Chapter 15

Arrangements of line segments in the plane

In an arrangement of n lines in the plane, all the cells are convex and thus have complexity $O(n)$. Moreover, given a point A, the cell in the arrangement that contains A can be computed in time $\Theta(n \log n)$: indeed, the problem reduces to computing the intersection of n half-planes bounded by the lines and containing A (see theorem 7.1.10).

In this chapter, we study arrangements of line segments in the plane. Consider a set $\mathcal{S}$ of n line segments in the plane. The arrangement of $\mathcal{S}$ includes cells, edges, and vertices of the planar subdivision of the plane induced by $\mathcal{S}$, and their incidence relationships.

Computing the arrangement of $\mathcal{S}$ can be achieved in time $O(n \log n + k)$ where k is the number of intersection points (see sections 3.3 and 5.3.2, and theorem 5.2.5). All the pairs of segments may intersect, so in the worst case we have $k = \Omega(n^2)$.

For a few applications, only a cell in this arrangement is needed. This is notably the case in robotics, for a polygonal robot moving amidst polygonal obstacles by translation (see exercise 15.6). The reachable positions are characterized by lying in a single cell of the arrangement of those line segments that correspond to the set of positions of the robot when a vertex of the robot slides along the edge of an obstacle, or when the edge of a robot maintains contact with an obstacle at a point. Since the robot may not cross over an obstacle, it is constrained in always lying inside the same cell of this arrangement. It is therefore important to bound the complexity of such a cell and to avoid computing the whole arrangement. Among the cells of $\mathcal{A}(\mathcal{S})$, a few contain the endpoints of some segments, and the others do not. The latter are naturally convex cells, their complexity is $O(n)$ and each can be computed in time $O(n \log n)$. The complexity of the former cells, however, is more difficult to analyze.

To conduct the combinatorial analysis, we introduce and study a certain class of words over a finite alphabet, the so-called *Davenport–Schinzel sequences* (see section 15.2). These words have a geometric interpretation that is both illuminating and useful: lower envelopes of functions (see section 15.3). Section 15.4 bounds the complexity of a cell and gives an algorithm that computes it. We first show that this complexity is almost linear, in contrast with the entire arrangement which may have $\Omega(n^2)$ edges in the worst case. The complexity of the algorithm is shown to be roughly proportional to the complexity of the cell it computes.

15.1 Faces in an arrangement

Let $\mathcal{S}$ be a set of n segments in the plane. To define a cell of their arrangement, we need to distinguish the two sides of a segment, or equivalently to consider that each line segment is a flat rectangle with an infinitesimally small width. The arrangement of $\mathcal{S}$ is formed by cells, edges, and vertices of the planar subdivision induced by $\mathcal{S}$, and their incidence relationships. More precisely, the connected components of $\mathbb{E}^2 \setminus \mathcal{S}$ are polygonal regions that may have holes (see figure 15.1): the cells of the arrangement are formed by the topological closures of these regions. The edges and vertices of this arrangement are the edges and vertices of the polygons that bound the cell. The arrangement of $\mathcal{S}$ will be denoted by $\mathcal{A}(\mathcal{S})$.

15.2 Davenport–Schinzel sequences

Given an alphabet with n symbols, a *word* on this alphabet is an ordered sequence of symbols in this alphabet, and a subsequence of a word $u_1 \dots u_n$ is a word $u_{i_1} \dots u_{i_k}$ for some indices $1 \leq i_1 < \cdots < i_k \leq n$. Given two symbols a and b, an *alternating sequence* of length s is a sequence $u_1 \dots u_s$ such that $u_i = a$ if i is odd and $u_i = b$ is i is even. An (n, s)-*Davenport–Schinzel sequence* is a word on an alphabet with n symbols such that:

1. Two successive symbols of this word are distinct.

2. For each two symbols a, b in the alphabet, the alternating sequence of length $s + 2$ is not a subsequence of this word.

In other words, no two symbols can alternate more than $s + 1$ times.

So consider the phrase 'A DAVENPORT-SCHINZEL SEQUENCE', considered as a word over the Roman alphabet. The reader will easily verify that the longest alternating subsequence over two symbols is the subsequence

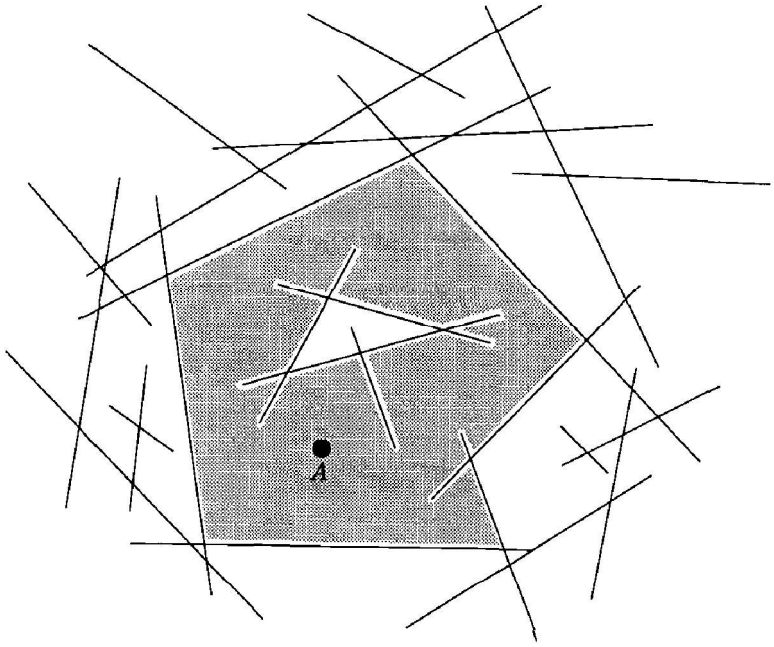

Figure 15.1. A cell in the arrangement of line segments.

'ESESE' (the other subsequences 'ENENE' and 'ECECE' are also suitable). The sequence 'A DAVENPORT-SCHINZEL SEQUENCE' is thus a $(26, 4)$-Davenport–Schinzel sequence!

Denote by $\lambda_s(n)$ the maximal length of an (n, s)-Davenport–Schinzel sequence. First of all, it is not even clear that $\lambda_s(n)$ is finite. In fact, it can be deduced from the connection with lower envelopes (see section 15.3) that $\lambda_s(n) \leq \frac{sn(n-1)}{2} + 1$. The following theorem gives more precise bounds on λ_1, λ_2, and λ_3.

Theorem 15.2.1 *The maximal length $\lambda_s(n)$ of an (n, s)-Davenport–Schinzel sequence is bounded by:*

$$\begin{aligned} \lambda_1(n) &= n \\ \lambda_2(n) &= 2n - 1 \\ \lambda_3(n) &= \Theta(n\alpha(n)) \end{aligned}$$

where $\alpha(n)$ is the very slow-growing inverse of Ackermann's function.[1]

Proof. The proof for $s = 1$ is trivial, since each symbol may appear only once. For $s = 2$, we proceed by induction on n. The result is true for $n = 1$, so we consider an $(n, 2)$-Davenport–Schinzel sequence ($n > 1$). Let a be its first letter,

[1]The definitions and order of magnitude of the inverse Ackermann function are given in subsection 1.1.3.

and put $S = aS'$. If a does not occur in S', then the induction applies for S' and so

$$|S| = 1 + |S'| \leq 1 + 2(n-1) - 1 = 2n - 2.$$

Otherwise we can write $S = aS_1aS_2$, where a does not occur in S_1 and $|S_1| > 0$. If S_2 is empty the length of aS_1a is smaller than $(2n-2)+1 = 2n-1$, as we have just shown. Otherwise, let k be the number of distinct symbols in S_1. By induction, $|S_1| \leq 2k-1$. Moreover, the definition of a Davenport–Schinzel sequence ensures that no symbol b occurs both in S_1 and S_2, otherwise $abab$ is a subsequence of S. Thus aS_2 may contain at most $n-k$ symbols (note that a may occur in S_2), and by induction we have $|aS_2| \leq 2(n-k)-1$. Hence

$$|S| = |S_1| + |aS_2| + 1 \leq 2n - 1.$$

To finish the proof for $s = 2$, we must also show that this bound is exact. This can be readily seen by considering the sequence $ab_1ab_2a \ldots ab_{n-1}a$ of length $2n-1$.

For $s = 3$, the proof goes into very technical details, so we will not prove the announced result here. We can show, however, the simpler result that $\lambda_3(n) = O(n \log n)$. Let S be a $(n,3)$-Davenport–Schinzel sequence, and $S(a)$ be the subsequence obtained from S by removing all the occurrences of a symbol a. In $S(a)$, there cannot be a subsequence $bcbcb$ and identical consecutive symbols can happen at most twice when the first and the last occurrences of a are surrounded by two b's. Let us call $S'(a)$ the sequence obtained by replacing in $S(a)$ two consecutive symbols b by a single b, whenever this happens. Then $S'(a)$ is an $(n-1,3)$-Davenport–Schinzel sequence, and

$$|S| \leq |S'(a)| + 2 + n_a \leq \lambda_3(n-1) + 2 + n_a$$

where n_a stands for the number of occurrences of a in S. Summing over all the symbols a appearing in S, we obtain:

$$n|S| \leq n\lambda_3(n-1) + 2n + |S|.$$

This is true for any sequence S, so that

$$\frac{\lambda_3(n)}{n} \leq \frac{\lambda_3(n-1)}{n-1} + \frac{2}{n-1}$$

whence $\lambda_3(n) = O(n \log n)$. □

15.3 The lower envelope of a set of functions

Consider n continuous functions $f_i(x), i = 1, \ldots, n$ defined over $\mathbb{R}$. The *lower envelope* of the f_i's is the graph of the function defined by

$$f(x) = \min_i f_i(x).$$

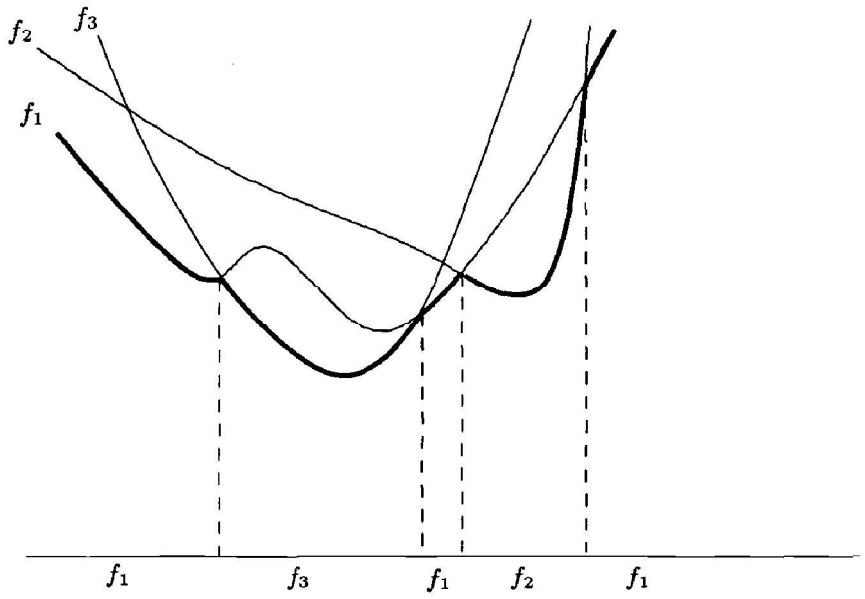

Figure 15.2. The lower envelope of a set of functions, and the corresponding Davenport–Schinzel sequence.

The lower envelope is formed by a sequence of *curved edges*, where each edge is a maximal connected subset of the envelope that belongs to the graph of a single function $f_i(x)$. The endpoints of these edges are located at the intersections of the graphs of the functions and are called the *vertices* of the envelope.

15.3.1 Complexity

Labeling each edge by the index of the corresponding function, we obtain a sequence of indices by enumerating these labels in the order in which they appear along the envelope (see figure 15.2). If the graphs of the functions have pairwise at most s intersection points, then this sequence is an (n, s)-Davenport–Schinzel sequence. Indeed, let A_i and A_j be two edges appearing in this order along the envelope, defined over two intervals I and J. The corresponding functions f_i and f_j being continuous, they must intersect in a point whose abscissa is greater than the right endpoint of I and smaller than the left endpoint of J. Having an alternating subsequence of length $s + 2$ for the two symbols i and j implies the existence of $s + 1$ intersection points between the graphs of f_i and f_j, a contradiction.

The number of edges on the lower envelope is thus bounded above by the maximal length $\lambda_s(n)$ of an (n, s)-Davenport–Schinzel sequence.

Consider the case when the functions are defined over closed intervals and not over the whole of $\mathbb{R}$. The lower envelope is not continuous and the argument used

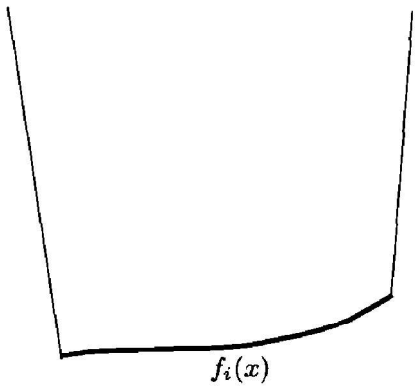

Figure 15.3. Extending the function f_i.

above to bound the number of its edges does not hold any more. This problem may be overcome by extending the domain of definition of the functions f_i to cover the whole of $\mathbb{R}$. More precisely, pick a positive real number μ. If f_i is defined over $[x_{i_{min}}, x_{i_{max}}]$, then we extend the graph of f_i for $x > x_{i_{max}}$ by the semi-infinite ray originating at $(x_{i_{max}}, f_i(x_{i_{max}}))$ whose slope is μ, and symmetrically for $x < x_{i_{min}}$ by the semi-infinite ray originating at $(x_{i_{min}}, f_i(x_{i_{min}}))$ whose slope is $-\mu$ (see figure 15.3). Thus we have a set of functions g_i which extend the functions f_i and are continuous. When μ is large enough, the sequence of labels of the edges on the lower envelope of the g_i's is identical to that of the lower envelope of the f_i's, and this lower envelope can be easily constructed knowing that of the g_i's.

It is readily verified that, for μ large enough, g_i and g_j have at most $s+2$ intersection points if the corresponding functions f_i and f_j intersect in at most s points. It follows that the sequence of labels of the edges on the lower envelope of $g_1, \ldots, g_n$ is a $(n, s+2)$-Davenport–Schinzel sequence.

The complexity of the lower envelope of the g_i's is thus bounded above by the maximal length $\lambda_{s+2}(n)$ of an $(n, s+2)$-Davenport–Schinzel sequence. The complexity of the lower envelope of the f_i's is also bounded by $\lambda_{s+2}(n)$.

Example. Consider the case of line segments. Two line segments intersect in at most one point, so the sequence of labels on the lower envelope of a set of segments is an $(n, 3)$-Davenport–Schinzel sequence. The complexity of this lower envelope is thus $O(n\alpha(n))$. In fact, this bound is achievable and one may actually construct line segments whose lower envelope has super-linear complexity $\Theta(n\alpha(n))$ (see the bibliographical notes at the end of this chapter).

Let us now consider the case when the functions f_i are only defined over semi-infinite intervals. We first consider the functions f_i whose domains of definition are intervals defined by $x \geq x_{i_{min}}$. If we extend these functions by a half-line starting at $(x_{i_{min}}, f_i(x_{i_{min}}))$ of slope $-\mu$ for μ big enough, then we obtain func-

tions g_i, defined over $\mathbb{R}$, whose graphs have pairwise at most $s+1$ intersection points if the graphs of the f_i's had pairwise at most s intersection points. The sequence of labels on the lower envelope $\mathcal{L}_r$ of the g_i's is an $(n, s+1)$-Davenport–Schinzel sequence. The complexity of $\mathcal{L}_r$ is thus $\lambda_{s+1}(n)$.

A similar result obviously holds for the lower envelope $\mathcal{L}_l$ of the functions f_j whose domains of definition are defined by $x < x_{j_{max}}$. The lower envelope of the n functions f_i is the lower envelope of the union of $\mathcal{L}_r$ and $\mathcal{L}_l$. Its complexity is $O(n_r + n_l) = O(\lambda_{s+1}(n))$ since both $\mathcal{L}_r$ and $\mathcal{L}_l$ are monotone chains.

Example. The lower envelope of n half-lines has complexity $O(n)$.

15.3.2 Computing the lower envelope

We now present an algorithm that computes the lower envelope of n functions f_i, $i = 1, ..., n$, defined over $\mathbb{R}$ such that no two graphs of these functions have more than s intersection points. The algorithm recursively computes the lower envelope $\mathcal{I}_1$ of $f_1(x), \ldots, f_{\lfloor n/2 \rfloor}$, and the lower envelope $\mathcal{I}_2$ of $f_{\lfloor n/2 \rfloor + 1}, \ldots, f_n(x)$. Both envelopes are monotone chains of complexity $\lambda_s(\frac{n}{2}) \leq \lambda_s(n)$, as was shown in the previous subsection. Monotonicity implies that we can compute the lower envelope of the union of $\mathcal{I}_1$ and $\mathcal{I}_2$ by sweeping the plane with a line parallel to the y-axis, in a manner that is similar to merging two sorted lists (see section 3.1.2). Let us call the current edges the two edges of $\mathcal{I}_1$ and $\mathcal{I}_2$ intersecting the sweep line. When the sweep line passes over a vertex of a current edge $\mathcal{I}_1$ or $\mathcal{I}_2$, this current edge is replaced by the edge that follows on the corresponding lower envelope. If this edge is part of the constructed lower envelope, a new edge is created for the lower envelope. When the sweep line encounters an intersection point between the two current edges, a new edge is created on the lower envelope. In either case, the next intersection point between the two current edges is computed. Merging the lower envelopes in this fashion takes time proportional to the total number of edges on $\mathcal{I}_1$ and $\mathcal{I}_2$, and to the number of intersection points between $\mathcal{I}_1$ and $\mathcal{I}_2$ which is $O(\lambda_s(n))$ as was shown in the previous section. We have thus proved that:

Theorem 15.3.1 *The lower envelope of n functions f_i, $i = 1, ..., n$, defined over $\mathbb{R}$ and whose graphs have pairwise at most s intersection points, has complexity $O(\lambda_s(n))$ and can be computed in time $O(\lambda_s(n) \log n)$.*

15.4 A cell in an arrangement of line segments

Let $\mathcal{S}$ be a set of n line segments in the plane. In the arrangement of $\mathcal{S}$, we may distinguish between cells whose boundaries contain at least one endpoint

of a segment (the *non-trivial* cells) and the cells whose boundaries contain no endpoints (the *trivial* cells). Trivial cells are convex and their complexity is $O(n)$. In section 15.3, we have seen that the complexity of the lower envelope of a set of n line segments in the plane can be $\Theta(n\alpha(n))$. So we can conclude that $\Omega(n\alpha(n))$ is a lower bound on the worst-case complexity of a non-trivial cell in the arrangement of n line segments. To show this, consider a set of n line segments whose lower envelope has complexity $\Theta(n\alpha(n))$. To $\mathcal{S}$, we add $2n$ segments, almost vertical, and long enough so that each of them stands above an endpoint of a segment in $\mathcal{S}$ (see figure 15.3). We also add a horizontal segment lying above all the segments in $\mathcal{S}$ while cutting all the almost vertical segments that we added. The new set of segments $\mathcal{S}'$ has $3n+1$ segments, and the edges on the boundary of the unbounded cell lying below all the segments are in one-to-one correspondence with the edges of the lower envelope of $\mathcal{S}'$. But the $\Omega(n\alpha(n))$ edges on the lower envelope of $\mathcal{S}$ also correspond to a subset of the edges on the lower envelope of $\mathcal{S}'$. It follows that the unbounded cell is at least as complex as the lower envelope of $\mathcal{S}$, so that it also has complexity $\Omega(n\alpha(n))$.

As we will see, this bound is also an upper bound, which shows that the complexity of cells in the arrangement of line segments depends almost linearly on the number of segments, while the total arrangement may have up to $\Omega(n^2)$ edges in the worst case. We will then explain how to efficiently compute such a cell.

15.4.1 Complexity

Consider a set $\mathcal{S}$ of n line segments in the plane. We will assume that these segments are in general position, meaning that no three segments have a common intersection and that any two segments intersect in at most one point. A standard perturbation argument shows that the complexity of a cell is maximal in this case. Indeed, if the segments are not in general position, one may perturb them slightly so that they are in general position, without decreasing the number of edges or vertices of the cell under consideration.

From now on, and as was done in section 15.1, we consider that each line segment S is a rectangle of infinitely small width whose boundary is formed by two copies of the segment S called the *sides* of S, and two infinitely short perpendicular segments at the vertices. Under the general position assumption, the boundary of the union of these rectangles is homeomorphic to the union of all the segments. Henceforth, we will thus make a distinction between a segment, considered as a infinitely thin rectangle, and a segment side. The number of sides is $2n$.

We orient the rectangles counter-clockwise, which induces a clockwise orientation for the connected components of the boundaries of each cell.

Let Γ be a connected component of the boundary of some cell C in the ar-

rangement $\mathcal{A}(\mathcal{S})$ of $\mathcal{S}$. Note that a segment may contain several edges of Γ.

Lemma 15.4.1 *Consider a segment S that contains at least one edge of Γ. The edges of Γ contained in S are traversed on the boundary of Γ in the same order as they are traversed on the boundary of S.*

Proof. Consider the infinitely thin rectangle S and the region R bounded by Γ that does not contain C. Then S is contained in R, and the result follows from a slight adaptation of the proof of theorem 9.4.1. □

We label each edge of Γ by the index of the side of the segment of $\mathcal{S}$ to which it belongs. The sequence Σ_Γ of these labels forms a circular sequence which we break into a linear sequence by choosing some origin O on Γ. The number of distinct labels in Σ_Γ is at most the number of sides, which is $2n$. Two successive labels are distinct. Since two segments have only one intersection point, it is tempting to conjecture that the sequence Σ_Γ is a $(2n, 3)$-Davenport–Schinzel sequence. The choice of O may induce some additional repeats, however. Indeed, if *ababab* is not a subsequence of the circular sequence, it may not always be possible to choose O so that the same is true for the linear sequence. For instance, consider figure 15.4: the linear sequence

$$\Sigma_\Gamma = a_1\ c_2\ c_1\ a_1\ a_2\ c_1\ b_1\ b_2\ c_1\ c_2\ b_2\ a_2\ a_1\ b_2\ b_1$$

does contain the subsequence $a_1c_1a_1c_1a_1$. We solve this technical problem by constructing another sequence Σ^*_Γ on at most $3n$ symbols which is at least as long as Σ_Γ. Let L be a side of a segment that supports several edges along Γ. These edges are naturally ordered by the orientation of L, so we let I be the first point of L that belongs to Γ and F the last point of L that belongs to Γ. The points I and F subdivide Γ into two chains ending at I and F. Denote by γ the oriented chain that contains the origin O. The idea is to give a different label to the edges on Γ that belong to $L \cap \gamma$ according to whether they are before O or after O. Then the new sequence Σ^*_Γ is merely the linear sequence of these new labels along the edges Γ. For instance, on figure 15.4, we now have

$$\Sigma^*_\Gamma = a''_1\ c_2\ c_1\ a''_1\ a_2\ c_1\ b''_1\ b_2\ c_1\ c_2\ b_2\ a_2\ a'_1\ b_2\ b'_1\ .$$

Σ^*_Γ has at most $3n$ distinct labels, since only one side of each segment needs to be relabeled.

Lemma 15.4.2 *Σ^*_Γ is a $(3n, 3)$-Davenport–Schinzel sequence.*

Proof. We already know that Σ^*_Γ has at most $3n$ distinct labels and does not contain two identical consecutive elements. It remains to see that *ababa* is not a subsequence of Σ^*_Γ for any two symbols $a \neq b$.

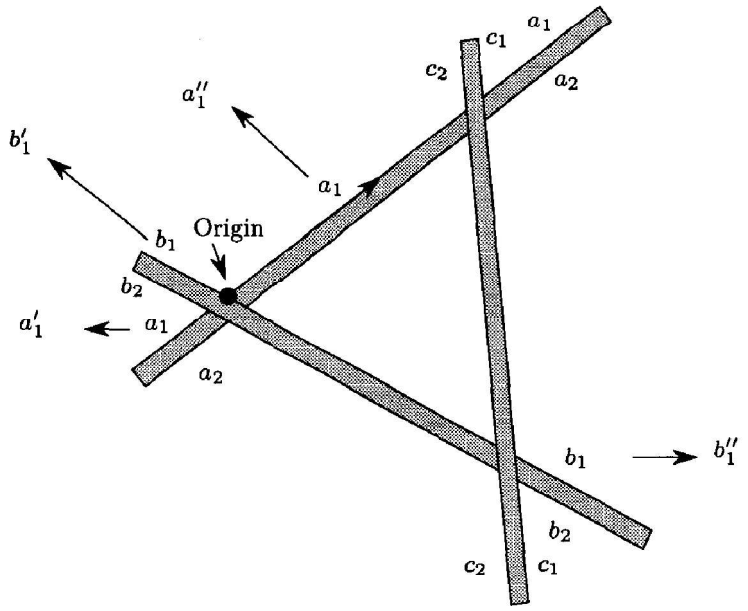

Figure 15.4. Circular and linear sequences.

We first show that, if *abab* is a subsequence of Σ_Γ^*, the sides labeled a and b must intersect. For this, let the subsequence *abab* correspond to the edges E_a', E_b', E_a'', E_b'' on Γ. Let S_a be the side labeled a that contains E_a' and E_a''. Pick a point A_1 in the relative interior of E_a' and a point A_2 in the relative interior of E_a'' (see figure 15.5). We define S_b, B_1 and B_2 similarly.

Let Λ be the union of the subchain Γ_{12} of Γ that joins A_1 to A_2 and of the simple polygonal chain contained in the interior[2] of S_a. Then Λ is a simple closed polygonal chain. The bounded polygonal region Δ enclosed by Λ contains, in a neighborhood of B_1, a portion of the segment B_1B_2. Indeed, if Λ is oriented by the orientation induced by Γ, then in a neighborhood of A_1 the side S_a is on the right of Λ and the cell lies to the left, and a similar statement holds for S_a in a neighborhood of A_2 and for S_b in a neighborhood of B_1. Moreover, Λ cannot cross the portion of Γ that joins A_2 to B_2, so that Δ cannot contain B_2. The segment B_1B_2 must therefore cross Λ. It cannot cross Γ_{12}, however, hence it must cross $\Lambda \setminus \Gamma_{12}$, and therefore also A_1A_2.

Assume now for a contradiction that *ababa* is a subsequence of Σ_Γ^*. In addition to the notation above, let us pick a point A_3 in the relative interior of E_a'' that is after A_2 on E_a'', and another point A_4 in the relative interior of the third edge E_a''' labeled a, and so supported by S_a. From the preceding argument, we know that A_1A_2 and B_1B_2 intersect, and similarly for B_1B_2 and A_3A_4 (simply consider

[2]We assume that the segments are in general position, and that they are infinitely thin rectangles.

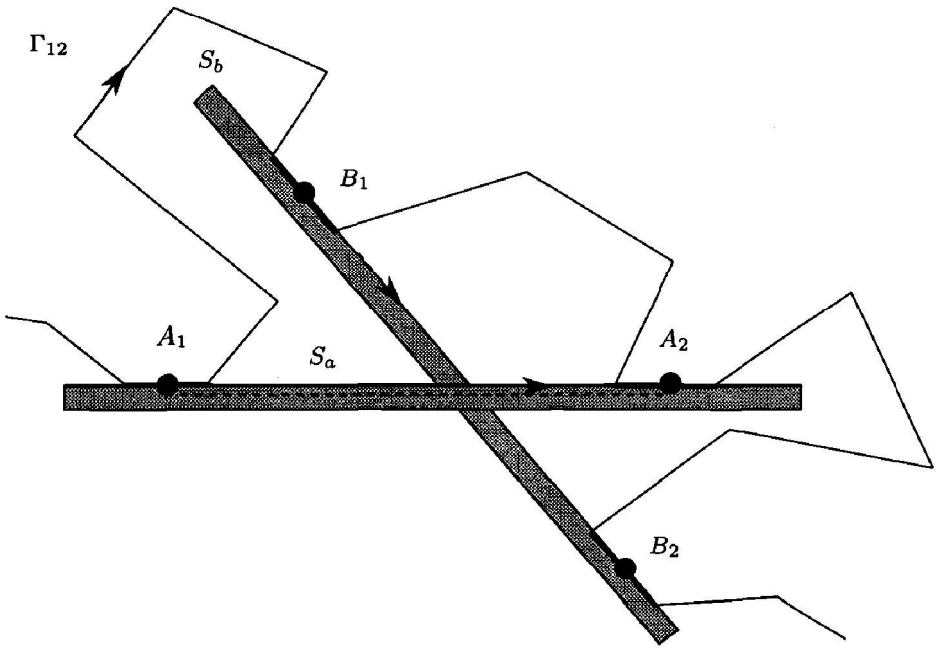

Figure 15.5. For the proof of lemma 15.4.2.

the subsequence *baba* of Σ_Γ). The two intersection points must be distinct since, owing to the relabeling and to lemma 15.4.1, the points A_i are all distinct and necessarily appear in the order A_1, A_2, A_3, A_4 on S_a. But this latter condition implies that A_1A_2 and A_3A_4 cannot intersect, so that S_b must cut S_a twice. This is impossible as two segments may only cross once. □

An immediate consequence of this lemma is:

Theorem 15.4.3 *The complexity of a cell in the arrangement of n line segments in the plane is* $O(n\alpha(n))$.

As we mentioned in section 15.3, it is possible to place segments in the plane so that the cell containing, say, the origin has complexity $\Omega(n\alpha(n))$, so the bound in the theorem above is tight.

15.4.2 Computing a cell

The algorithm

Again let $\mathcal{S}$ be a set of n line segments in the plane, assumed to be in general position, and pick a point A that does not belong to any of the segments in $\mathcal{S}$. Our goal is to compute the cell $C(\mathcal{S})$ in the arrangement of $\mathcal{S}$ that contains A.

The algorithm we present here is a variant of the randomized on-line algorithm that computes the vertical decomposition of $\mathcal{S}$. The reader unfamiliar with that

algorithm is invited to refer to subsection 5.3.2 for more details. We will only recall here the main definitions. The vertical decomposition is obtained by casting a ray upwards and downwards from any endpoint of the segments. The ray stops as soon as it encounters a segment in $\mathcal{S}$ (see figure 5.4a,d). The vertical segments (sometimes half-lines) traced by the rays are called walls, and together with the segments in $\mathcal{S}$ they decompose the plane into trapezoids that may degenerate into triangles or unbounded trapezoids. The algorithm also computes the vertical adjacencies of the trapezoids.[3]

To apply the formalism of chapter 4, we defined the problem in subsection 5.3.2 in terms of objects, regions, and conflicts between objects and regions. For this problem, an object is a segment. A region is a trapezoid in the decomposition of a subset of the segments. Each region is determined by at most four segments. There is a conflict between an object and a region if and only if the segment intersects the trapezoid. Computing the vertical decomposition is thus the same as computing the set of regions defined and without conflicts over $\mathcal{S}$.

The algorithm we present to compute a cell $C(\mathcal{S})$ in fact computes a vertical decomposition of that cell (see figure 15.6). To generalize the algorithm of subsection 5.3.2 to compute only a single cell is not straightforward, however: the regions that interest us are not all the trapezoids defined and without conflict over the set $\mathcal{S}$ of segments, but only those contained in the cell $C(\mathcal{S})$. Unfortunately, whether a trapezoid is contained in the cell $C(\mathcal{S})$ cannot be decided locally by examining only that trapezoid and the segments that define it. This forbids *verbatim* use of the formalism and results of chapters 4 and 5.

To avoid this difficulty, we proceed as follows. Let $\mathcal{R}$ be the subset of segments already inserted into the data structure, and let $C(\mathcal{R})$ be the cell in the arrangement of $\mathcal{R}$ that contains A. We allow the algorithm to compute, in addition to the trapezoids in the decomposition of $C(\mathcal{R})$, other trapezoids in the arrangement of $\mathcal{R}$ that are not trapezoids of $C(\mathcal{R})$. In order not to degrade the performances of the algorithm, at certain incremental steps we perform a *clean-up* step, during which we remove the trapezoids that do not belong to the cell $C(\mathcal{R})$. Only the trapezoids that belong to $C(\mathcal{R})$ will be subdivided during subsequent incremental insertions. To distinguish between these trapezoids, we traverse the connected component in the vertical adjacency graph $\mathcal{G}$ of the current vertical decomposition that contains the trapezoid containing A. This latter trapezoid is maintained throughout the incremental steps. The other leaves of the graph that are not traversed are *deactivated*: they correspond to trapezoids in the current decomposition that are not contained in the cell $C(\mathcal{R})$. These trapezoids will not be subdivided, and the corresponding leaves in the graph will not have children in subsequent insertions. Figure 15.7 shows an intermediate situation in the algorithm.

[3]Recall that two trapezoids are vertically adjacent if they share a common vertical wall.

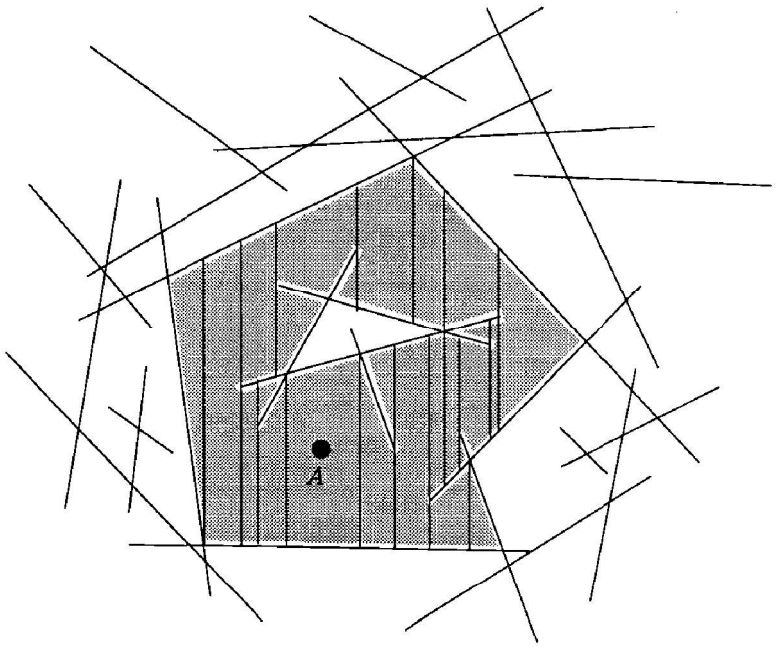

Figure 15.6. Vertical decomposition of the cell that contains A.

Between two clean-up steps, the algorithm is similar to the one described in subsection 5.3.2, apart from a few details which will be noted below. For each insertion of a new segment S, we locate S using the influence graph, then update the decomposition by subdividing the active trapezoids intersected by S. In the influence graph, this corresponds to creating new children for the active nodes that conflict with S.

Between two clean-up steps, and inside each trapezoid which has not been deactivated, we build the decomposition of the arrangement of the segments which conflict with this trapezoid and are inserted between the two clean-up steps. Let $\mathcal{T}_p$ be the set of nodes in the influence graph which were not deactivated during the previous clean-up step p. To each node in $\mathcal{T}_p$ we assign a secondary influence graph. This *secondary graph* is rooted at T and its nodes are the descendants of T created between step p and the next clean-up step. The secondary graph computed just as in subsection 5.3.2 under the incremental insertions of the segments inserted between step p and the next clean-up step. Its construction differs from that of a usual influence graph in a minor detail: the removal of superfluous walls. When inserting a segment S, if it intersects a wall, then only one of the two parts of that wall intersected by S is a wall in the new arrangement, and the other part must be removed and the two adjacent trapezoids must be merged. This procedure is detailed in subsection 5.3.2, and we apply it here to adjacent trapezoids that belong to the same secondary influence graph and also to trapezoids in different secondary influence graphs. A merge of the latter kind is called an *ex-*

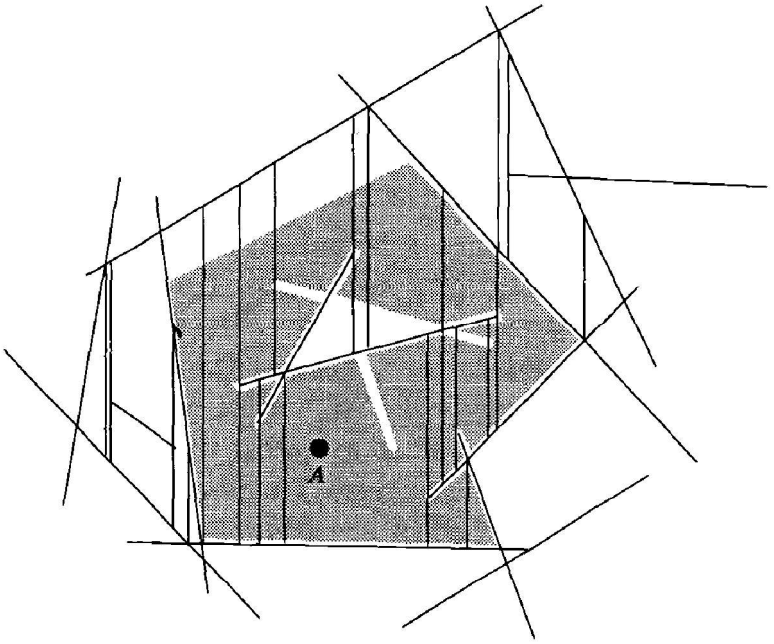

Figure 15.7. Intermediate situation in the computation of the cell that contains A. The shaded zone represents the final cell. The trapezoids which are neither entirely nor partially shaded are deactivated.

ternal merge. The vertical adjacencies are updated accordingly. External merges therefore introduce certain links between nodes in distinct secondary influence graphs.

The clean-up steps ensure that not too many trapezoids are created. Nevertheless, they must not be so frequent that the algorithm becomes inefficient. We perform clean-up steps after the insertion of the 2^i-th segment, for $i = 1, \ldots, \lfloor \log n \rfloor - 1$. Note that the last clean-up step was performed at step p_f where p_f is the greatest power of 2 such that $2p_f \leq n$.

Analysis of the algorithm

Suppose for now that n is a power of 2. We will analyze the complexity of the algorithm between two clean-up steps p and $2p$.

Denote by $\mathcal{S}_i$ the set of segments inserted during steps $1, \ldots, 2^i$. Each trapezoid T of $C(\mathcal{S}_p)$ is subdivided into trapezoids by the segments with which it conflicts. Let $\mathcal{S}_{2p}^T$ stand for the set of segments of $\mathcal{S}_{2p}$ that conflict with T, and let Σ_{2p}^T be the corresponding chronological sequence. The portion of the decomposition of the segments in $\mathcal{S}_{2p}^T$ that lies inside T has complexity $O(|\mathcal{S}_{2p}^T|^2)$.

To make things simpler, we assume that the algorithm does not perform the external merges. The number of nodes is only greater, so the location phase is

always more complex. The cost of the external merges is proportional to the number of nodes killed (and hence visited) during the steps, so the external merges are accounted for by the location phase. The bounds we obtain on the complexity of the algorithm that does not perform the external merges will thus still be valid for the algorithm that performs the external merges.

Subsection 5.3.2 provides us with a bound on the number of nodes and the storage needed by the secondary influence graphs. For a trapezoid T the bound is $O(|\mathcal{S}_{2p}^T|^2)$. To bound the storage required by all the secondary influence graphs computed between steps p and $2p$ (and ignoring the external merges), we sum this quantity over all the trapezoids of $C(\mathcal{S}_p)$. Here we need a moment theorem analogous to theorem 4.2.6, but usable in a context where regions are not defined locally. Such a theorem is stated in exercise 4.4 and bounds this sum by a function of the expected complexity $g_0(r, \mathcal{Z})$ of a cell in the arrangement of a random sample of r segments in a set $\mathcal{Z}$. (Note that this complexity is linearly equivalent to the complexity of its vertical decomposition.) Therefore, the number of nodes in all the secondary influence graphs computed between steps p and $2p$ is

$$\sum_T O(|\mathcal{S}_{2p}^T|^2) = O\left(\left(\frac{2p}{p}\right)^2 g_0(p, \mathcal{S}_{2p})\right) = O(p\alpha(p)).$$

The storage needed by the whole influence graph is thus

$$\sum_{i=1}^{\lfloor \log n \rfloor - 1} O(2^i \alpha(2^i)) = O(n\alpha(n)).$$

The complexity of the algorithm can be accounted for by three terms that correspond to the location phase, the update phase, and the clean-up steps. The location phase is analyzed in much the same way as the storage. We first evaluate the average number of nodes visited during step p, in the secondary influence graph rooted at a node that corresponds to a trapezoid T in $C(\mathcal{S}_p)$. The node is also denoted by T for simplicity. As before, we denote by $\mathcal{S}_{2p}^T$ (resp. Σ_{2p}^T) the subset (or the chronological sequence) of the segments that conflict with T and that are inserted before step $2p$. Denote by $\mathcal{S}_T$ (resp. Σ_T) the subset (or the chronological sequence) of all the segments that conflict with T. Note that $\mathcal{S}_{2p}^T$ is a random subset of $\mathcal{S}_T$. Under the assumption above, it all happens as if we were locating the segments in the sequence $\mathcal{S}_T$ in the secondary graph rooted at T. This graph is the influence graph corresponding to the decomposition of $\mathcal{S}_{2p}^T$ inside the interior of T. A slight adaptation of the proof of theorem 5.3.4 yields an upper bound on the expected number of nodes visited in the secondary influence graph. Let $f_0(r, \mathcal{Z})$ be the expected number of trapezoids in the decomposition of a random sample of r segments in a set $\mathcal{Z}$. Then $f_0(r, \mathcal{Z}) = O(r^2)$. If we assume

that $\mathcal{S}_{2p}^T$ is given, then we may use theorem 5.3.4 to bound the cost of inserting the last object by

$$O\left(\sum_{r=1}^{|\mathcal{S}_{2p}^T|} \frac{f_0(\lfloor r/2 \rfloor, \mathcal{S}_{2p}^T)}{r^2}\right).$$

This expression bounds the cost of inserting all the segments in $\mathcal{S}_{2p}^T$ as well as the cost of locating in the secondary graph rooted at T all the segments in $\mathcal{S}_T \setminus \mathcal{S}_{2p}^T$ that are inserted after step $2p$. The number of nodes visited in the secondary influence graph during the successive insertions, averaged over all random samples $\mathcal{S}_{2p}^T$ in $\mathcal{S}_T$, is thus

$$O\left(E\left(|\mathcal{S}_T| \sum_{r=1}^{|\mathcal{S}_{2p}^T|} \frac{f_0(\lfloor r/2 \rfloor, \mathcal{S}_{2p}^T)}{r^2}\right)\right) = O\left(|\mathcal{S}_T|\, E(|\mathcal{S}_{2p}^T|)\right).$$

Hence, the expected number of segments inserted before step $2p$ that conflict with T is

$$E(|\mathcal{S}_{2p}^T|) = \frac{p}{n-p}|\mathcal{S}_T| = O\left(\frac{p}{n}|\mathcal{S}_T|\right),$$

since $p \leq \frac{n}{2}$. The average number of nodes visited in the secondary influence graph rooted at T is finally $O\left(\frac{p}{n}|\mathcal{S}_T|^2\right)$.

Summing over all the nodes of $C(\mathcal{S}_p)$, we then obtain a bound on the expected number m of nodes visited in all the secondary influence graphs rooted at these nodes:

$$m = O\left(\frac{p}{n}\sum_T |\mathcal{S}_T|^2\right).$$

Once again, we can use the adapted moments theorem in exercise 4.4, to obtain

$$m = O\left(\frac{p}{n}\left(\frac{n}{p}\right)^2 p\alpha(p)\right) = O(n\alpha(p)). \tag{15.1}$$

Summing over all the clean-up steps, we get

$$\sum_{i=1}^{\lfloor \log n \rfloor - 1} O(n\alpha(2^i)) = O(n\alpha(n)\log n).$$

The update phases and clean-up steps are easily analyzed. Indeed, the update phases require time proportional to the number of nodes created, which is $O(n\alpha(n))$. Identifying the trapezoids of $C(\mathcal{S}_p)$ during the clean-up step p requires time proportional to the number of trapezoids in $C(\mathcal{S}_p)$, which is $O(p\alpha(p))$.

To deactivate the trapezoids during the different clean-up steps takes time proportional to the number of created nodes, which is again $O(n\alpha(n))$. The total cost of the clean-up steps is thus

$$\sum_{i=1}^{\lfloor \log n \rfloor - 1} O(2^i \alpha(2^i)) = O(n\alpha(n)).$$

This finishes the proof of the theorem stated below when n is a power of 2. To analyze the general case, we must also analyze the cost of inserting the segments at steps $2p_f + 1, \ldots, n$. But this is word for word the same as the analysis above and produces the same results (and notably equation 15.1) if we note that $p_f > \frac{n}{4}$.

Theorem 15.4.4 *A single cell in the arrangement of n line segments in the plane can be computed in expected time $O(n\alpha(n)\log n)$ and storage $O(n\alpha(n))$.*

15.5 Exercises

Exercise 15.1 (Optimal computation of lower envelopes) Show that the lower envelope of n line segments in the plane can be computed in optimal time $O(n \log n)$.

Hint: The lower bound $\Omega(n \log n)$ is proved by reduction to sorting. As for the upper bound, first project the endpoints of the segments on the x-axis. They define $2n - 1$ consecutive non-overlapping intervals. Build a balanced binary tree whose leaves are the intervals in the appropriate order. To a node corresponds an interval which is the union of the intervals at the leaves in the subtree. A segment S is assigned to the node whose interval is the smallest that still contains the projected endpoints of the segments. (This node is the first common ancestor of all the leaves covered by S.) Show that the lower envelope of the segments assigned to a single node has complexity $O(m)$ and not $O(m\alpha(m))$, using that there exists a vertical line that intersects all these segments and using also the result on half-lines mentioned in section 15.3. Observing that the projections of two segments assigned to different nodes at the same level in the tree do not overlap, show that the lower envelopes of the segments assigned to the nodes on a given level of the tree also have linear complexity, and can be computed in time $O(n \log n)$. These $O(\log n)$ lower envelopes can be merged in time $O(n\alpha(n) \log\log n)$, which is $O(n \log n)$.

Exercise 15.2 (Airport scheduling) Consider a set $\mathcal{M}$ of n points in $\mathbb{E}^d$ that move along algebraic curves of bounded degree at given constant speeds. At each moment t, we want to know the point $M(t)$ in $\mathcal{M}$ that is the closest to the origin. Show that the sequence Σ of points $M(t)$ for $t \in [0, +\infty[$ is almost linear, and that it may be computed in time $O((n + |\Sigma|) \log n)$.

Exercise 15.3 (Computing a view) Consider a scene formed by n line segments in the plane (not necessarily disjoint). Show that a view from a given point (defined as the portions of segments visible from that point) may be computed in optimal time $O(n \log n)$. Similar question for more general objects.

Hint: A projective transformation sends the origin to $(-\infty, 0)$, and the corresponding problem is exactly that of computing the lower envelope of n segments, see exercise 15.1.

Exercise 15.4 (Convex hull of objects) Show that computing the convex hull of n objects in the plane reduces to computing the lower envelope of n functions. If the objects are convex and disjoint, the graphs of these functions have at most two intersection points. Give bounds on the combinatorial and computational complexities of convex hulls of curved objects, in particular circles, ellipses, etc.

Hint: For each object, consider the set of its tangent lines, and use polarity to work in the dual plane (where points correspond to lines in the original plane).

Exercise 15.5 (Stabbing lines) Given n objects in the plane, compute the set of lines that simultaneously stab all of them.

Hint: Use the same polarity as in the preceding exercise.

Exercise 15.6 (Motion planning of a polygon under translation) Consider a polygon $\mathcal{M}$ with m sides that moves under translation within a polygonal region $\mathcal{E}$ with n sides.

1. Show that the set of translations that bring a vertex of $\mathcal{M}$ (resp. of $\mathcal{E}$) in contact with an edge of $\mathcal{E}$ (resp. of $\mathcal{M}$) is a set $\mathcal{C}$ of mn line segments (identifying the vector OM of the translation with the point M). Conclude that the set of feasible translations of $\mathcal{M}$ in $\mathcal{E}$ consists of one or several polygonal regions with total complexity $O(m^2n^2)$. If $\mathcal{M}$ is convex, the complexity is only $O(mn)$ (see exercise 19.8).

2. Show that the set of positions of $\mathcal{M}$ in $\mathcal{E}$ that are accessible from a given position I is the cell in the arrangement of $\mathcal{C}$ that contains I. Conclude that it is possible to determine whether two positions I and J are accessible one from the other, and if so, compute a feasible path for $\mathcal{M}$ from I to J in time $O(mn\alpha(mn)\log(mn))$.

3. Show that the complexity of the arrangement of $\mathcal{C}$ may be as bad as $\Omega(m^2n^2)$ (see figure 15.8).

4. Show that, in some cases, any path from I to J may have complexity $\Omega(mn)$. For instance, consider a carpenter's folding rule with m segments, but in a semi-folded rigid configuration, that tries to pass through n consecutive doors.

Exercise 15.7 (Non-trivial boundary) Consider two connected polygonal regions $\mathcal{B}$ and $\mathcal{R}$, which may have holes. A connected component of the intersection $\mathcal{B} \cap \mathcal{R}$ is called *non-trivial* if its boundary includes at least one vertex of $\mathcal{B}$ or $\mathcal{R}$. The *non-trivial boundary* of the intersection $\mathcal{B} \cap \mathcal{R}$ is the union of the polygons that bound all the non-trivial connected components of this intersection. Show that the complexity of the non-trivial boundary of the intersection $\mathcal{B} \cap \mathcal{R}$ is $O(|\mathcal{B}| + |\mathcal{R}|)$, where $|\mathcal{B}|$ and $|\mathcal{R}|$ respectively stand for the number of sides of $\mathcal{B}$ and $\mathcal{R}$. Hence any cell of the intersection of two polygonal regions $\mathcal{B}$ and $\mathcal{R}$ has complexity $O(|\mathcal{B}| + |\mathcal{R}|)$.

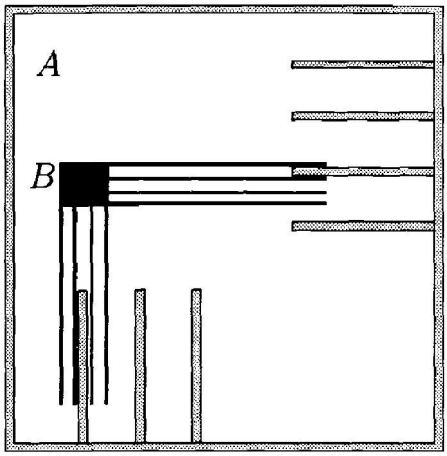

Figure 15.8. There are $\Omega(m^2n^2)$ feasible positions of $\mathcal{M}$ that belong to the same number of distinct cells of $\mathcal{C}$.

Hint: Put $\mathcal{V} = \mathcal{B} \cap \mathcal{R}$ for the intersection of $\mathcal{B}$ and $\mathcal{R}$. Since $\mathcal{R}$ and $\mathcal{B}$ play entirely symmetric roles, it suffices to look at the contribution of the boundary of $\mathcal{B}$ to the non-trivial boundary of $\mathcal{V}$. For this, follow the edges on the boundary of $\mathcal{B}$, and count the number of edges of $\mathcal{V}$ contained in each edge of $\mathcal{B}$.

For each edge E on the boundary of $\mathcal{B}$, count the edges of $\mathcal{V}$ contained in E. We distinguish the first one along E from the others. Among the others, count separately those that belong to the same connected component of $\mathcal{V}$, those that do not belong to the same connected component of the boundary of $\mathcal{V}$, and the remaining edges.

Exercise 15.8 (Computing the non-trivial boundary) Show that the non-trivial boundary of two polygonal regions $\mathcal{B}$ and $\mathcal{R}$ (see exercise 15.7) can be computed in time $O(m \log m)$, if m is the total number of edges of $\mathcal{B}$ and $\mathcal{R}$.

Hint: We sweep the plane with a line going in two directions, first going from left to right and then from right to left. During the sweep, we maintain three structures which respectively represent the segments of $\mathcal{B}$, of $\mathcal{R}$, and of the resulting non-trivial boundary that intersect the sweep line. During the left-to-right sweep, we only create a new interval for the result when the current event is a vertex of $\mathcal{B}$ contained in $\mathcal{R}$, or a vertex of $\mathcal{R}$ contained in $\mathcal{B}$. We call such a vertex a *remarkable* vertex. Then we are assured that this interval is contained in a non-trivial cell. We do not discover the entire non-trivial cell, however, rather we only know the portion of this cell that can join a remarkable vertex by a decreasing x-monotone path. This is why we need to sweep the plane in the other direction, from right to left.

Exercise 15.9 (Computing a cell) Devise a deterministic algorithm that computes a single cell in the arrangement of n line segments in the plane, in time $O(n \log^2 n)$. The cell is characterized by a point A that belongs to it.

Hint: Use the divide-and-conquer method. Split the set of n segments into two subsets of roughly the same size to obtain two cells C_1^A and C_2^A in the sub-arrangements that

contain A. Merging these two cells can be done using a variant of the sweep method of exercise 15.8. This variant in fact computes the non-trivial boundary of the intersection $C_1^A \cap C_2^A$ as well as the boundary of the cell in this intersection that contains A, even if it does not belong to the non-trivial boundary. It remains to extract the description of the cell C_A in the current divide-and-conquer step that contains A.

Exercise 15.10 (Half-lines) Show that the complexity of a cell in the arrangement of n half-lines is $O(n)$. Devise a deterministic algorithm that computes it in optimal time $O(n \log n)$.

Hint: Applying a rotation if necessary, we may assume that no half-line is vertical. Suppose that the cell is characterized by a point that belongs to it. Distinguish between the subset $\mathcal{E}^+$ of the half-lines that intersect $y = +\infty$ and the subset $\mathcal{E}^-$ of the half-lines that intersect $y = -\infty$. For each of these subsets, we explain how to compute the unbounded cell that contains the origins of some half-lines. (The other cells can be computed in a similar way.) For $\mathcal{E}^-$, we compute a left tree and a right tree by sweeping the plane from top to bottom with a line parallel to the x-axis. At each intersection I between two half-lines, we keep only the portion of the half-lines which lies to the left (for the left tree) or to the right (for the right tree). The boundary of the unbounded cell of $\mathcal{E}^-$ is obtained by computing the boundary of the unbounded cell of the union of both trees. Finally, exercise 15.8 can be used to compute the intersection of the cells of $\mathcal{E}^+$ and $\mathcal{E}^-$ that contain A.

Exercise 15.11 (Curved arcs) Bound the complexity of a cell of an arrangement of curved arcs in the plane and devise an algorithm that computes the cell that contains some given point.

Exercise 15.12 (Manipulator) A planar manipulator is formed by two rigid bodies articulated in a point A. One body is fixed to the origin O. The manipulator has two degrees of freedom: a rotation around O and a rotation around A. The configuration of the manipulator is parameterized by the corresponding two angles. Given some obstacles, devise an algorithm that computes the set of configurations for which the manipulator does not collide with an obstacle. Devise also an algorithm that determines whether two positions are reachable one from the other and, if so, outputs a path that realizes this change of configuration.

Hint: Express the constraints that limit the motion of the manipulator in the configuration space (which has dimension 2) and use exercise 15.11.

15.6 Bibliographical notes

The connection between lower envelopes of functions and Davenport–Schinzel sequences was established in a paper by Davenport and Schinzel [74]. Atallah [13], then Sharir and collaborators [1, 122, 206] proved bounds on the length of Davenport–Schinzel sequences. Wiernik and Sharir [219] showed how to realize a lower envelope of n segments in the

plane that has complexity $\Omega(n\alpha(n))$. The solution to exercise 15.1 is due to Hershberger [124].

The analyses of the complexity and of the computation of the unbounded cell in the arrangement of line segments are given by Pollack, Sharir, and Sifrony [189]. Their result is extended by Guibas, Sharir, and Sifrony [118] to the case of a cell in the arrangement of curved arcs. Other results on curved arcs are given in [64, 90]. The complexity and computation of m cells is studied by Edelsbrunner, Guibas, and Sharir in [91, 93]. Solutions to exercises 15.7, 15.8, and 15.9 can be found in their papers.

Alevizos, Boissonnat, and Preparata [7] study the arrangements of half-lines and give a solution to exercise 15.10. These arrangements find applications in pattern recognition [8, 33].

The randomized algorithm that computes a single cell described in this chapter is due to de Berg, Dobrindt, and Schwarzkopf [76]. This algorithm can be generalized to dimension 3, which is not the case for a previous algorithm due to Chazelle *et al.* [52].

A comprehensive survey of Davenport–Schinzel sequences and their geometric applications can be found in the book by Sharir and Agarwal [207].