Figure 13.2 for illustration). Insert a new vertex at this point, splitting F and h and continue in the same way with the face on the other side of h.

The insertion of a new vertex involves splitting two halfedges and thus is a constant time operation. But what is the time needed for the traversal? The complexity of $\mathcal{A}(\{\ell_1, \ldots, \ell_{i-1}\})$ is $\Theta(i^2)$, but we will see that the region traversed by a single line has linear complexity only.

13.3 Zone Theorem

For a line ℓ and an arrangement $\mathcal{A}(L)$, the **zone** $Z_{\mathcal{A}(L)}(\ell)$ of ℓ in $\mathcal{A}(L)$ is the set of cells from $\mathcal{A}(L)$ whose closure intersects ℓ.

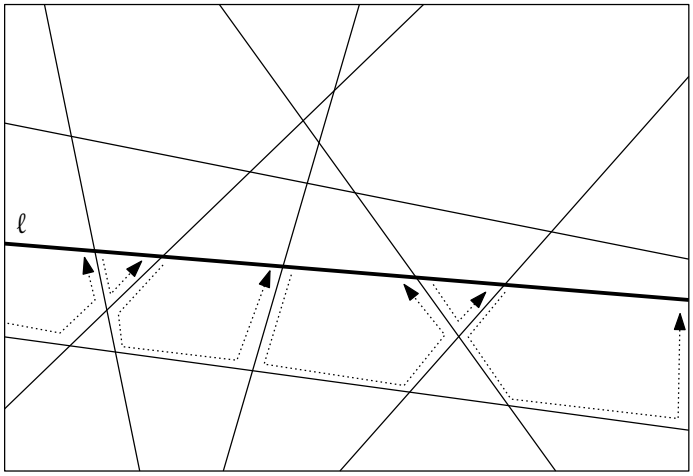
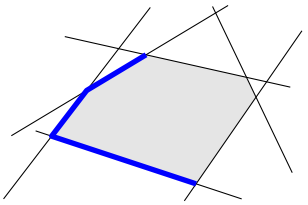


Figure 13.2: *Incremental construction: Insertion of a line ℓ. (Only part of the arrangement is shown in order to increase readability.)*

Theorem 13.7 *Given an arrangement $\mathcal{A}(L)$ of n lines in $\mathbb{R}^2$ and a line ℓ (not necessarily from L), the total number of edges in all cells of the zone $Z_{\mathcal{A}(L)}(\ell)$ is at most $6n$.*

Proof. Without loss of generality suppose that ℓ is horizontal and that none of the lines from L is horizontal. (The first condition can be addressed by rotating the plane and the second by deciding that the left vertex of a horizontal edge is higher than the right vertex.)

For each cell of $Z_{\mathcal{A}(L)}(\ell)$ split its boundary at its topmost vertex and at its bottommost vertex and orient all edges from bottom to top. Those edges that have the cell to their right are called *left-bounding* for the cell and those edges that have the cell to their left are called right-bounding. For instance, for the cell depicted to the right all left-bounding edges are shown blue and bold.



We will show that there are at most $3n$ left-bounding edges in $Z_{\mathcal{A}(L)}(\ell)$ by induction on n. By symmetry, the same bound holds also for the number of right-bounding edges in $Z_{\mathcal{A}(L)}(\ell)$.

For $n = 1$, there is at most one (exactly one, unless ℓ is parallel to and lies below the only line in L) left-bounding edge in $Z_{\mathcal{A}(L)}(\ell)$ and $1 \leqslant 3n = 3$. Assume the statement is true for $n - 1$.

If no line from L intersects ℓ, then all lines in $L \cup \{\ell\}$ are parallel and there are at most $2 < 3n$ left-bounding edges in $Z_{\mathcal{A}(L)}(\ell)$. Else consider the rightmost line r from L intersecting ℓ and the arrangement $\mathcal{A}(L \setminus \{r\})$. By the induction hypothesis there are at most $3n - 3$ left-bounding edges in $Z_{\mathcal{A}(L\setminus\{r\})}(\ell)$. Adding r back adds at most three new left-bounding edges: At most two edges (call them ℓ_0 and ℓ_1) of the rightmost cell of $Z_{\mathcal{A}(L\setminus\{r\})}(\ell)$ are intersected by r and thereby split in two. Both of these two edges

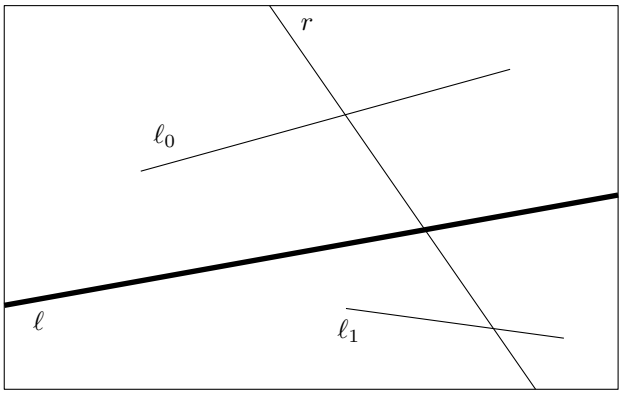


Figure 13.3: *At most three new left-bounding edges are created by adding* r *to* $\mathcal{A}(L \setminus \{r\})$.

may be left-bounding and thereby increase the number of left-bounding edges by at most two. In any case, r itself contributes exactly one more left-bounding edge to that cell. The line r cannot contribute a left-bounding edge to any cell other than the rightmost: to the left of r, the edges induced by r form right-bounding edges only and to the right of r all other cells touched by r (if any) are shielded away from ℓ by one of ℓ_0 or ℓ_1. Therefore, the total number of left-bounding edges in $Z_{\mathcal{A}(L)}(\ell)$ is bounded from above by $3 + 3n - 3 = 3n$. □

Corollary 13.8 *The arrangement of* n *lines in* $\mathbb{R}^2$ *can be constructed in optimal* $O(n^2)$ *time and space.*

Proof. Use the incremental construction described above. In Step i, for $1 \leqslant i \leqslant n$, we do a linear search among $i-1$ elements to find the starting face and then traverse (part of) the zone of the line ℓ_i in the arrangement $\mathcal{A}(\{\ell_1, \ldots, \ell_{i-1}\})$. By Theorem 13.7 the complexity of this zone and hence the time complexity of Step i altogether is $O(i)$. Overall we obtain $\sum_{i=1}^{n} ci = O(n^2)$ time (and space), for some constant $c > 0$, which is optimal by Theorem 13.5. □

The corresponding bounds for hyperplane arrangements in $\mathbb{R}^d$ are $\Theta(n^d)$ for the complexity of a simple arrangement and $O(n^{d-1})$ for the complexity of a zone of a hyperplane.

Exercise 13.9 *For an arrangement* $\mathcal{A}$ *of a set of* n *lines in* $\mathbb{R}^2$*, let* $\mathcal{F} := \bigcup_{C \text{ is cell of } \mathcal{A}} \overline{C}$ *denote the union of the closure of all bounded cells. Show that the complexity (number of vertices and edges of the arrangement lying on the boundary) of* $\mathcal{F}$ *is* $O(n)$.

13.4 The Power of Duality

The real beauty and power of line arrangements becomes apparent in context of projective point $\leftrightarrow$ line duality. The following problems all can be solved in $O(n^2)$ time and space

by constructing the dual arrangement.

General position test. Given n points in $\mathbb{R}^2$, are any three of them collinear? (Dual: do three lines meet in a point?)

Minimum area triangle. Given n points in $\mathbb{R}^2$, what is the minimum area triangle spanned by any three of them? For any vertex ℓ^* of the dual arrangement (primal: line ℓ through two points p and q) find the closest point vertically above/below ℓ^* through which an input line passes (primal: closest line below/above and parallel to ℓ that passes through an input point). In this way one can find $O(n^2)$ candidate triangles by constructing the arrangement of the n dual lines[1] The smallest among those candidates can be determined by a straightforward minimum selection (comparing the area of the corresponding triangles). Observe that vertical distance is not what determines the area of the corresponding triangle but orthogonal distance. However, the points that minimize these measures for any fixed line are the same...

Exercise 13.10 *A set P of n points in the plane is said to be in* ε-general position *for $\varepsilon > 0$ if no three points of the form*

$$p + (x_1, y_1), q + (x_2, y_2), r + (x_3, y_3)$$

are collinear, where $p, q, r \in P$ and $|x_i|, |y_i| < \varepsilon$, for $i \in \{1, 2, 3\}$. In words: P remains in general position under changing point coordinates by less than ε each.

Give an algorithm with runtime $O(n^2)$ for checking whether a given point set P is in ε-general position.

13.5 Sorting all Angular Sequences.

Theorem 13.11 *Consider a set P of n points in the plane. For a point $q \in P$ let $c_P(q)$ denote the circular sequence of points from $S \setminus \{q\}$ ordered counterclockwise around q (in order as they would be encountered by a ray sweeping around q). All $c_P(q)$, $q \in P$, collectively can be obtained in $O(n^2)$ time.*

Proof. Assume without loss of generality that no two points in P have the same x-coordinate (else rotate the plane infinitesimally). Consider the projective dual P^* of P. An angular sweep around a point $q \in P$ in the primal plane corresponds to a traversal of the line q^* from left to right in the dual plane. (A collection of lines through a single point q corresponds to a collection of points on a single line q^* and slope corresponds to x-coordinate.) Clearly, the sequence of intersection points along all lines in P^* can

[1] For instance, maintain over the incremental construction for each vertex a vertically closest line. The number of vertices to be updated during insertion of a line ℓ corresponds to the complexity of the zone of ℓ in the arrangement constructed so far. Therefore maintaining this information comes at no extra cost asymptotically.

be obtained by constructing the arrangement in $O(n^2)$ time. In the primal plane, any such sequence corresponds to an order of the remaining points according to the slope of the connecting line; to construct the circular sequence of points as they are encountered around q, we have to split the sequence obtained from the dual into those points that are to the left of q and those that are to the right of q; concatenating both yields the desired sequence. □

Exercise 13.12 (Eppstein [1]) *Describe an $O(n^2)$ time algorithm that given a set P of n points in the plane finds a subset of five points that form a strictly convex empty pentagon (or reports that there is none if that is the case). Empty means that the convex pentagon may not contain any other points of P.*
Hint: Start with a point $p \in P$ that is extremal in one direction and try to find out whether there is a solution P' containing p. For this, consider the star-shaped polygon that visits all points in radial order, as seen from p.
Remark: It was shown by Harborth [5] that every set of ten or more points in general position contains a subset of five points that form a strictly convex empty pentagon.

13.6 Segment Endpoint Visibility Graphs

A fundamental problem in motion planning is to find a short(est) path between two given positions in some domain, subject to certain constraints. As an example, suppose we are given two points $p, q \in \mathbb{R}^2$ and a set $S \subset \mathbb{R}^2$ of obstacles. What is the shortest path between p and q that avoids S?

Observation 13.13 *The shortest path (if it exists) between two points that does not cross a finite set of finite polygonal obstacles is a polygonal path whose interior vertices are obstacle vertices.*

One of the simplest type of obstacle conceivable is a line segment. In general the plane may be disconnected with respect to the obstacles, for instance, if they form a closed curve. However, if we restrict the obstacles to pairwise disjoint line segments then there is always a free path between any two given points. Apart from start and goal position, by the above observation we may restrict our attention concerning shortest paths to straight line edges connecting obstacle vertices, in this case, segment endpoints.

Definition 13.14 *Consider a set S of n disjoint line segments in $\mathbb{R}^2$. The* **segment endpoint visibility graph** *$\mathcal{V}(S)$ is a geometric straight line graph defined on the segment endpoints. Two segment endpoints p and q are connected in $\mathcal{V}(S)$ if and only if*

- *the line segment $\overline{pq}$ is in S or*
- *$\overline{pq} \cap s \subseteq \{p, q\}$ for every segment $s \in S$.*

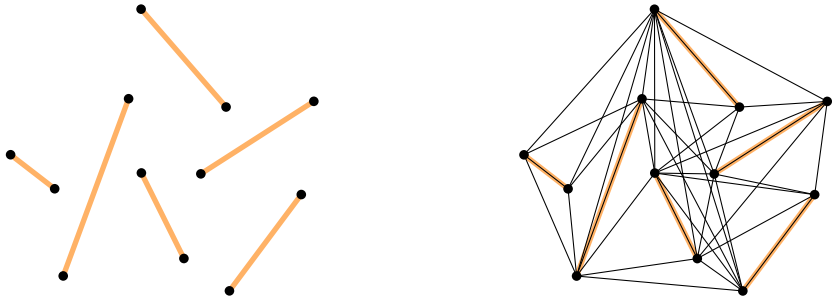


Figure 13.4: *A set of disjoint line segments and their endpoint visibility graph.*

If all segments are on the convex hull, the visibility graph is complete. If they form parallel chords of a convex polygon, the visibility graph consists of copies of K_4, glued together along opposite edges and the total number of edges is linear only.

These graphs also appear in the context of the following question: Given a set of disjoint line segments, is it possible to connect them to form (the boundary of) a simple polygon? Is it easy to see that this is not possible in general: Just take three parallel chords of a convex polygon (Figure 13.5a). However, if we do not insist that the segments appear on the boundary, but allow them to be diagonals or epigonals, then it is always possible [7, 6]. In other words, the segment endpoint visibility graph of disjoint line segments is Hamiltonian, unless all segments are collinear. It is actually essential to allow epigonals and not only diagonals [9, 4] (Figure 13.5b).

Figure 13.5: *Sets of disjoint line segments that do not allow certain polygons.*

Constructing $\mathcal{V}(S)$ for a given set S of disjoint segments in a brute force way takes $O(n^3)$ time. (Take all pairs of endpoints and check all other segments for obstruction.)

Theorem 13.15 (Welzl [10]) *The segment endpoint visibility graph of n disjoint line segments can be constructed in worst case optimal $O(n^2)$ time.*

Proof. For simplicity we assume general position, that is, no three endpoints are collinear and no two have the same x-coordinate. It is no problem to handle such degeneracies explicitly.

We have seen above how all sorted angular sequences can be obtained from the dual line arrangement in $O(n^2)$ time. Topologically sweep the arrangement from left to right (corresponds to changing the slope of the primal rays from $-\infty$ to $+\infty$) while maintaining for each segment endpoint p the segment $s(p)$ it currently "sees" (if any). Initialize by

brute force in $O(n^2)$ time (direction vertically downwards). Each intersection of two lines corresponds to two segment endpoints "seeing" each other along the primal line whose dual is the point of intersection. In order to process an intersection, we only need that all preceding (located to the left) intersections of the two lines involved have already been processed. This order corresponds to a topological sort of the arrangement graph where all edges are directed from left to right. (Clearly, this graph is acyclic, that is, it does not contain a directed cycle.) A topological sort can be obtained, for instance, via (reversed) post order DFS in time linear in the size of the graph (number of vertices and edges), which in our case here is $O(n^2)$.

When processing an intersection, there are four cases. Let p and q be the two points involved such that p is to the left of q.

1. The two points belong to the same input segment $\rightarrow$ output the edge pq, no change otherwise.

2. q is obscured from p by $s(p) \rightarrow$ no change.

3. q is endpoint of $s(p) \rightarrow$ output pq and update $s(p)$ to $s(q)$.

4. Otherwise q is endpoint of a segment t that now obscures $s(p) \rightarrow$ output pq and update $s(p)$ to t.

Thus any intersection can be processed in constant time and the overall runtime of this algorithm is quadratic. □

13.7 Ham Sandwich Theorem

Suppose two thieves have stolen a necklace that contains rubies and diamonds. Now it is time to distribute the prey. Both, of course, should get the same number of rubies and the same number of diamonds. On the other hand, it would be a pity to completely disintegrate the beautiful necklace. Hence they want to use as few cuts as possible to achieve a fair gem distribution.

To phrase the problem in a geometric (and somewhat more general) setting: Given two finite sets R and D of points, construct a line that bisects both sets, that is, in either halfplane defined by the line there are about half of the points from R and about half of the points from D. To solve this problem, we will make use of the concept of levels in arrangements.

Definition 13.16 *Consider an arrangement $\mathcal{A}(L)$ induced by a set L of n non-vertical lines in the plane. We say that a point p is on the* k**-level** *in $\mathcal{A}(L)$ if and only p lies on some line from L and there are at most $k-1$ lines below and at most $n-k$ lines above p. The 1-level and the n-level are also referred to as* **lower** *and* **upper envelope**, *respectively.*

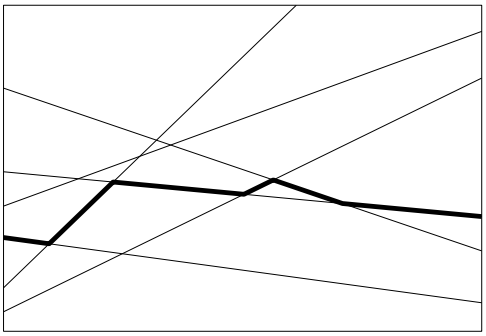


Figure 13.6: *The 3-level of an arrangement.*

Another way to look at the k-level is to consider the lines to be real functions; then the lower envelope is the pointwise minimum of those functions, and the k-level is defined by taking pointwise the k^{th}-smallest function value.

Theorem 13.17 *Let* $R, D \subset \mathbb{R}^2$ *be finite sets of points. Then there exists a line that bisects both* R *and* D*. That is, in either open halfplane defined by* ℓ *there are no more than* $|R|/2$ *points from* R *and no more than* $|D|/2$ *points from* D*.*

Proof. Without loss of generality suppose that both $|R|$ and $|D|$ are odd. (If, say, $|R|$ is even, simply remove an arbitrary point from R. Any bisector for the resulting set is also a bisector for R.) We may also suppose that no two points from $R \cup D$ have the same x-coordinate. (Otherwise, rotate the plane infinitesimally.)

Let R^* and D^* denote the set of lines dual to the points from R and D, respectively. Consider the arrangement $\mathcal{A}(R^*)$. The median level of $\mathcal{A}(R^*)$ defines the bisecting lines for R. As $|R| = |R^*|$ is odd, both the leftmost and the rightmost segment of this level are defined by the same line ℓ_r from R^*, the one with median slope. Similarly there is a corresponding line ℓ_d in $\mathcal{A}(D^*)$.

Since no two points from $R \cup D$ have the same x-coordinate, no two lines from $R^* \cup D^*$ have the same slope, and thus ℓ_r and ℓ_d intersect. Consequently, being piecewise linear continuous functions, the median level of $\mathcal{A}(R^*)$ and the median level of $\mathcal{A}(D^*)$ intersect (see Figure 13.7 for an example). Any point that lies on both median levels corresponds to a primal line that bisects both point sets simultaneously. □

How can the thieves use Theorem 13.17? If they are smart, they drape the necklace along some convex curve, say, a circle. Then by Theorem 13.17 there exists a line that simultaneously bisects the set of diamonds and the set of rubies. As any line intersects the circle at most twice, the necklace is cut at most twice. It is easy to turn the proof given above into an $O(n^2)$ algorithm to construct a line that simultaneously bisects both sets.

You can also think of the two point sets as a discrete distribution of a ham sandwich that is to be cut fairly, that is, in such a way that both parts have the same amount of ham and the same amount of bread. That is where the name "ham sandwich cut" comes

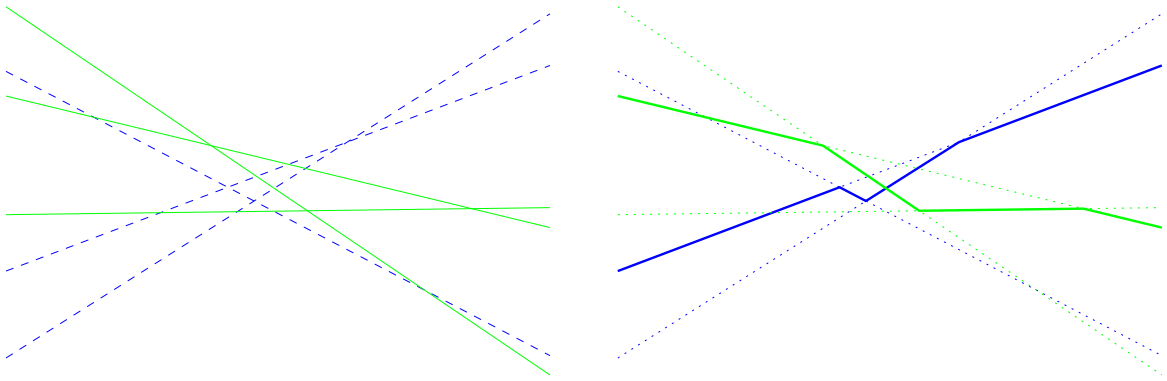


Figure 13.7: *An arrangement of 3 green lines (solid) and 3 blue lines (dashed) and their median levels (marked bold on the right hand side).*

from. The theorem also holds in $\mathbb{R}^d$, saying that any d finite point sets (or finite Borel measures, if you want) can simultaneously be bisected by a hyperplane. This implies that the thieves can fairly distribute a necklace consisting of d types of gems using at most d cuts.

Algorithmically the problem gets harder in higher dimension. But in the plane, a ham sandwich cut can be found in linear time using a sophisticated prune and search algorithm by Lo, Matoušek and Steiger [8].

Exercise 13.18 *The goal of this exercise is to develop a data structure for halfspace range counting.*

a) Given a set $P \subset \mathbb{R}^2$ of n points in general position, show that it is possible to partition this set by two lines such that each region contains at most $\lceil \frac{n}{4} \rceil$ points.

b) Design a data structure of size $O(n)$, which can be constructed in time $O(n \log n)$ and allows you, for any halfspace h, to output the number of points $|P \cap h|$ of P contained in this halfspace h in time $O(n^\alpha)$, for some $0 < \alpha < 1$.

Exercise 13.19 *Prove or disprove the following statement: Given three finite sets A, B, C of points in the plane, there is always a circle or a line that bisects A, B and C simultaneously (that is, no more than half of the points of each set are inside or outside the circle or on either side of the line, respectively).*

13.8 3-Sum

The 3-Sum problem is the following: Given a set S of n integers, does there exist a three-tuple[2] of elements from S that sum up to zero? By testing all three-tuples this

[2]That is, an element of S may be chosen twice or even three times, although the latter makes sense for the number 0 only. :-)

can obviously be solved in $O(n^3)$ time. If the tuples to be tested are picked a bit more cleverly, we obtain an $O(n^2)$ algorithm.

Let $(s_1, \ldots, s_n)$ be the sequence of elements from S in increasing order. Then we test the tuples as follows.

```
For i = 1, ..., n {
    j = i, k = n.
    While k ⩾ j {
        If s_i + s_j + s_k = 0 then exit with triple s_i, s_j, s_k.
        If s_i + s_j + s_k > 0 then k = k − 1 else j = j + 1.
    }
}
```

The runtime is clearly quadratic (initial sorting can be done in $O(n \log n)$ time). Regarding the correctness observe that the following is an invariant that holds at the start of every iteration of the inner loop: $s_i + s_x + s_k < 0$, for all $i \leqslant x < j$, and $s_i + s_j + s_x > 0$, for all $k < x \leqslant n$.

Interestingly, this is essentially the best algorithm known for 3-Sum. It is widely believed that the problem cannot be solved in sub-quadratic time, but so far this has been proved in some very restricted models of computation only, such as the linear decision tree model [2].

13.9 3-Sum hardness

There is a whole class of problems that are equivalent to 3-Sum up to sub-quadratic time reductions [3]; such problems are referred to as **3-Sum-hard**.

Definition 13.20 *A problem* P *is* **3-Sum-hard** *if and only if every instance of 3-Sum of size* n *can be solved using a constant number of instances of* P*—each of* $O(n)$ *size—and* $o(n^2)$ *additional time.*

For instance, it is not hard to show that the following variation of 3-Sum—let us denote it by 3-Sum°—is 3-Sum hard: Given a set S of n integers, does there exist a three-element subset of S whose elements sum up to zero?

As another example, consider the Problem **GeomBase**: Given n points on the three horizontal lines $y = 0$, $y = 1$, and $y = 2$, is there a non-horizontal line that contains at least three of them?

3-Sum can be reduced to GeomBase as follows. For an instance $S = \{s_1, \ldots, s_n\}$ of 3-Sum, create an instance P of GeomBase in which for each s_i there are three points in P: $(s_i, 0)$, $(-s_i/2, 1)$, and $(s_i, 2)$. If there are any three collinear points in P, there must be one from each of the lines $y = 0$, $y = 1$, and $y = 2$. So suppose that $p = (s_i, 0)$, $q = (-s_j/2, 1)$, and $r = (s_k, 2)$ are collinear. The inverse slope of the line through p and q is $\frac{-s_j/2 - s_i}{1-0} = -s_j/2 - s_i$ and the inverse slope of the line through q and r is

$\frac{s_k+s_j/2}{2-1} = s_k + s_j/2$. The three points are collinear if and only if the two slopes are equal, that is, $-s_j/2 - s_i = s_k + s_j/2 \iff s_i + s_j + s_k = 0$.

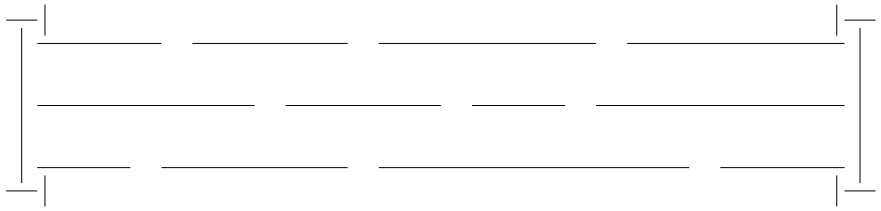
A very similar problem is **General Position**, in which one is given n arbitrary points and has to decide whether any three are collinear. For an instance S of 3-Sum°, create an instance P of General Position by projecting the numbers s_i onto the curve $y = x^3$, that is, $P = \{(a, a^3) \mid a \in S\}$.

Suppose three of the points, say, (a, a^3), (b, b^3), and (c, c^3) are collinear. This is the case if and only if the slopes of the lines through each pair of them are equal. (Observe that a, b, and c are pairwise distinct.)

$$\begin{aligned}
(b^3 - a^3)/(b - a) &= (c^3 - b^3)/(c - b) \iff \\
b^2 + a^2 + ab &= c^2 + b^2 + bc \iff \\
b &= (c^2 - a^2)/(a - c) \iff \\
b &= -(a + c) \iff \\
a + b + c &= 0\,.
\end{aligned}$$

Minimum Area Triangle is a strict generalization of General Position and, therefore, also 3-Sum-hard.

In **Segment Splitting/Separation**, we are given a set of n line segments and have to decide whether there exists a line that does not intersect any of the segments but splits them into two non-empty subsets. To show that this problem is 3-Sum-hard, we can use essentially the same reduction as for GeomBase, where we interpret the points along the three lines $y = 0$, $y = 1$, and $y = 2$ as sufficiently small "holes". The parts of the lines that remain after punching these holes form the input segments for the Splitting problem. Horizontal splits can be prevented by putting constant size gadgets somewhere beyond the last holes, see the figure below. The set of input segments for the segment



splitting problem requires sorting the points along each of the three horizontal lines, which can be done in $O(n \log n) = o(n^2)$ time. It remains to specify what "sufficiently small" means for the size of those holes. As all input numbers are integers, it is not hard to see that punching a hole of $(x - 1/4, x + 1/4)$ around each input point x is small enough.

In **Segment Visibility**, we are given a set S of n horizontal line segments and two segments $s_1, s_2 \in S$. The question is: Are there two points, $p_1 \in s_1$ and $p_2 \in s_2$ which can see each other, that is, the open line segment $\overline{p_1 p_2}$ does not intersect any segment from S? The reduction from 3-Sum is the same as for Segment Splitting, just put s_1 above and s_2 below the segments along the three lines.

In **Motion Planning**, we are given a robot (line segment), some environment (modeled as a set of disjoint line segments), and a source and a target position. The question is: Can the robot move (by translation and rotation) from the source to the target position, without ever intersecting the "walls" of the environment?

To show that Motion Planning is 3-Sum-hard, employ the reduction for Segment Splitting from above. The three "punched" lines form the doorway between two rooms, each modeled by a constant number of segments that cannot be split, similar to the boundary gadgets above. The source position is in one room, the target position in the other, and to get from source to target the robot has to pass through a sequence of three collinear holes in the door (suppose the doorway is sufficiently small compared to the length of the robot).

Exercise 13.21 *The 3-Sum' problem is defined as follows: given three sets S_1, S_2, S_3 of n integers each, are there $a_1 \in S_1$, $a_2 \in S_2$, $a_3 \in S_3$ such that $a_1 + a_2 + a_3 = 0$? Prove that the 3-Sum' problem and the 3-Sum problem as defined in the lecture ($S_1 = S_2 = S_3$) are equivalent, more precisely, that they are reducible to each other in subquadratic time.*

Questions

56. *How can one construct an arrangement of lines in $\mathbb{R}^2$?* Describe the incremental algorithm and prove that its time complexity is quadratic in the number of lines (incl. statement and proof of the Zone Theorem).

57. *How can one test whether there are three collinear points in a set of n given points in $\mathbb{R}^2$?* Describe an $O(n^2)$ time algorithm.

58. *How can one compute the minimum area triangle spanned by three out of n given points in $\mathbb{R}^2$?* Describe an $O(n^2)$ time algorithm.

59. *What is a ham-sandwich cut? Does it always exist? How to compute it?* State and prove the theorem about the existence of a ham-sandwich cut in $\mathbb{R}^2$ and describe an $O(n^2)$ algorithm to compute it.

60. *What is the endpoint visibility graph for a set of disjoint line segments in the plane and how can it be constructed?* Give the definition and explain the relation to shortest paths. Describe the $O(n^2)$ algorithm by Welzl, including full proofs of Theorem 13.11 and Theorem 13.15.

61. *Is there a subquadratic algorithm for General Position?* Explain the term 3-Sum hard and its implications and give the reduction from 3-Sum to General Position.

62. *Which problems are known to be 3-Sum-hard?* List at least three problems (other than 3-Sum) and briefly sketch the corresponding reductions.

References

[1] David Eppstein, Happy endings for flip graphs. *J. Comput. Geom.*, **1**, 1, (2010), 3–28, URL http://jocg.org/index.php/jocg/article/view/21.

[2] Jeff Erickson, Lower bounds for linear satisfiability problems. *Chicago J. Theoret. Comput. Sci.*, **1999**, 8, URL http://cjtcs.cs.uchicago.edu/articles/1999/8/contents.html.

[3] Anka Gajentaan and Mark H. Overmars, On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, **5**, (1995), 165–185, URL http://dx.doi.org/10.1016/0925-7721(95)00022-2.

[4] Branko Grünbaum, Hamiltonian polygons and polyhedra. *Geombinatorics*, **3**, 3, (1994), 83–89.

[5] Heiko Harborth, Konvexe Fünfecke in ebenen Punktmengen. *Elem. Math.*, **33**, (1979), 116–118, URL http://dx.doi.org/10.5169/seals-32945.

[6] Michael Hoffmann, *On the existence of paths and cycles*. Ph.D. thesis, ETH Zürich, 2005, URL http://dx.doi.org/10.3929/ethz-a-004945082.

[7] Michael Hoffmann and Csaba D. Tóth, Segment endpoint visibility graphs are Hamiltonian. *Comput. Geom. Theory Appl.*, **26**, 1, (2003), 47–68, URL http://dx.doi.org/10.1016/S0925-7721(02)00172-4.

[8] Chi-Yuan Lo, Jiří Matoušek, and William L. Steiger, Algorithms for ham-sandwich cuts. *Discrete Comput. Geom.*, **11**, (1994), 433–452, URL http://dx.doi.org/10.1007/BF02574017.

[9] Masatsugu Urabe and Mamoru Watanabe, On a counterexample to a conjecture of Mirzaian. *Comput. Geom. Theory Appl.*, **2**, 1, (1992), 51–53, URL http://dx.doi.org/10.1016/0925-7721(92)90020-S.

[10] Emo Welzl, Constructing the visibility graph for n line segments in $O(n^2)$ time. *Inform. Process. Lett.*, **20**, (1985), 167–171, URL http://dx.doi.org/10.1016/0020-0190(85)90044-4.

Chapter 14

Davenport-Schinzel Sequences

The complexity of a simple arrangement of n lines in $\mathbb{R}^2$ is $\Theta(n^2)$ and so every algorithm that uses such an arrangement explicitly needs $\Omega(n^2)$ time. However, there are many scenarios in which we do not need the whole arrangement but only some part of it. For instance, to construct a ham-sandwich cut for two sets of points in $\mathbb{R}^2$ one needs the median levels of the two corresponding line arrangements only. As mentioned in the previous section, the relevant information about these levels can actually be obtained in linear time. Similarly, in a motion planning problem where the lines are considered as obstacles we are only interested in the cell of the arrangement we are located in. There is no way to ever reach any other cell, anyway.

This chapter is concerned with analyzing the complexity—that is, the number of vertices and edges—of a single cell in an arrangement of n curves in $\mathbb{R}^2$. In case of a line arrangement this is mildly interesting only: Every cell is convex and any line can appear at most once along the cell boundary. On the other hand, it is easy to construct an example in which there is a cell C such that every line appears on the boundary ∂C.

But when we consider arrangement of line segments rather than lines, the situation changes in a surprising way. Certainly a single segment can appear several times along the boundary of a cell, see the example in Figure 14.1. Make a guess: What is the maximal complexity of a cell in an arrangement of n line segments in $\mathbb{R}^2$?

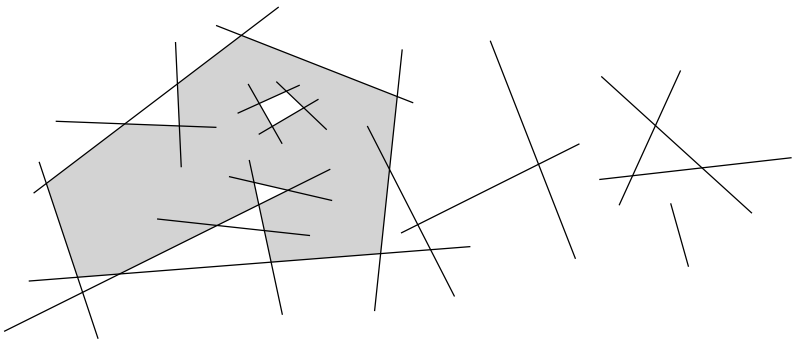


Figure 14.1: *A single cell in an arrangement of line segments.*

You will find out the correct answer soon, although we will not prove it here. But my guess would be that it is rather unlikely that your guess is correct, unless, of course, you knew the answer already. :-)

For a start we will focus on one particular cell of any arrangement that is very easy to describe: the lower envelope or, intuitively, everything that can be seen vertically from below. To analyze the complexity of lower envelopes we use a combinatorial description using strings with forbidden subsequences, so-called Davenport-Schinzel sequences. These sequences are of independent interest, as they appear in a number of combinatorial problems [2] and in the analysis of data structures [7]. The techniques used apply not only to lower envelopes but also to arbitrary cells of arrangements.

14.1 Davenport-Schinzel Sequences

Definition 14.1 *A* (n, s)-**Davenport-Schinzel sequence** *is a sequence over an alphabet* A *of size* n *in which*

- *no two consecutive characters are the same and*
- *there is no alternating subsequence of the form* $\ldots a \ldots b \ldots a \ldots b \ldots$ *of* $s + 2$ *characters, for any* $a, b \in A$.

Let $\lambda_s(n)$ *be the length of a longest* (n, s)*-Davenport-Schinzel sequence.*

For example, $abcbacb$ is a $(3, 4)$-DS sequence but not a $(3, 3)$-DS sequence because it contains the subsequence $bcbcb$.

Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a collection of real-valued continuous functions defined on a common interval $I \subset \mathbb{R}$. The *lower envelope* $\mathcal{L}_{\mathcal{F}}$ of $\mathcal{F}$ is defined as the pointwise minimum of the functions f_i, $1 \leqslant i \leqslant n$, over I. Suppose that any pair f_i, f_j, $1 \leqslant i < j \leqslant n$, intersects in at most s points. Then I can be decomposed into a finite sequence $I_1, \ldots, I_\ell$ of (maximal connected) pieces on each of which a single function from $\mathcal{F}$ defines $\mathcal{L}_{\mathcal{F}}$. Define the sequence $\phi(\mathcal{F}) = (\phi_1, \ldots, \phi_\ell)$, where f_{ϕ_i} is the function from $\mathcal{F}$ which defines $\mathcal{L}_{\mathcal{F}}$ on I_i.

Observation 14.2 $\phi(\mathcal{F})$ *is an* (n, s)*-Davenport-Schinzel sequence.*

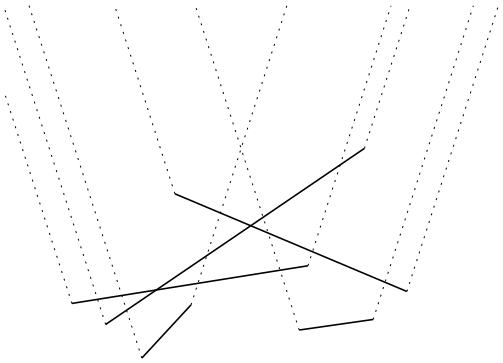
In the case of line segments the above statement does not hold because a set of line segments is in general not defined on a common real interval.

Proposition 14.3 *Let* $\mathcal{F}$ *be a collection of* n *real-valued continuous functions each of which is defined on some real interval. If any two functions from* $\mathcal{F}$ *intersect in at most* s *points then* $\phi(\mathcal{F})$ *is an* $(n, s + 2)$*-Davenport-Schinzel sequence.*

Proof. Let I denote the union of all intervals on which one of the functions from $\mathcal{F}$ is defined. Consider any function $f \in \mathcal{F}$ defined on $[a, b] \subseteq I = [c, d]$. Extend f on I by extending it using almost vertical rays pointing upward, from a use a ray of sufficiently

small slope, from b use a ray of sufficiently large slope. For all functions use the same slope on these two extensions such that no extensions in the same direction intersect. By *sufficiently small/large* we mean that for any extension ray there is no function endpoint nor an intersection point of two functions in the open angular wedge bounded by the extension ray and the vertical ray starting from the same source. (There may be such points *on* the vertical ray, but not in the open wedge between the two rays.)

Denote the resulting collection of functions totally defined on I by $\mathcal{F}'$. If the rays are sufficiently close to vertical then $\phi(\mathcal{F}') = \phi(\mathcal{F})$.



For any $f \in \mathcal{F}'$ a single extension ray can create at most one additional intersection with any $g \in \mathcal{F}'$. (Let $[a_f, b_f]$ and $[a_g, b_g]$ be the intervals on which the function f and g, respectively, was defined originally. Consider the ray r extending f from a_f to the left. If $a_f \in [a_g, b_g]$ then r may create a new intersection with g, if $a_f > b_g$ then r creates a new intersection with the right extension of g from b_g, and if $a_f < a_g$ then r does not create any new intersection with g.)

On the other hand, for any pair s, t of segments, neither the left extension of the leftmost segment endpoint nor the right extension of the rightmost segment endpoint can introduce an additional intersection. Therefore, any pair of segments in $\mathcal{F}'$ intersects at most $s + 2$ times and the claim follows. □

Next we will give an upper bound on the length of Davenport-Schinzel sequences for small s.

Lemma 14.4 *$\lambda_1(n) = n$, $\lambda_2(n) = 2n - 1$, and $\lambda_3(n) \leqslant 2n(1 + \log n)$.*

Proof. $\lambda_1(n) = n$ is obvious. $\lambda_2(n) = 2n - 1$ is given as an exercise. We prove $\lambda_3(n) \leqslant 2n(1 + \log n) = O(n \log n)$.

For $n = 1$ it is $\lambda_3(1) = 1 \leqslant 2$. For $n > 1$ consider any $(n, 3)$-DS sequence σ of length $\lambda_3(n)$. Let a be a character that appears least frequently in σ. Clearly a appears at most $\lambda_3(n)/n$ times in σ. Delete all appearances of a from σ to obtain a sequence σ' on $n-1$ symbols. But σ' is not necessarily a DS sequence because there may be consecutive appearances of a character b in σ', in case that $\sigma = \ldots bab \ldots$.

Claim: There are at most two pairs of consecutive appearances of the same character in σ'. Indeed, such a pair can be created around the first and last appearance

of a in σ only. If any intermediate appearance of a creates a pair bb in σ' then $\sigma = \ldots a \ldots bab \ldots a \ldots$, in contradiction to σ being an $(n, 3)$-DS sequence.

Therefore, one can remove at most two characters from σ' to obtain a $(n-1, 3)$-DS sequence $\tilde{\sigma}$. As the length of $\tilde{\sigma}$ is bounded by $\lambda_3(n-1)$, we obtain $\lambda_3(n) \leqslant \lambda_3(n-1) + \lambda_3(n)/n + 2$. Reformulating yields

$$\underbrace{\frac{\lambda_3(n)}{n}}_{=:f(n)} \leqslant \underbrace{\frac{\lambda_3(n-1)}{n-1}}_{=f(n-1)} + \frac{2}{n-1} \leqslant \underbrace{1}_{=f(1)} + 2\sum_{i=1}^{n-1} \frac{1}{i} = 1 + 2H_{n-1}$$

and together with $2H_{n-1} < 1 + 2\log n$ we obtain $\lambda_3(n) \leqslant 2n(1 + \log n)$. □

Bounds for higher-order Davenport-Schinzel sequences. As we have seen, $\lambda_1(n)$ (no aba) and $\lambda_2(n)$ (no $abab$) are both linear in n. It turns out that for $s \geqslant 3$, $\lambda_s(n)$ is slightly superlinear in n (taking s fixed). The bounds are known almost exactly, and they involve the inverse Ackermann function $\alpha(n)$, a function that grows extremely slowly.

To define the inverse Ackermann function, we first define a hierarchy of functions $\alpha_1(n)$, $\alpha_2(n)$, $\alpha_3(n)$, ... where, for every fixed k, $\alpha_k(n)$ grows much more slowly than $\alpha_{k-1}(n)$:

We first let $\alpha_1(n) = \lceil n/2 \rceil$. Then, for each $k \geqslant 2$, we define $\alpha_k(n)$ to be the number of times we must apply α_{k-1}, starting from n, until we get a result not larger than 1. In other words, $\alpha_k(n)$ is defined recursively by:

$$\alpha_k(n) = \begin{cases} 0, & \text{if } n \leqslant 1; \\ 1 + \alpha_k(\alpha_{k-1}(n)), & \text{otherwise.} \end{cases}$$

Thus, $\alpha_2(n) = \lceil \log_2 n \rceil$, and $\alpha_3(n) = \log^* n$.

Now fix n, and consider the sequence $\alpha_1(n)$, $\alpha_2(n)$, $\alpha_3(n)$, ... For every fixed n, this sequence decreases rapidly until it settles at 3. We define $\alpha(n)$ (the inverse Ackermann function) as the function that, given n, returns the smallest k such that $\alpha_k(n)$ is at most 3:

$$\alpha(n) = \min\{k \mid \alpha_k(n) \leqslant 3\}.$$

We leave as an exercise to show that for every fixed k we have $\alpha_k(n) = o(\alpha_{k-1}(n))$ and $\alpha(n) = o(\alpha_k(n))$.

Coming back to the bounds for Davenport-Schinzel sequences, for $\lambda_3(n)$ (no $ababa$) it is known that $\lambda_3(n) = \Theta(n\alpha(n))$ [4]. In fact it is known that $\lambda_3(n) = 2n\alpha(n) \pm O(n\sqrt{\alpha(n)})$ [5, 6]. For $\lambda_4(n)$ (no $ababab$) we have $\lambda_4(n) = \Theta(n \cdot 2^{\alpha(n)})$ [3].

For higher-order sequences the known upper and lower bounds are almost tight, and they are of the form $\lambda_s(n) = n \cdot 2^{\text{poly}(\alpha(n))}$, where the degree of the polynomial in the exponent is roughly $s/2$ [3, 6].

Realizing DS sequences as lower envelopes. There exists a construction of a set of n segments in the plane whose lower-envelope sequence has length $\Omega(n\alpha(n))$. (In fact, the lower-envelope sequence has length $n\alpha(n) - O(n)$, with a leading coefficient of 1; it is an open problem to get a leading coefficient of 2, or prove that this is not possible.)

It is an open problem to construct a set of n parabolic arcs in the plane whose lower-envelope sequence has length $\Omega(n \cdot 2^{\alpha(n)})$.

Generalizations of DS sequences. Also generalizations of Davenport-Schinzel sequences have been studied, for instance, where arbitrary subsequences (not necessarily an alternating pattern) are forbidden. For a word σ and $n \in \mathbb{N}$ define $\mathrm{Ex}(\sigma, n)$ to be the maximum length of a word over $A = \{1, \ldots, n\}^*$ that does not contain a subsequence of the form σ. For example, $\mathrm{Ex}(ababa, n) = \lambda_3(n)$. If σ consists of two letters only, say a and b, then $\mathrm{Ex}(\sigma, n)$ is super-linear if and only if σ contains $ababa$ as a subsequence [1]. This highlights that the alternating forbidden pattern is of particular interest.

Exercise 14.5 *Prove that* $\lambda_2(n) = 2n - 1$.

Exercise 14.6 *Prove that* $\lambda_s(n)$ *is finite for all* s *and* n.

Exercise 14.7 *Show that every* (n, s)*-Davenport-Schinzel sequence can be realized as the lower envelope of* n *continuous functions from* $\mathbb{R}$ *to* $\mathbb{R}$*, every pair of which intersect at most* s *times.*

Exercise 14.8 *Show that every Davenport-Schinzel sequence of order two can be realized as a lower envelope of* n *parabolas.*

It is a direct consequence of the symmetry in the definition that the property of being a Davenport-Schinzel sequence is invariant under permutations of the alphabet. For instance, $\sigma = bcacba$ is a $(3, 3)$-DS sequence over $A = \{a, b, c\}$. Hence the permutation $\pi = (ab)$ induces a $(3, 3)$-DS sequence $\pi(\sigma) = acbcab$ and similarly $\pi' = (cba)$ induces another $(3, 3)$-DS sequence $\pi'(\sigma) = abcbac$.

When counting the number of Davenport-Schinzel sequences of a certain type we want to count *essentially distinct* sequences only. Therefore we call two sequences over a common alphabet A *equivalent* if and only if one can be obtained from the other by a permutation of A. Then two sequences are *distinct* if and only if they are not equivalent. A typical way to select a representative from each equivalence class is to order the alphabet and demand that the first appearance of a symbol in the sequence follows that order. For example, ordering $A = \{a, b, c\}$ alphabetically demands that the first occurrence of a precedes the first occurrence of b, which in turn precedes the first occurrence of c.

Exercise 14.9 *Let* P *be a convex polygon with* $n+1$ *vertices. Find a bijection between the triangulations of* P *and the set of pairwise distinct* $(n, 2)$*-Davenport-Schinzel sequences of maximum length* $(2n - 1)$*. It follows that the number of distinct*

maximum $(n, 2)$*-Davenport-Schinzel sequences is exactly* $C_{n-1} = \frac{1}{n}\binom{2n-2}{n-1}$*, which is the* $(n-1)$*-st Catalan number.*

Questions

63. *What is an* (n, s) *Davenport-Schinzel sequence and how does it relate to the lower envelope of real-valued continuous functions?* Give the precise definition and some examples. Explain the relationship to lower envelopes and how to apply the machinery to partial functions like line segments.

64. *What is the value of* $\lambda_1(n)$ *and* $\lambda_2(n)$*?*

65. *What is the asymptotic value of* $\lambda_3(n)$*,* $\lambda_4(n)$*, and* $\lambda_s(n)$ *for larger* s*?*

66. *What is the combinatorial complexity of the lower envelope of a set of* n *lines/parabolas/line segments?*

References

[1] Radek Adamec, Martin Klazar, and Pavel Valtr, Generalized Davenport-Schinzel sequences with linear upper bound. *Discrete Math.*, **108**, (1992), 219–229, URL http://dx.doi.org/10.1016/0012-365X(92)90677-8.

[2] Pankaj K. Agarwal and Micha Sharir, *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, New York, NY, 1995.

[3] Pankaj K. Agarwal, Micha Sharir, and Peter W. Shor, Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences. *J. Combin. Theory Ser. A*, **52**, 2, (1989), 228–274, URL http://dx.doi.org/10.1016/0097-3165(89)90032-0.

[4] Sergiu Hart and Micha Sharir, Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorica*, **6**, (1986), 151–177, URL http://dx.doi.org/10.1007/BF02579170.

[5] Martin Klazar, On the maximum lengths of Davenport-Schinzel sequences. In R. Graham et al., ed., *Contemporary Trends in Discrete Mathematics*, vol. 49 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 169–178, Amer. Math. Soc., Providence, RI, 1999.

[6] Gabriel Nivasch, Improved bounds and new techniques for Davenport-Schinzel sequences and their generalizations. In *Proc. 20th ACM-SIAM Sympos. Discrete Algorithms*, pp. 1–10, 2009, URL http://doi.acm.org/10.1145/1496770.1496771.

[7] Seth Pettie, Splay trees, Davenport-Schinzel sequences, and the deque conjecture. In *Proc. 19th ACM-SIAM Sympos. Discrete Algorithms*, pp. 1115–1124, 2008, URL http://doi.acm.org/10.1145/1347082.1347204.

Chapter 15

Epsilon Nets

15.1 Motivation

Here is our scenario for this chapter. We are given a set A of points in $\mathbb{R}^d$ and a family $\mathcal{R}$ of *ranges* $r \subseteq \mathbb{R}^d$, for example the set of all balls, halfspaces, or convex sets in $\mathbb{R}^d$. A is huge and probably not even known completely; similarly, $\mathcal{R}$ may not be accessible explicitly (in the examples above, it is an uncountable set). Still, we want to learn something about A and $\mathcal{R}$.

The situation is familiar, definitely, if we don't insist on the geometric setting. For example, let A be the set of consumers living in Switzerland, and let $\tilde{r}$ be the subset of consumers who frequently eat a certain food product, say Lindt chocolate. We have similar subsets for other food products, and together, they form the family of ranges $\mathcal{R}$.

If we want to learn something about $\tilde{r}$, e.g. the ratio $\frac{|\tilde{r}|}{|A|}$ (the fraction of consumers frequently eating Lindt chocolate), then we typically sample a subset S of A and see what portion of S lies in $\tilde{r}$. We want to believe that

$$\frac{|\tilde{r} \cap S|}{|S|} \quad \text{approximates} \quad \frac{|\tilde{r}|}{|A|},$$

and statistics tells us to what extent this is justified. In fact, consumer surveys are based on this approach: in our example, S is a sample of consumers who are being asked about their chocolate preferences. After this, the quantity $|\tilde{r} \cap S|/|S|$ is known and used to predict the "popularity" $|\tilde{r}|/|A|$ of Lindt chocolate among Swiss consumers.

In this chapter, we consider a different kind of approximation. Suppose that we are interested in the most popular food products in Switzerland, the ones which are frequently eaten by more than an ε-fraction of all consumers, for some fixed $0 \leqslant \varepsilon \leqslant 1$. The goal is to find a small subset N of consumers that "represent" all popular products. Formally, we want to find a set $N \subseteq A$ such that

$$\text{for all } r\text{:} \quad \frac{|r|}{|A|} > \varepsilon \quad \Rightarrow \quad r \cap N \neq \emptyset.$$

Such a subset is called an *epsilon net*. Obviously, $N = A$ is an epsilon net for all ε, but as already mentioned above, the point here is to have a *small* set N.

Epsilon nets are very useful in many contexts that we won't discuss here. But already in the food consumption example above, it is clear that a small representative set of consumers is a good thing to have; for example if you quickly need a statement about a particular popular food product, you know that you will find somebody in your representative set who knows the product.

The material of this chapter is classic and goes back to Haussler and Welzl [1].

15.2 Range spaces and ε-nets.

Here is the formal framework. Let X be a (possibly infinite) set and $\mathcal{R} \subseteq 2^X$. The pair $(X, \mathcal{R})$ is called a *range space*[1], with X its *points* and the elements of $\mathcal{R}$ its *ranges*.

Definition 15.1 *Let $(X, \mathcal{R})$ be a range space. Given $A \subseteq X$, finite, and $\varepsilon \in \mathbb{R}$, $0 \leqslant \varepsilon \leqslant 1$, a subset N of A is called an* ε-net of A (w.r.t. $\mathcal{R}$) *if*

$$\text{for all } r \in \mathcal{R}: \quad |r \cap A| > \varepsilon |A| \quad \Rightarrow \quad r \cap N \neq \emptyset \ .$$

This definition is easy to write down, but it is not so easy to grasp, and this is why we will go through a couple of examples below. Note that we have a slightly more general setup here, compared to the motivating Section 15.1 where we had $X = A$.

15.2.1 Examples

Typical examples of range spaces in our geometric context are

- $(\mathbb{R}, \mathcal{H}_1)$ with $\mathcal{H}_1 := \{(-\infty, a] \mid a \in \mathbb{R}\} \cup \{[a, \infty) \mid a \in \mathbb{R}\}$ (*half-infinite intervals*), and
- $(\mathbb{R}, \mathcal{I})$ with $\mathcal{I} := \{[a, b] \| \, a, b \in \mathbb{R}, a \leqslant b\}$ (*intervals*),

and higher-dimensional counter-parts

- $(\mathbb{R}^d, \mathcal{H}_d)$ with $\mathcal{H}_d$ the closed *halfspaces* in $\mathbb{R}^d$ bounded by hyperplanes,
- $(\mathbb{R}^d, \mathcal{B}_d)$ with $\mathcal{B}_d$ the closed *balls* in $\mathbb{R}^d$,
- $(\mathbb{R}^d, \mathcal{S}_d)$ with $\mathcal{S}_d$ the d-dimensional *simplices* in $\mathbb{R}^d$, and
- $(\mathbb{R}^d, \mathcal{C}_d)$ with $\mathcal{C}_d$ the *convex sets* in $\mathbb{R}^d$.

[1] In order to avoid confusion: A range space is nothing else but a set system, sometimes also called hypergraph. It is the context, where we think of X as points and $\mathcal{R}$ as ranges in some geometric ambient space, that suggests the name at hand.

ε-Nets w.r.t. $(\mathbb{R}, \mathcal{H}_1)$ are particularly simple to obtain. For $A \subseteq \mathbb{R}$, $N := \{\min A, \max A\}$ is an ε-net for every ε—it is even a 0-net. That is, there are ε-nets of size 2, independent from $|A|$ and ε.

The situation gets slightly more interesting for the range space $(\mathbb{R}, \mathcal{I})$ with intervals. Given ε and A with elements

$$a_1 < a_2 < \cdots < a_n \ ,$$

we observe that an ε-net must contain at least one element from any contiguous sequence $\{a_i, a_{i+1}, \ldots, a_{i+k-1}\}$ of $k > \varepsilon n$ (i.e. $k \geqslant \lfloor \varepsilon n \rfloor + 1$) elements in A. In fact, this is a necessary and sufficient condition for ε-nets w.r.t. intervals. Hence,

$$\{a_{\lfloor \varepsilon n \rfloor + 1}, a_{2\lfloor \varepsilon n \rfloor + 2}, \ldots\}$$

is an ε-net of size[2] $\left\lfloor \frac{n}{\lfloor \varepsilon n \rfloor + 1} \right\rfloor \leqslant \lceil \frac{1}{\varepsilon} \rceil - 1$. So while the size of the net depends now on ε, it is still independent of $|A|$.

15.2.2 No point in a large range.

Let us start with a simple exercise, showing that large ranges are easy to "catch". Assume that $|r \cap A| > \varepsilon |A|$ for some fixed r and ε, $0 \leqslant \varepsilon \leqslant 1$.

Now consider the set S obtained by drawing s elements uniformly at random from A (with replacement). We write

$$S \sim A^s,$$

indicating that S is chosen uniformly at random from the set A^s of s-element sequences over A.

What is the probability that $S \sim A^s$ fails to intersect with r, i.e. $S \cap r = \emptyset$? For $p := \frac{|r \cap A|}{|A|}$ (note $p > \varepsilon$) we get[3]

$$\text{prob}(S \cap r = \emptyset) = (1-p)^s < (1-\varepsilon)^s \leqslant e^{-\varepsilon s} \ .$$

That is, if $s = \frac{1}{\varepsilon}$ then this probability is at most $e^{-1} \approx 0.368$, and if we choose $s = \lambda \frac{1}{\varepsilon}$, then this probability decreases exponentially with λ: It is at most $e^{-\lambda}$.

For example, if $|A| = 10000$ and $|r \cap A| > 100$ (r contains more than 1% of the points in A), then a sample of 300 points is disjoint from r with probability at most $e^{-3} \approx 0.05$.

[2] The number L of elements in the set is the largest ℓ such that $\ell(\lfloor \varepsilon n \rfloor + 1) \leqslant n$, hence $L = \left\lfloor \frac{n}{\lfloor \varepsilon n \rfloor + 1} \right\rfloor$. Since $\lfloor \varepsilon n \rfloor + 1 > \varepsilon n$, we have $\frac{n}{\lfloor \varepsilon n \rfloor + 1} < \frac{1}{\varepsilon}$, and so $L < \frac{1}{\varepsilon}$, i.e. $L \leqslant \lceil \frac{1}{\varepsilon} \rceil - 1$.

[3] We make use of the inequality $1 + x \leqslant e^x$ for all $x \in \mathbb{R}$.

15.2.3 Smallest enclosing balls.

Here is a potential use of this for a geometric problem. Suppose A is a set of n points in $\mathbb{R}^d$, and we want to compute the smallest enclosing ball of A. In fact, we are willing to accept some mistake, in that, for some given ε, we want a small ball that contains all but at most εn points from A. So let's choose a sample S of $\lambda\frac{1}{\varepsilon}$ points drawn uniformly (with replacement) from A and compute the smallest enclosing ball B of S. Now let $r := \mathbb{R}^d \setminus B$, the complement of B in $\mathbb{R}^d$, play the role of the range in the analysis above. Obviously $r \cap S = \emptyset$, so it is unlikely that $|r \cap A| > \varepsilon|A|$, since—if so—the probability of $S \cap r = \emptyset$ was at most $e^{-\lambda}$.

It is important to understand that this was complete nonsense!

For the probabilistic analysis above we have to first choose r and then draw the sample—and not, as done in the smallest ball example, first draw the sample and then choose r *based on the sample*. That cannot possibly work, since we could always choose r simply as the complement $\mathbb{R}^d \setminus S$—then clearly $r \cap S = \emptyset$ and $|r \cap A| > \varepsilon|A|$, unless $|S| \geqslant (1-\varepsilon)|A|$.

While you hopefully agree on this, you might find the counterargument with $r = \mathbb{R}^d \setminus S$ somewhat artificial, e.g. complements of balls cannot be that selective in 'extracting' points from A. It is exactly the purpose of this chapter to understand to what extent this advocated intuition is justified or not.

15.3 Either almost all is needed or a constant suffices.

Let us reveal the spectrum of possibilities right away, although its proof will have to await some preparatory steps.

Theorem 15.2 *Let $(X, \mathcal{R})$ be an infinite range space. Then one of the following two statements holds.*

(1) For every $n \in \mathbb{N}$ there is a set $A_n \subseteq X$ with $|A_n| = n$ such that for every ε, $0 \leqslant \varepsilon \leqslant 1$, an ε-net must have size at least $(1-\varepsilon)n$.

(2) There is a constant δ depending on $(X, \mathcal{R})$, such that for every finite $A \subseteq X$ and every ε, $0 < \varepsilon \leqslant 1$, there is an ε-net of A w.r.t. $\mathcal{R}$ of size at most $\frac{8\delta}{\varepsilon}\log_2\frac{4\delta}{\varepsilon}$ (independent of the size of A).

That is, either we have always ε-nets of size independent of $|A|$, or we have to do the trivial thing, namely choosing all but εn points for an ε-net. Obviously, the range spaces $(\mathbb{R}, \mathcal{H}_1)$ and $(\mathbb{R}, \mathcal{I})$ fall into category (2) of the theorem.

For an example for (1), consider $(\mathbb{R}^2, \mathcal{C}_2)$. For any $n \in \mathbb{N}$, let A_n be a set of n points in convex position. For every $N \subseteq A_n$ there is a range $r \in \mathcal{C}_2$, namely the convex hull of $A_n \setminus N$, such that $A_n \cap r = A_n \setminus N$ (hence, $r \cap N = \emptyset$); see Figure 15.1. Therefore,

$N \subseteq A_n$ cannot be an ε-net of A_n w.r.t. $\mathcal{C}_2$ if $|A_n \setminus N| = n - |N| > \varepsilon n$. Consequently, an ε-net must contain at least $n - \varepsilon n = (1-\varepsilon)n$ points.[4]

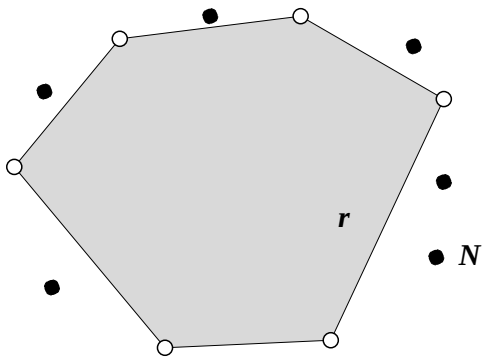


Figure 15.1: *If $\mathcal{R}$ consists of all convex sets in the plane, then only trivial epsilon nets exist: for every subset N (black points) of a set A_n in convex position, the range $r = \mathrm{conv}(A_n \setminus N)$ fails to intersect N.*

So what distinguishes $(\mathbb{R}^2, \mathcal{C}_2)$ from $(\mathbb{R}, \mathcal{H}_1)$ and $(\mathbb{R}, \mathcal{I})$? And which of the two cases applies to the many other range spaces we have listed above? Will all of this eventually tell us something about our attempt of computing a small ball covering all but at most εn out of n given points? This and more should be clear by the end of this chapter.

15.4 What makes the difference: VC-dimension

Given a range space $(X, \mathcal{R})$ and $A \subseteq X$, we let

$$\mathcal{R}\big|_A := \{r \cap A \mid r \in \mathcal{R}\},$$

the *projection of $\mathcal{R}$ to A*. Even if $\mathcal{R}$ is infinite, $\mathcal{R}\big|_A$ is always of size at most 2^n if A is an n-element set. The significance of projections in our context becomes clear if we rewrite Definition 15.1 in terms of projections: $N \subseteq A$ is an ε-net if

$$\text{for all } r \in \mathcal{R}\big|_A\text{:} \quad |r| > \varepsilon|A| \quad \Rightarrow \quad r \cap N \neq \emptyset.$$

All of a sudden, the conditions for an ε-net have become discrete, and they only depend on the finite range space $(A, \mathcal{R}\big|_A)$.

[4] If we were satisfied with any abstract example for category (1), we could have taken $(X, 2^X)$ for any infinite set X.

Note that, for A a set of n points in convex position in the plane, $\mathcal{C}_2\big|_A = 2^A$; we get every subset of A by an intersection with a convex set (this is also the message of Figure 15.1). That is $|\mathcal{C}_2\big|_A| = 2^n$, the highest possible value.

For A a set of n points in $\mathbb{R}$, we can easily see that[5] $|\mathcal{I}\big|_A| = \binom{n+1}{2} + 1 = O(n^2)$. A similar argument shows that $|\mathcal{H}_1\big|_A| = 2n$. Now comes the crucial definition.

Definition 15.3 *Given a range space $(X, \mathcal{R})$, a subset A of X is* shattered by $\mathcal{R}$ *if $\mathcal{R}\big|_A = 2^A$. The* VC-dimension[6] of $(X, \mathcal{R})$, $\mathrm{VCdim}(X, \mathcal{R})$, *is the cardinality (possibly infinite) of the largest subset of X that is shattered by $\mathcal{R}$. If no set is shattered (i.e. not even the empty set which means that $\mathcal{R}$ is empty), we set the VC-dimension to -1.*

We had just convinced ourselves that $(\mathbb{R}^2, \mathcal{C}_2)$ has arbitrarily large sets that can be shattered. Therefore, $\mathrm{VCdim}(\mathbb{R}^2, \mathcal{C}_2) = \infty$.

Consider now $(\mathbb{R}, \mathcal{I})$. Two points $A = \{a, b\}$ can be shattered, since for each of the 4 subsets, $\emptyset$, $\{a\}$, $\{b\}$, and $\{a, b\}$, of A, there is an interval that generates that subset by intersection with A. However, for $A = \{a, b, c\}$ with $a < b < c$ there is no interval that contains a and c but not b. Hence, $\mathrm{VCdim}(\mathbb{R}, \mathcal{I}) = 2$.

Exercise 15.4 *What is $\mathrm{VCdim}(\mathbb{R}, \mathcal{H}_1)$?*

Exercise 15.5 *Prove that if $\mathrm{VCdim}(X, \mathcal{R}) = \infty$, then we are in case (1) of Theorem 15.2, meaning that only trivial epsilon nets always exist.*

15.4.1 The size of projections for finite VC-dimension.

Here is the (for our purposes) most important consequence of finite VC dimension: there are only polynomially many ranges in every projection.

Lemma 15.6 (Sauer's Lemma) *If $(X, \mathcal{R})$ is a range space of finite VC-dimension at most δ, then*

$$|\mathcal{R}\big|_A| \leqslant \Phi_\delta(n) := \sum_{i=0}^{\delta} \binom{n}{i}$$

for all $A \subseteq X$ with $|A| = n$.

[5] Given A as $a_1 < a_2 \cdots < a_n$ we can choose another $n+1$ points b_i, $0 \leqslant i \leqslant n$, such that

$$b_0 < a_1 < b_1 < a_2 < b_2 < \cdots b_{n-1} < a_n < b_n \ .$$

Each nonempty intersection of A with an interval can be uniquely written as $A \cap [b_i, b_j]$ for $0 \leqslant i < j \leqslant n$. This gives $\binom{n+1}{2}$ plus one for the empty set.

[6] 'VC' in honor of the Russian statisticians V. N. Vapnik and A. Ya. Chervonenkis, who discovered the crucial role of this parameter in the late sixties.

Proof. First let us observe that $\Phi : \mathbb{N}_0 \cup \{-1\} \times \mathbb{N}_0 \to \mathbb{N}_0$ is defined by the recurrence[7]

$$\Phi_\delta(n) = \begin{cases} 0 & \delta = -1, \\ 1 & n = 0 \text{ and } \delta \geqslant 0, \text{ and} \\ \Phi_\delta(n-1) + \Phi_{\delta-1}(n-1) & \text{otherwise.} \end{cases}$$

Second, we note that the VC-dimension cannot increase by passing from $(X, \mathcal{R})$ to a projection (A, R), $R := \mathcal{R}\big|_A$. Hence, it suffices to consider the finite range space (A, R)—which is of VC-dimension at most δ—and show $|R| \leqslant \Phi_\delta(n)$ (since Φ is monotone in δ).

Now we proceed to a proof by induction of this inequality. If $A = \emptyset$ or $R = \emptyset$ the statement is trivial. Otherwise, we consider the two 'derived' range spaces for some fixed $x \in A$:

$$(A \setminus \{x\}, R - x), \qquad \text{with } R - x := \{r \setminus \{x\} \,|\, r \in R\}$$

(note $R - x = R\Big|_{A \setminus \{x\}}$) and

$$(A \setminus \{x\}, R^{(x)}), \qquad \text{with } R^{(x)} := \{r \in R \,|\, x \notin r, r \cup \{x\} \in R\}.$$

Observe that the ranges in $R^{(x)}$ are exactly those ranges in $R - x$ that have two preimages under the map

$$R \ni \quad r \quad \mapsto \quad r \setminus \{x\} \quad \in R - x \,,$$

all other ranges have a unique preimage. Consequently,

$$|R| = |R - x| + |R^{(x)}| \,.$$

We have $|R - x| \leqslant \Phi_\delta(n-1)$. If $A' \subseteq A \setminus \{x\}$ is shattered by $R^{(x)}$, then $A' \cup \{x\}$ is shattered by R. Hence, $(A \setminus \{x\}, R^{(x)})$ has VC-dimension at most $\delta - 1$ and $|R^{(x)}| \leqslant \Phi_{\delta-1}(n-1)$. Summing up, it follows that

$$|R| \leqslant \Phi_\delta(n-1) + \Phi_{\delta-1}(n-1) = \Phi_\delta(n)$$

which yields the assertion of the lemma. □

In order to see that the bound given in the lemma is tight, consider the range space

$$(X, \bigcup_{i=0}^{\delta} \binom{X}{i}) \,.$$

Obviously, a set of more than δ elements cannot be shattered (hence, the VC-dimension is at most δ), and for any finite $A \subseteq X$, the projection of the ranges to A is $\bigcup_{i=0}^{\delta} \binom{A}{\delta}$—with cardinality $\Phi_\delta(|A|)$.

[7] We recall that the binomial coefficients $\binom{n}{k}$ (with $k, n \in \mathbb{N}_0$) satisfy the recurrence $\binom{n}{k} = 0$ if $n < k$, $\binom{n}{0} = 1$, and $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$.

We note that a rough, but for our purposes good enough estimate for Φ is given by[8]

$$\Phi_\delta(n) \leqslant n^\delta \quad \text{for } \delta \geqslant 2.$$

We have seen now that the maximum possible size of projections either grows exponentially (2^n in case of infinite VC-dimension) or it is bounded by a polynomial n^δ in case of finite VC-dimension δ). The latter is the key to the existence of small ε-nets. Before shedding light on this, let us better understand when the VC-dimension is finite.

15.5 VC-dimension of Geometric Range Spaces

15.5.1 Halfspaces.

Let us investigate the VC-dimension of $(\mathbb{R}^2, \mathcal{H}_2)$. It is easily seen that three points in the plane can be shattered by halfplanes, as long a they do not lie on a common line. Hence, the VC-dimension is at least 3. Now consider 4 points. If three of them lie on a common line, there is no way to separate the middle point on this line from the other two by a halfplane. So let us assume that no three points lie on a line. Either three of them are vertices of a triangle that contains the fourth point—then we cannot possibly separate the fourth point from the remaining three points by a halfplane. Or the four points are vertices of a convex quadrilateral—then there is no way of separating the endpoints from a diagonal from the other two points. Consequently, four points cannot be shattered, and $\mathrm{VCdim}(\mathbb{R}^2, \mathcal{H}_2) = 3$ is established.

The above argument gets tedious in higher dimensions, if it works in a rigorous way at all. Fortunately, we can employ a classic.[9]

Lemma 15.7 (Radon's Theorem) *Let $d \in \mathbb{N}$. Every set A of $n \geqslant d+2$ points in $\mathbb{R}^d$ can be partitioned as $A = A_1 \dot\cup A_2$ such that $\mathrm{conv}(A_1) \cap \mathrm{conv}(A_2) \neq \emptyset$.*

Proof. For $n \geqslant d+2$, n points $\{p_1, p_2, \ldots, p_n\}$ in $\mathbb{R}^d$ are affinely dependent, which—by definition—means that there are real coefficients λ_i, $i \in \{1, 2, \ldots, n\}$, not all 0, with

$$\sum_{i=1}^{n} \lambda_i = 0 \quad \text{and} \quad \sum_{i=1}^{n} \lambda_i p_i = 0 \ .$$

Now let $I_1 := \{i \,|\, \lambda_i > 0\}$ and $I_2 := \{i \,|\, \lambda_i \leqslant 0\}$, which both have to be non-empty. We have $\sum_{i\in I_1} \lambda_i = -\sum_{i\in I_2} \lambda_i$, and we let λ denote this value (which has to be positive). Now

$$p := \frac{1}{\lambda} \sum_{i\in I_1} \lambda_i p_i = -\frac{1}{\lambda} \sum_{i\in I_2} \lambda_i p_i$$

[8] A better estimate, at least for $\delta \geqslant 3$, is given by $\Phi_\delta(n) < \left(\frac{e\,n}{\delta}\right)^d$ for all $n, d \in \mathbb{N}$, $d \leqslant n$.

[9] Radon's Theorem forms a classical triumvirate with Carathéodory's Theorem 3.8 and Helly's Theorem 12.6.

denotes a point that lies in the convex hulls of $A_1 := \{p_i \mid i \in I_1\}$ and $A_2 := \{p_i \mid i \in I_2\}$, respectively. Therefore, A_1 and A_2 provide a partition as required in the lemma. □

We get, as an easy implication, that a set A of at least $d+2$ points in $\mathbb{R}^d$ cannot be shattered by halfspaces. Indeed, let $A_1 \dot\cup A_2$ be a partition as guaranteed by Randon's Lemma. Now every halfspace containing A_1 must contain at least one point of A_2, hence $h \cap A = A_1$ is impossible for a halfspace h and thus A is not shattered by $\mathcal{H}_d$. Moreover, it is easily seen that the vertex set of a d-dimensional simplex (there are $d+1$ vertices) can be shattered by halfspaces (each subset of the vertices forms a face of the simplex and can thus be separated from the rest by a hyperplane). We summarize that

$$\mathrm{VCdim}(\mathbb{R}^d, \mathcal{H}_d) = d+1 \ .$$

Let us now consider the range space $(\mathbb{R}^d, \check{\mathcal{H}}_d)$, where $\check{\mathcal{H}}_d$ denotes the set of all closed halfspaces below non-vertical hyperplanes[10]–we call these *lower halfspaces*. Since $\check{\mathcal{H}}_d \subseteq \mathcal{H}_d$, the VC-dimension of $(\mathbb{R}^d, \check{\mathcal{H}}_d)$ is at most $d+1$, but, in fact, it not too difficult to show

$$\mathrm{VCdim}(\mathbb{R}^d, \check{\mathcal{H}}_d) = d \ . \tag{15.8}$$

(Check this claim at least for $d = 2$.) This range space is a geometric example where the bound of Sauer's Lemma is attained. Indeed, for any set A of n points in $\mathbb{R}^d$ in general position[11], it can be shown that

$$|\check{\mathcal{H}}_d|_A| = \Phi_d(n) \ .$$

15.5.2 Balls.

It is easy to convince oneself that the VC-dimension of disks in the plane is 3: Three points not on a line can be shattered and four points cannot. Obviously not, if one of the points is in the convex hull of the other, and for four vertices of a convex quadrilateral, it is not possible for both diagonals to be separated from the endpoints of the respective other diagonal by a circle (if you try to draw a picture of this, you see that you get two circles that intersect four times, which we know is not be the case).

A more rigorous argument which works in all dimensions is looming with the help of (15.8), if we employ the following transformation called *lifting map* that we have already encountered for $d = 2$ in Section 6.3:

$$\begin{array}{rcl} \mathbb{R}^d & \longrightarrow & \mathbb{R}^{d+1} \\ (x_1, x_2, \ldots, x_d) = p & \mapsto & \ell(p) = (x_1, x_2, \ldots, x_d,\ {x_1}^2 + {x_2}^2 + \cdots + {x_d}^2) \end{array}$$

[10] A hyperplane is called non-vertical if it can be specified by a linear equation $\sum_{i=1}^{d} \lambda_i x_i = \lambda_{d+1}$ with $\lambda_d \neq 0$; see also Section 1.2

[11] No $i+2$ on a common i-flat for $i \in \{1, 2, \ldots, d-1\}$; in particular, no $d+1$ points on a common hyperplane.

(For a geometric interpretation, this is a vertical projection of $\mathbb{R}^d$ to the unit paraboloid $x_{d+1} = {x_1}^2 + {x_2}^2 + \cdots + {x_d}^2$ in $\mathbb{R}^{d+1}$.) The remarkable property of this transformation is that it maps balls in $\mathbb{R}^d$ to halfspaces in $\mathbb{R}^{d+1}$ in the following sense.

Consider a ball $B_d(c, \rho)$ ($c = (c_1, c_2, \ldots, c_d) \in \mathbb{R}^d$ the center, and $\rho \in \mathbb{R}^+$ the radius). A point $p = (x_1, x_2, \ldots, x_d)$ lies in this ball iff

$$\sum_{i=1}^{d} (x_i - c_i)^2 \leqslant \rho^2 \Leftrightarrow \sum_{i=1}^{d} ({x_i}^2 - 2x_i c_i + {c_i}^2) \leqslant \rho^2$$

$$\Leftrightarrow \quad \left(\sum_{i=1}^{d} (-2c_i) x_i \right) + ({x_1}^2 + {x_2}^2 + \cdots + {x_d}^2) \leqslant \rho^2 - \sum_{i=1}^{d} {c_i}^2 \ ;$$

this equivalently means that $\ell(p)$ lies below the non-vertical hyperplane (in $\mathbb{R}^{d+1}$)

$$h = h(c, \rho) = \{x \in \mathbb{R}^{d+1} \mid \sum_{i=1}^{d+1} h_i x_i = h_{d+2}\} \ \text{ with}$$

$$(h_1, h_2, \ldots, h_d, h_{d+1}, h_{d+2}) = \left((-2c_1), (-2c_2), \ldots, (-2c_d), 1, \rho^2 - \sum_{i=1}^{d} {c_i}^2 \right) \ .$$

It follows that a set $A \subseteq \mathbb{R}^d$ is shattered by $\mathcal{B}_d$ (the set of closed balls in $\mathbb{R}_d$) iff $\ell(A) := \{\ell(p) \mid p \in A\}$ is shattered by $\check{\mathcal{H}}_{d+1}$. Assuming (15.8), this readily yields

$$\mathrm{VCdim}(\mathbb{R}^d, \mathcal{B}_d) = d + 1 \ .$$

The lifting map we have employed here is a special case of a more general paradigm called *linearization* which maps non-linear conditions to linear conditions in higher dimensions.

We have clarified the VC-dimension for all examples of range spaces that we have listed in Section 15.2.1, except for the one involving simplices. Before we elaborate on this, let us prove a first bound on the size of ε-nets when the VC-dimension is finite.

15.6 Small ε-Nets, an Easy Warm-up Version

Theorem 15.9 *Let $n, d \in \mathbb{N}$, $d \geqslant 2$, $\varepsilon \in \mathbb{R}^+$. Let $(X, \mathcal{R})$ be a range space of VC-dimension d. If $A \subseteq X$ with $|A| = n$, then there exists an ε-net N of A w.r.t. $\mathcal{R}$ with $|N| \leqslant \lceil \frac{d \ln n}{\varepsilon} \rceil$.*

Proof. We restrict our attention to the finite projected range space (A, R), $R := \mathcal{R}|_A$, for which we know $|R| \leqslant \Phi_d(n) \leqslant n^d$. It suffices to show that there is a set $N \subseteq A$ with $|N| \leqslant \frac{d \ln n}{\varepsilon}$ which contains an element from each $r \in R_\varepsilon := \{r \in R \mid |r| > \varepsilon n\}$.

Suppose, for some $s \in \mathbb{N}$ (to be determined), we let $N \sim A^s$. For each $r \in R_\varepsilon$, we know that $\text{prob}(r \cap N = \emptyset) < (1-\varepsilon)^s \leqslant e^{-\varepsilon s}$. Therefore,

$$\begin{aligned}\text{prob}(N \text{ is not } \varepsilon\text{-net of } A) &= \text{prob}(\exists r \in R_\varepsilon : r \cap N = \emptyset) \\ &= \text{prob}(\bigvee_{r \in R_\varepsilon} (r \cap N = \emptyset)) \\ &\leqslant \sum_{r \in R_\varepsilon} \text{prob}(r \cap N = \emptyset) < |R_\varepsilon| e^{-\varepsilon s} \leqslant n^d e^{-\varepsilon s} \ .\end{aligned}$$

It follows that if s is chosen so that $n^d e^{-\varepsilon s} \leqslant 1$, then $\text{prob}(N \text{ is not } \varepsilon\text{-net of } A) < 1$ and there remains a positive probability for the event that N is an ε-net of A. Now

$$n^d e^{-\varepsilon s} \leqslant 1 \quad \Leftrightarrow \quad n^d \leqslant e^{\varepsilon s} \quad \Leftrightarrow \quad d \ln n \leqslant \varepsilon s.$$

That is, for $s = \lceil \frac{d \ln n}{\varepsilon} \rceil$, the probability of obtaining an ε-net is positive, and therefore an ε-net of that size has to exist.[12] □

If we are willing to invest a little more in the size of the random sample N, then the probability of being an ε-net grows dramatically. More specifically, for $s = \lceil \frac{d \ln n + \lambda}{\varepsilon} \rceil$, we have

$$n^d e^{-\varepsilon s} \leqslant n^d e^{-d \ln n - \lambda} = e^{-\lambda} \ ,$$

and, therefore, a sample of that size is an ε-net with probability at least $1 - e^{-\lambda}$.

We realize that we need $\frac{d \ln n}{\varepsilon}$ sample size to compensate for the (at most) n^d subsets of A which we have to hit—it suffices to ensure positive success probability. The extra $\frac{\lambda}{\varepsilon}$ allows us to boost the success probability.

Also note that if A were shattered by $\mathcal{R}$, then $R = 2^A$ and $|R| = 2^n$. Using this bound instead of n^d in the proof above would require us to choose s to be roughly $\frac{n \ln 2}{\varepsilon}$, a useless estimate which even exceeds n unless ε is large (at least $\ln 2 \approx 0.69$).

15.6.1 Smallest enclosing balls, again

It is time to rehabilitate ourselves a bit concerning the suggested procedure for computing a small ball containing all but at most $\varepsilon|A|$ points from an n-point set $A \subseteq \mathbb{R}^d$.

Let $(\mathbb{R}^d, \mathcal{B}_d{}^{\text{compl}})$ be the range space whose ranges consist of all complements of closed balls in $\mathbb{R}^d$. This has the same VC-dimension $d+1$ as $(\mathbb{R}^d, \mathcal{B}_d)$. Indeed, if $A \cap r = A'$ for a ball r, then $A \cap (\mathbb{R}^d \setminus r) = A \setminus A'$, so A is shattered by $\mathcal{B}_d$ if and only if A is shattered by $\mathcal{B}_d{}^{\text{compl}}$.

Hence, if we choose a sample N of size $s = \lceil \frac{(d+1) \ln n + \lambda}{\varepsilon} \rceil$ then, with probability at least $1 - e^{-\lambda}$ this is an ε-net for A w.r.t. $\mathcal{B}_d{}^{\text{compl}}$. Let us quickly recall what this

[12] This line of argument "If an experiment produces a certain object with positive probability, then it has to exist", as trivial as it is, admittedly needs some time to digest. It is called *The Probabilistic Method*, and was used and developed to an amazing extent by the famous Hungarian mathematician Paul Erdős starting in the thirties.

means: whenever the complement of some ball B has empty intersection with N, then this complement contains at most an $\varepsilon|A|$ points of A. As a consequence, the smallest ball enclosing N has at most $\varepsilon|A|$ points of A outside, with probability at least $1 - e^{-\lambda}$.

15.7 Even Smaller ε-Nets

We still need to prove part 2 of Theorem 15.2, the existence of ε-nets whose size is independent of A. For this, we employ the same strategy as in the previous section, i.e. we sample elements from A uniformly at random, with replacement; a refined analysis will show that—compared to the bound of Theorem 15.9—much less elements suffice. Here is the main technical lemma.

Lemma 15.10 *Let $(X, \mathcal{R})$ be a range space of VC-dimension $\delta \geqslant 2$, and let $A \subseteq X$ be finite. If $x \sim A^m$ for $m \geqslant 8/\varepsilon$, then for the set N_x of elements occurring in x, we have*

$$\text{prob}(N_x \text{ is not an } \varepsilon\text{-net for } A \text{ w.r.t. } \mathcal{R}) \leqslant 2\Phi_\delta(2m)2^{-\frac{\varepsilon m}{2}}.$$

Before we prove this, let us derive the bound of Theorem 15.2 from it. We want that

$$2\Phi_\delta(2m)2^{-\frac{\varepsilon m}{2}} < 1,$$

since then we know that an ε-net of size m exists. We have

$$\begin{aligned}
& 2\Phi_\delta(2m)2^{-\frac{\varepsilon m}{2}} < 1 \\
\Leftarrow\ & 2(2m)^\delta < 2^{\frac{\varepsilon m}{2}} \\
\Leftrightarrow\ & 1 + \delta \log_2(2m) < \frac{\varepsilon m}{2} \\
\Leftarrow\ & 2\delta \log_2 m < \frac{\varepsilon m}{2} \\
\Leftrightarrow\ & \frac{4\delta}{\varepsilon} < \frac{m}{\log_2 m}.
\end{aligned}$$

In the second to last implication, we have used $\delta, m \geqslant 2$. Now we claim that the latter inequality is satisfied for $m = 2\frac{4\delta}{\varepsilon}\log_2\frac{4\delta}{\varepsilon}$. To see this, we need that $\frac{m}{\log_2 m} > \alpha$ for $m = 2\alpha\log_2\alpha$ and $\alpha = \frac{4\delta}{\varepsilon}$. We compute

$$\frac{m}{\log_2 m} = \frac{2\alpha\log_2\alpha}{\log_2(2\alpha\log_2\alpha)} = \frac{2\alpha\log_2\alpha}{1 + \log_2\alpha + \log_2\log_2\alpha} = \alpha\frac{2\log_2\alpha}{1 + \log_2\alpha + \log_2\log_2\alpha} \geqslant \alpha$$

as long as

$$\log_2\alpha \geqslant 1 + \log_2\log_2\alpha \Leftrightarrow \log_2\alpha \geqslant \log_2 2\log_2\alpha \Leftrightarrow \alpha \geqslant 2\log_2\alpha,$$

which holds as long as $\alpha \geqslant 2$, and this is satisfied for $\alpha = \frac{4\delta}{\varepsilon}$.

Proof. (Lemma 15.10) By going to the projection $(A, \restriction_{\mathcal{R}} A)$, we may assume that $X = A$, so we have a finite range space over n elements (see also Section 15.4). Fix $\varepsilon \in \mathbb{R}^+$. For $t \in \mathbb{N}$, a range $r \in \mathcal{R}$ and $x \in A^t$ (a t-vector of elements from A), we define

$$\text{count}(r, x) = |\{i \in [t] : x_i \in r\}|.$$

Now we consider two events over A^{2m}. The first one is the bad event of not getting an ε-net when we choose m elements at random from A (plus another m elements that don't matter). Recall that $R_\varepsilon = \{r \in \mathcal{R} : |r| > \varepsilon n\}$.

$$Q := \{xy \in A^{2m} \mid \exists r \in R_\varepsilon : \text{count}(r, x) = 0\}$$

Thus $\text{prob}(Q) = \text{prob}(N_x \text{ is not } \varepsilon\text{-net})$ which is exactly what we want to bound.

The second auxiliary event looks somewhat weird.

$$J := \{xy \in A^{2m} \mid x \in A^m, y \in A^m, \exists r \in R_\varepsilon : \text{count}(r, x) = 0 \text{ and } \text{count}(r, y) \geqslant \frac{\varepsilon m}{2}\}.$$

This event satisfies $J \subseteq Q$ and contains pairs of sequences x and y with somewhat contradicting properties for some r. While x fails to contain any element from r, y has many elements from r.

Claim 1. $\text{prob}(J) \leqslant \text{prob}(Q) < 2\text{prob}(J)$.

The first inequality is a consequence of $J \subseteq Q$. To prove the second inequality, we show that

$$\frac{\text{prob}(J)}{\text{prob}(Q)} = \frac{\text{prob}(J \cap Q)}{\text{prob}(Q)} = \text{prob}(J \mid Q) \geqslant \frac{1}{2}.$$

So suppose that $xy \in Q$, with "witness" r, meaning that $r \in R_\varepsilon$ and $\text{count}(r, x) = 0$. We show that $xy \in J$ with probability at least $1/2$, for every fixed such x and y chosen randomly from A^m. This entails the claim.

The random variable $\text{count}(r, y)$ is a sum of m independent Bernoulli experiments with success probability $p := |r|/n > \varepsilon$ (thus expectation p and variance $p(1-p)$). Using linearity of expectation and variance (the latter requires independence of the experiments), we get

$$\begin{aligned} E(\text{count}(r, y)) &= pm, \\ \text{Var}(\text{count}(r, y)) &= p(1-p)m. \end{aligned}$$

Now we use Chebyshew's inequality[13] to bound the probability of the "bad" event that

[13] $\text{prob}(|X - E(X)| \geqslant k\text{Var}(X)) \leqslant \frac{1}{k^2 \text{Var}(X)}$

$\text{count}(r,y) < \frac{\varepsilon m}{2}$. We have

$$\begin{aligned}
& \text{prob}\left(\text{count}(r,y) < \frac{\varepsilon m}{2}\right) \\
\leqslant\ & \text{prob}\left(\text{count}(r,y) < \frac{pm}{2}\right) \\
\leqslant\ & \text{prob}\left(|\text{count}(r,y) - E(\text{count}(r,y))| > \frac{pm}{2}\right) \\
=\ & \text{prob}\left(|\text{count}(r,y) - E(\text{count}(r,y))| > \frac{1}{2(1-p)}\text{Var}(\text{count}(r,y))\right) \\
\leqslant\ & \frac{4(1-p)^2}{p(1-p)m} = \frac{4(1-p)}{pm} \leqslant \frac{4}{pm} < \frac{4}{\varepsilon m} \leqslant \frac{1}{2},
\end{aligned}$$

since $m \geqslant \frac{8}{\varepsilon}$. Hence

$$\text{prob}\left(\text{count}(r,y) \geqslant \frac{\varepsilon m}{2}\right) \geqslant \frac{1}{2},$$

and the claim is proved.

Now the weird event J reveals its significance: we can nicely bound its probability. The idea is this: We enumerate A as $A = \{a_1, a_2, \ldots, a_n\}$ and let the *type* of $z \in A^t$ be the n-sequence

$$\text{type}(z) = (\text{count}(a_1, z), \text{count}(a_2, z), \ldots, \text{count}(a_n, z)).$$

For example, the type of $z = (1,2,4,2,2,4)$ w.r.t. $A = \{1,2,3,4\}$ is $\text{type}(z) = (1,3,0,2)$.

Now fix an arbitrary type τ. We will bound the conditional probability $\text{prob}(xy \in J \mid xy \text{ has type } \tau)$, and the value that we get is independent of τ. It follows that the same value also bounds $\text{prob}(J)$.

Claim 2. $\text{prob}(xy \in J \mid xy \text{ has type } \tau) \leqslant 2\Phi_\delta(2m)2^{-\varepsilon m/2}$.

To analyze the probability in question, we need to sample $z = xy$ uniformly at random from all sequences of type τ. This can be done as follows: take an arbitrary sequence z' of type τ, and then apply a random permutation $\pi \in S_{2m}$ to obtain $z = \pi(z')$, meaning that

$$z_i = z'_{\pi(i)} \text{ for all } i.$$

Why does this work? First of all, π preserves the type, and it is easy to see that all sequences z of type τ can be obtained in this way. Now we simply count the number of permutations that map z' to a fixed z and see that this number only depends on the τ, so it is the same for all z. Indeed, for every element $a_i \in A$, there are $\tau_i!$ many ways of mapping the a_i's in z' to the a_i's in z. The number of permutations that map z' to z is therefore given by

$$\prod_{i=1}^{n} \tau_i!.$$

By these considerations,

$$\text{prob}(xy \in J \mid xy \text{ has type } \tau) = \text{prob}(\pi(z') \in J)\}.$$

To estimate this, we let S be the set of distinct elements in z' (this is also a function of the type τ). Since $(X, \mathcal{R})$ has VC-dimension δ, we know from Lemma 15.6 that $|\mathcal{R}|_S| \leqslant \Phi_\delta(2m)$, so at most that many different subsets T of S can be obtained by intersections with ranges $r \in \mathcal{R}$, and in particular with ranges $r \in R_\varepsilon$.

Now we look at some fixed such T, consider a permutation π and write $\pi(z') = xy$. We call T a *witness* for π if no element of x is in T, but at least $\varepsilon m/2$ elements of y are in T. According to this definition,

$$\pi(z') \in J \quad \Leftrightarrow \quad \pi \text{ has some witness } T \subseteq S.$$

By the union bound,

$$\text{prob}(xy \in J \mid xy \text{ has type } \tau) = \text{prob}(\pi(z') \in J) \leqslant \sum_T \text{prob}(T \text{ is a witness for } \pi).$$

Suppose that z' contains $\ell \geqslant \varepsilon m/2$ occurences of elements of T (for smaller ℓ, T cannot be a witness). The probability of T being a witness for a random permutation is then

$$\frac{\binom{m}{\ell}}{\binom{2m}{\ell}} = \frac{m(m-1)\cdots m(m-\ell+1)}{2m(2m-1)\cdots(2m-l+1)} \leqslant 2^{-\ell} \leqslant 2^{-\frac{\varepsilon m}{2}},$$

since among all the $\binom{2m}{\ell}$ equally likely ways in which π distributes the ℓ occurences in z, exactly the $\binom{m}{\ell}$ equally likely ways of putting them into y are good.[14] Summing up over all at most $\Phi_\delta(2m)$ sets T, we get

$$\text{prob}(xy \in J \mid xy \text{ has type } \tau) \leqslant \Phi_\delta(2m)2^{-\frac{\varepsilon m}{2}},$$

for all τ.

The proof is finished by combining the two claims:

$$\text{prob}(N_x \text{ is not } \varepsilon\text{-net}) = \text{prob}(Q) \leqslant 2\text{prob}(J) \leqslant 2\Phi_\delta(2m)2^{-\frac{\varepsilon m}{2}}.$$

□

Questions

67. *What is a range space?* Give a definition and a few examples.

68. *What is an epsilon net?* Provide a formal definition, and explain in words what it means.

[14] This argument can be made more formal by explicitly counting the permutations that map $\{1, 2, \ldots, \ell\}$ to the set $\{1, 2, \ldots, m\}$. We leave this to the reader.

69. *What is the VC dimension of a range space* Give a definition, and compute the VC dimension of a few example range spaces.

70. *Which range spaces always have small epsilon nets (and how small), and which ones don't?* Explain Theorem 15.2.

71. *How can you compute small epsilon nets?* Explain the general idea, and give the analysis behind the weaker bound of Theorem 15.9.

72. *How can you compute smaller epsilon nets?* Sketch the main steps of the proof of the stronger bound of Theorem 15.2.

References

[1] D. Haussler and Emo Welzl, Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, **2**, (1987), 127–151.