Algorithms for embedded graphs

Éric Colin de Verdière

MPRI, 2013–2014

Foreword and introduction

Foreword

These are the course notes for half of the MPRI course "Algorithms for embedded graphs". Announcements for this course may be found on the webpage https://wikimpri.dptinfo.ens-cachan.fr/doku.php?id=cours:c-2-38-1. The other half of the course will be taught by Claire Mathieu, who will provide notes independently.

These notes are certainly not in final shape, and comments by e-mail are welcome. The course may depart from these notes both in content and presentation.

It is strongly recommended to work on the exercises. Each exercise is labeled with one to three stars, supposed to be an indication of its *importance* (in particular, depending on whether it is used later), not of its difficulty.

Introduction

This is an introduction to the computational aspects of graphs drawn without crossings in the plane or in more complicated surfaces. This topic has been a subject of active research, especially over the last decade, and is related to rather diverse fields and communities:

- in *graph algorithms*: As we shall see, because planar graphs bear important properties, many general graph problems become easier when restricted to planar graphs (shortest path, flow and cut, minimum spanning trees, vertex cover, graph isomorphism, etc.). The same holds for graphs on surfaces, to some extent;
- in *graph theory*, the theory of graph minors founded by Robertson and Seymour makes heavy use of graphs embeddable on a fixed surface, as well as graphs excluding a fixed minor. Edge-width and face-width are closely related to the notion of shortest non-contractible closed curve;
- in *topology*, the classification of surfaces, as discovered in the beginning of the 20th century, is inherently algorithmic. Surfaces play a prominent role in the deep theories of knots and three-manifolds; there are also many algorithmic questions in these areas;
- in *computational geometry*, surfaces arise naturally in various applications. Operations in geometric spaces such as decomposition, extraction of important features, and shortest path computation are basic computational geometry tasks that are relevant in particular for surfaces, usually embedded in $\mathbb{R}^3$, or even planar surfaces.

Many graphs encountered in practice are geometric, and either are planar or have a few crossings (think of a road network with a few overpasses and underpasses). Thus it makes sense to look for efficient algorithms dedicated to such graphs. In addition, in computer graphics, one needs to efficiently process surfaces represented by triangular meshes, e.g., to cut them to make them planar; we shall introduce algorithms for such purposes.

The first chapter introduces planar graphs from the topological and combinatorial point of view. The second chapter considers the problem of testing whether a graph is planar, and, if so, of drawing it without crossings in the plane. Then we move on with some general graph problems, for which we give efficient algorithms when the input graph is planar.

In the second part of the course, we consider graphs on surfaces (planar graphs being an important special case). In Chapter 4, we introduce surfaces from the topological point of view; in Chapter 5, we present algorithms using the cut locus to build short curves and decompositions of

Date of this version: September 11, 2013. Latest version available at http://www.di.ens.fr/~colin/cours/algo-embedded-graphs.pdf.

surfaces. In Chapter 6, we use such decompositions to give an efficient algorithm for computing minimum cuts in surface-embedded graphs.

Only a part of the material covered in this course appeared in textbooks. For further reading or different expositions, mostly on the topological aspects, recommended books are Mohar and Thomassen [41], Armstrong [2], and Stillwell [47]. For the algorithmic aspects and a wider perspective, see the very recent course notes by Erickson [18].

Acknowledgments

I would like to thank several people who suggested some corrections in previous versions: Jeff Erickson, Francis Lazarus, Arthur Milchior, and Vincent Pilaud. Thanks to Éric Fusy for helpful informations on graph drawing algorithms.

Chapter 1

Basic properties of planar graphs

1.1 Topology

1.1.1 Preliminaries on topology

We assume some familiarity with basic topology, but we recall some definitions nonetheless.

A *topological space* is a set X with a collection of subsets of X, called *open sets*, satisfying the three following axioms:

- the empty set and X are open;
- any union of open sets is open;
- any finite intersection of open sets is open.

There is, in particular, no notion of metric (or distance, angle, area) in a topological space. The open sets give merely an information of *neighborhood*: a neighborhood of $x \in X$ is a set containing an open set containing x. This is already a lot of information, allowing to define continuity, homeomorphisms, connectivity, boundary, isolated points, dimension. . . . Specifically, a map $f : X \to Y$ is *continuous* if the inverse image of any open set by f is an open set. If X and Y are two topological spaces, a map $f : X \to Y$ is a *homeomorphism* if it is continuous, bijective, and if its inverse f^{-1} is also continuous. A point of detail, ruling out pathological spaces: the topological spaces considered in these notes are assumed to be

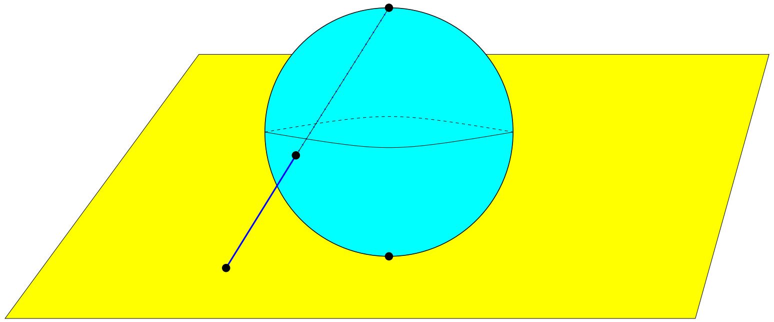

Figure 1.1. The stereographic projection.

Hausdorff, which means that two distinct points have disjoint neighborhoods.

Example 1.1. Most of the topological spaces here are endowed with a natural metric, which should be "forgotten", but define the topology:

- $\mathbb{R}^n$ ($n \geq 1$);
- the n-dimensional sphere $\mathbb{S}^n$, i.e., the set of unit vectors of $\mathbb{R}^{n+1}$;
- the n-dimensional ball B^n, i.e., the set of vectors in $\mathbb{R}^n$ of norm at most 1; in particular $B^1 = [-1, 1]$ and $[0, 1]$ are homeomorphic;
- the set of lines in $\mathbb{R}^2$, or more generally the *Grasmannian*, the set of k-dimensional vector spaces in $\mathbb{R}^n$.

Exercise 1.2 (stereographic projection)**.** ☆☆ Prove that the plane is homeomorphic to $\mathbb{S}^2$ with an arbitrary point removed. (Indication: see Figure 1.1.)

A *closed set* in X is the complement of an open set. The *closure* of a subset of X is the (unique) smallest closed set containing it. The *interior* of a subset of X is the (unique) largest open set contained in it. The *boundary* of a subset of X equals its closure minus its interior. A topological space X is *compact* if any set of open sets whose union is X admits a finite subset whose union is still X.

A *path* in X is a continuous map $p : [0, 1] \to X$; its *endpoints* are $p(0)$ and $p(1)$. Its *relative interior* is the image by p of the open interval $(0, 1)$. A path is *simple* if it is one-to-one. A space X is *connected*[1] if it is non-empty and, for any points a and b in X, there exists a path in X whose endpoints are a and b. The *connected components* of a topological space X are the classes of the equivalence relation on X defined by: a is equivalent to b if there exists a path between a and b. A topological space X is *disconnected* (or *separated*) by $Y \subseteq X$ if and only if $X \setminus Y$ is not connected; points in different connected components of $X \setminus Y$ are *separated* by Y.

1.1.2 Graphs and embeddings

We will use standard terminology for graphs. Unless noted otherwise, all graphs are undirected and finite but may have loops and multiple edges. A *circuit* in a graph G is a closed walk without repeated vertices.[2]

A graph yields naturally a topological space:

- for each edge e, let X_e be a topological space homeomorphic to $[0, 1]$; let X be the disjoint union of the X_e;
- for e, e', identify (quotient topology), in X, endpoints of X_e and $X_{e'}$ if these endpoints correspond to the same vertex in G.

An *embedding* of G in the plane $\mathbb{R}^2$ is a continuous one-to-one map from G (viewed as a topological space) to $\mathbb{R}^2$. Said differently, it is a "crossing-free drawing" of G on $\mathbb{R}^2$, being the data of two maps:

- Γ_V, which associates to each vertex of G a point of $\mathbb{R}^2$;
- Γ_E, which associates to each edge e of G a path in $\mathbb{R}^2$ between the images by Γ_V of the endpoints of e,

in such a way that:

- the map Γ_V is one-to-one (two distinct vertices are sent to distinct points of $\mathbb{R}^2$);

[1] In this course, the only type of connectivity considered is path connectivity.

[2] This is often called a *cycle*; however, in the context of these notes, this word is also used to mean a homology cycle or a closed curve, so it seems preferable to avoid overloading it again.

- for each edge e, the relative interior of $\Gamma_E(e)$ is one-to-one (the image of an edge is a simple path, except possibly at its endpoints);
- for all distinct edges e and e', the relative interiors of $\Gamma_E(e)$ and $\Gamma_E(e')$ are disjoint (two edges cannot cross);
- for each edge e and for each vertex v, the relative interior of $\Gamma_E(e)$ does not meet $\Gamma_V(v)$ (no edge passes through a vertex).

We can actually replace $\mathbb{R}^2$ above with *any* topological space Y and talk about an embedding of a graph in Y.

When we speak of embedded graphs, we sometimes implicitly identify the graph, its embedding, and the image of its embedding.

1.1.3 Planar graphs and the Jordan curve theorem

In the remaining part of this chapter, we only consider embeddings of graphs into the sphere $\mathbb{S}^2$ or the plane $\mathbb{R}^2$.

A graph is *planar* if it admits an embedding into the plane. By Exercise 1.2, this is equivalent to the existence of an embedding into the sphere $\mathbb{S}^2$.

The *faces* of a graph embedding are the connected components of the complement of the image of the vertices and edges of the graph.

Here are the most-often used results in the area.

Theorem 1.3 (Jordan curve theorem, reformulated; see [49]). *Let G be a graph embedded on $\mathbb{S}^2$ (or $\mathbb{R}^2$). Then G disconnects $\mathbb{S}^2$ if and only if it contains a circuit.*

Theorem 1.4 (Jordan–Schönflies theorem; see [49]). *Let $f : \mathbb{S}^1 \to \mathbb{S}^2$ be a one-to-one continuous map. Then $\mathbb{S}^2 \setminus f(\mathbb{S}^1)$ is homeomorphic to two disjoint copies of the open disk.*

Exercise 1.5. ☆☆ Sketch a proof of the Jordan curve theorem in the case where G is embedded in the plane with polygonal edges.

These results are, perhaps surprisingly, difficult to prove: the difficulty comes from the generality of the hypotheses (only continuity is required). For example, if in the Jordan curve theorem one assumes that G is embedded in the plane with polygonal edges, the theorem is not hard to prove.

A graph is *cellularly embedded* if its faces are (homeomorphic to) open disks. Henceforth, we only consider cellular embeddings. It turns out that a graph embedded on the sphere is cellularly embedded if and only if it is connected.[3]

1.2 Combinatorics

So far, we have considered curves and graph embeddings in the plane that are rather general.

1.2.1 Combinatorial maps for planar graph embeddings

We now focus on the combinatorial properties of cellular graph embeddings in the sphere. Since we are not interested in the geometric properties, it suffices to specify how the faces are "glued together", or alternately the cyclic order of the edges around a vertex. Embeddings of graphs on the plane are treated similarly: just choose a distinguished face of the embedding into $\mathbb{S}^2$, representing the "infinite" face of the embedding in the plane.

An algorithmically sound way of representing combinatorially a cellular graph embedding in $\mathbb{S}^2$ is via *combinatorial maps*, which we now describe. The basic notion is the *flag*, which represents an incidence between a vertex, an edge, and a face of the embedding. Three involutions allow to move to a nearby flag, and, by iterating, to visit the whole graph embedding; see Figure 1.2:

- *vi* moves to the flag with the same edge-face incidence, but with a different vertex incidence;

[3] Although this statement should be intuitively clear, it is not so obvious to prove. It may help to use the results of Chapter 4, especially the fact that every face of a graph embedding is a surface with boundary.

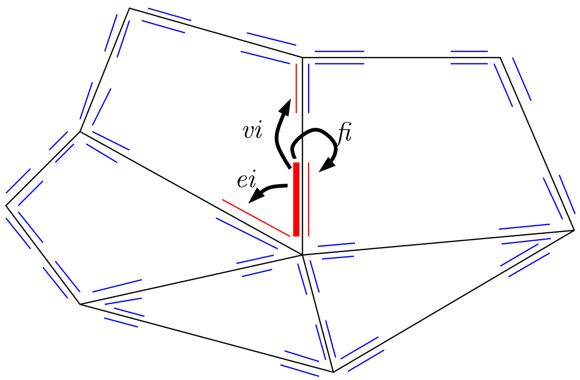

Figure 1.2. The flags are represented as line segments parallel to the edges; there are four flags per edge. The involutions *vi*, *ei*, and *fi* on the thick flag are also shown.

```
int vertex_degree(Flag fl) {
  int j=0;
  Flag fl2=fl;
  do {
    ++j;
    fl2=fl2->ei()->fi();
  } while (fl2!=fl);
  return j;
}
```

```
int face_degree(Flag fl) {
  int j=0;
  Flag fl2=fl;
  do {
    ++j;
    fl2=fl2->ei()->vi();
  } while (fl2!=fl);
  return j;
}
```

Figure 1.3. C++ code for degree computation.

- *ei* moves to the flag with the same vertex-face incidence, but with a different edge incidence;
- *fi* moves to the flag with the same vertex-edge incidence, but with a different face incidence.

Example 1.6. Figure 1.3, left, presents code to compute the degree of a vertex, i.e., the number of vertex-edge incidences of this vertex. The function takes as input a flag incident with that vertex. Note that a loop incident with the vertex makes a contribution of two to the degree. Dually, on the right, code to compute the degree of a face (the number of edge-face incidences of this face) is shown.

Note that a flag is not necessarily uniquely defined by its triple (vertex, edge, and face), as shows the example of a graph with a single vertex and a single (loop) edge.

The *complexity* of a graph $G = (V, E)$ is $|V| + |E|$. The *complexity* of a cellular graph embedding is the total number of flags involved, which is linear in the number of edges (every edge bears four flags), and also in the number of vertices, edges, and faces. Therefore the complexity of a graph cellularly embedded in the plane and of one of its embeddings are linearly related.

The data structure indicated above allows to "navigate" throughout the data structure, but does not store vertices, edges, and faces explicitly. In many cases, however, it is necessary to have one data structure ("object") per vertex, edge, or face. For example:

- if one has to be able to check in constant time whether an edge is a loop (incident twice to the same vertex), the data structure given above is not sufficient. On the other hand, if every flag has a pointer to the incident vertex, then testing whether an edge is a loop can be done by testing the equality of two pointers in constant time;
- in coloring problems, one need to store colors on the vertices of the graph. Such information can be stored in the data structure used for each vertex.

For such purposes, each flag can have a pointer to the underlying vertex, edge, and face (called respectively *vu*, *eu*, *fu*). Each such vertex, edge, or face contains no information on the incident elements, only information needed in the algorithms. If needed, one may similarly put some information in the vertex-edge, edge-face, vertex-face, and vertex-edge-face incidences. Maintaining such informations, however, comes with a cost, which is not always desirable. For example, assume we want to be able to remove one edge incident to two different faces in constant time. If we keep the information *fu*, this must take time proportional to the smaller degree of the two faces (since the two faces are merged, the *fu* pointer has to be updated at least on one side of the edge). If we only keep *vu*, say,

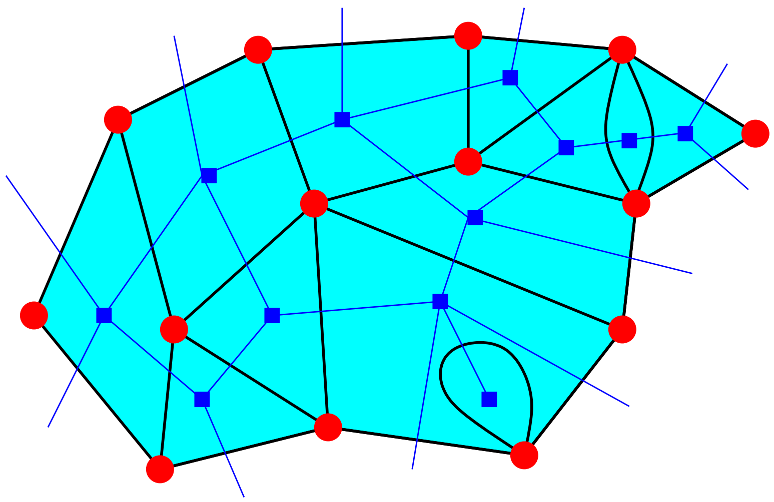

Figure 1.4. Duality.

then such an update is not needed, and this edge removal can be done in constant time.

1.2.2 Duality and Euler's formula

A *dual graph* of a cellular graph embedding $G = (V, E)$ on $\mathbb{S}^2$ is a graph embedding G^* defined as follows: put one vertex f^* of G^* in the interior of each face f of G; for each edge e of G, create an edge e^* in G^* crossing e and no other edge of G (if e separates faces f_1 and f_2, then e^* connects f_1^* and f_2^*). See Figure 1.4.

A dual graph embedding is also cellular. The combinatorial map of the dual graph is unique. Actually, with the map representation, dualizing is easy: simply replace *fi* with *vi* and vice-versa. This in particular proves that duality is an involution: $G^{**} = G$.

Exercise 1.7 (easy). ☆☆☆ Every tree (acyclic connected graph) with v vertices and e edges satisfies $v - e = 1$.

Lemma 1.8. *Let $G = (V, E)$ be a cellular graph embedding in $\mathbb{S}^2$, and let $G^* = (F^*, E^*)$ be its dual graph. Furthermore, let $E' \subseteq E$.*

Then (V, E') is acyclic if and only if $(F^*, (E \setminus E')^*)$ is connected. In particular, (V, E') is a spanning tree if and only if $(F^*, (E \setminus E')^*)$ is a spanning tree.

Proof. (V, E') is acyclic if and only if $\mathbb{S}^2 \setminus E'$ is connected, by the Jordan curve theorem 1.3. Furthermore, $\mathbb{S}^2 \setminus E'$ is connected if and only if $(F^*, (E \setminus E')^*)$ is connected: Two points x and x' in faces f and f' of G can be connected by a path avoiding E' and not entering any face other than f and f' if and only if f and f' are adjacent by some edge not in E', i.e. if and only if f^* and f'^* are adjacent in $(F^*, (E \setminus E')^*)$. □

Corollary 1.9 (Euler's formula for cellular graph embeddings in $\mathbb{S}^2$)**.** *For every cellular graph embedding in $\mathbb{S}^2$ with v vertices, e edges, and f faces, we have $v - e + f = 2$.*

Hence this formula also holds for every embedding of a connected graph in the plane.

Proof. Let T be the edge set of a spanning tree of G. The dual edges of its complement, $(E \setminus T)^*$, is also a spanning tree. The number of edges of G is $e = |T| + |(E \setminus T)^*|$, which, by Exercise 1.7, equals $(v - 1) + (f - 1)$. □

Exercise 1.10 (easy direction of Kuratowski's theorem)**.** ☆☆☆ Show that the complete graph with 5 vertices, K_5, is not planar. Indication: Use Euler's formula and double-counting on the number of vertex-edge and edge-face incidences. Also show that the bipartite graph $K_{3,3}$ (with 6 vertices $v_1, v_2, v_3, w_1, w_2, w_3$ and 9 edges, connecting every possible pair $\{v_i, w_j\}$) is not planar.

1.3 Notes

For more information on basic topology, see for example Armstrong [2] or Henle [29]; see also Stillwell [47]. For more informations on planar graphs, see (the next two chapters and) Mohar and Thomassen [41, Chapter 2].

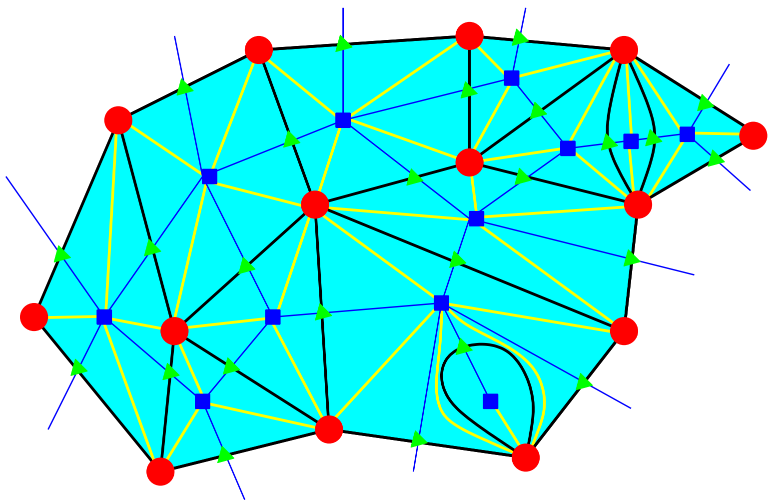

Figure 1.5. The barycentric subdivision of the part of the graph shown in Figure 1.4.

There are many essentially equivalent ways of representing planar graph embeddings [16, 33]; the computational geometry library CGAL implements one of them[4]. We will see later that (most of) these data structures generalize to graphs embedded on surfaces. There are further generalizations to higher dimensions [3, 37, 38]; this is important especially in geometric modelling.

Eppstein provides many proofs of Euler's formula[5].

Exercise 1.10 shows that K_5 and $K_{3,3}$ are not planar. There is a converse statement: Kuratowski's theorem asserts that a graph G is planar if and only if it does not contain K_5 or $K_{3,3}$ as a subdivision; in other words, if and only if one cannot obtain K_5 or $K_{3,3}$ from G by removing edges and isolated vertices and replacing every degree-two vertex and its two incident edges with a single edge [34, 39, 48].

Let G be a cellular embedding of a graph on $\mathbb{S}^2$. By overlaying G with its dual graph G^*, we obtain a *quadrangulation*: a cellular embedding of a graph G^+ where each face has degree four. See Figure 1.4. Every face of G^+ is incident with four vertices: one vertex v_G of G, one vertex v_{G^*} of G^*, and two vertices that are the intersection of an edge of G and an edge of G^*. If, within each face, we connect v_G with v_{G^*}, we obtain a triangulation, called the *barycentric subdivision* of G (Figure 1.5). Each triangle in the barycentric subdivision corresponds to a flag; its three neighbors are the flags reachable via the operations *vi*, *ei*, and *fi*.

[4] http://www.cgal.org/Manual/3.4/doc_html/cgal_manual/HalfedgeDS/Chapter_main.html.

[5] http://www.ics.uci.edu/~eppstein/junkyard/euler/.

Chapter 2

Planarity testing and graph drawing

Given a graph G in a "usual" form, e.g., where each vertex has a linked list of pointers to its incident edges, and each edge has two pointers to its incident vertices, how can we determine whether G is planar? Section 2.1 answers this question. Then we move on by considering algorithms to draw a planar graph in the plane.

2.1 Planarity testing

Given a graph G, how hard is it to determine whether G is planar?

Theorem 2.1. *Given a graph G, one can, in (optimal) linear time, determine whether G is planar, and if so, compute a combinatorial map of G in the plane.*

We shall here prove this theorem with a weaker, cubic complexity. With much care, refining these ideas indeed leads to a linear-time algorithm [30].

To check whether G is planar, we can obviously assume that G is connected. Furthermore, we can assume that every edge belongs to a circuit of G (another equivalent terminology is that G is 2-edge-connected); for if an edge e belongs to no circuit of G, we can remove it; G is planar if and only if $G - e$ is planar. In other words, define a *bridge* of a graph any edge that belongs to no circuit of the graph (equivalently, it is an edge whose removal disconnects the graph). It is a standard fact that one can determine all bridges of a graph in linear time:

Lemma 2.2. *Let G be a graph of complexity n. One can in $O(n)$ time determine all the bridge edges of G.*

Proof. Run a depth-first search on the graph G, starting from an arbitrary root vertex. Recall that this partitions the edges of G into *link edges*, which belong to the rooted search tree T, and *back edges*, which connect a vertex v with an ascendent of v in T. Clearly, no back edge is a bridge. A link edge e with endpoints u and v, where u is visited before v, is a non-bridge edge if and only if there exists a back edge from a descendent of v (maybe v itself) to an ascendent of u (maybe u itself). The algorithm will consider each back edge (uv) in turn and mark as non-bridge the edges on the path from u to v in T; the remaining edges are exactly the bridge edges.

To achieve this in linear time, take all back edges $(x_1, y_1), \dots, (x_k, y_k)$ (where y_k is an ascendent of x_k), ordered such that $y_1, \dots, y_k$ are discovered in this order during the depth-first search (such an ordering can easily be found in $O(n)$ time). Starting from x_1, and walking towards the root of T, mark every edge as being a non-bridge edge until reaching y_1. Start from x_2, and walk towards the root of the tree, marking every edge as non-bridge, until either reaching y_2 or reaching an edge e that is already marked as non-bridge. If the latter possibility occurs, y_1 must be an ancester of y_2 in T by the choice of the ordering, so all edges between e and y_2 must be already marked. Continue similarly with the other back edges. This process clearly takes linear time in total. □

We can then remove all bridge edges of G in linear time. Now, each edge of G lies on a circuit. We can furthermore assume that G is connected, by considering connected components separately.

Let C be a circuit of G. We partition the edges of $G - C$ into *pieces* as follows (see Figure 2.1): Two edges are in the same class if there is a path in G between them that does not contain any vertex of C. The vertices of a piece P that are in C are called its *attachments*. Since G is bridgeless, each piece has at least two attachments.

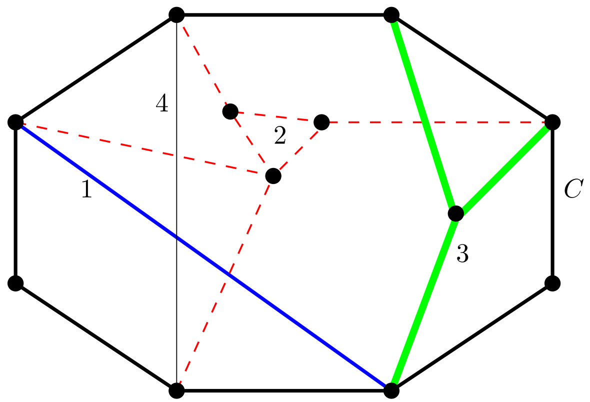

Figure 2.1. A graph G with a circuit C (on the outside of the figure) and the four pieces with respect to C numbered from 1 to 4. All pairs of pieces conflict except $(1,3)$ and $(3,4)$.

Lemma 2.3. *In linear time, we can either compute a circuit of G that has at least two pieces, or certify that G is planar.*

Proof. First compute any circuit C, using, e.g., depth-first search. Determine the pieces of C. If C has no piece, then $G = C$, thus G is planar. If C has two or more pieces, then C satisfies the conclusion, so we are done. So assume that C has a single piece P. Let $v_1, \ldots, v_k$ be the attachments of P on C, in cyclic order around C. Let p be a path in P between v_1 and v_2. Now, let C' be the circuit obtained by concatenating p with the subpath of C with endpoints v_1 and v_2 that also contains $v_3, \ldots, v_k$ (pick either of the two subpaths if $k = 2$). One piece of C' is the other subpath of C, and another piece of C' is $P \setminus p$, unless $P = p$, in which case $G = C \cup \{p\}$ is planar.

All of this takes linear time. □

If G is planar then, in a planar drawing of G, each piece of a circuit C must be entirely inside or outside C. We say that two pieces P and Q of G are *non-conflicting* with respect to C if, intuitively, in any planar drawing of G (if it exists), exactly one of P and Q must be drawn inside C. More formally, P and Q are non-conflicting if there are two (possibly identical) vertices u and v of C, splitting C into two subpaths C_1 and C_2 with endpoints u and v, such that all attachments of P are in C_1 and all attachments of Q are in C_2. Otherwise, P and Q are in *conflict*. The *conflict graph* of G with respect to C is a graph with vertex set the pieces of C; two pieces are connected if and only if they conflict.

Lemma 2.4. *Let C be a circuit of G. The graph G is planar if and only if the following conditions are satisfied:*

i. The conflict graph of G with respect to C is bipartite;

ii. for every piece P of G with respect to C, the graph obtained by adding P to C is planar.

Proof. Assume first that G is planar. In a planar embedding, each piece is drawn either entirely inside or outside C. Furthermore, two pieces drawn on the same side of C must be non-conflicting because, in the cyclic order around C, edges of P and of Q cannot be interlaced. (Otherwise, we would essentially have, after removal, contractions, and expansions of edges if needed, four vertices v_1, v_2, v_3, v_4 in this order on circuit C, with v_1 connected to v_3 and v_2 connected to v_4 by edges inside C; adding a new vertex outside C and connecting it to all four vertices, we would get K_5, which is nonplanar.) This implies that the conflict graph is bipartite. The second property is trivial.

For the opposite direction, by (i), we consider a bipartition of the conflict graph $\mathcal{P} \cup \mathcal{Q}$. Our embedding will have all pieces of $\mathcal{P}$ inside C and all pieces of $\mathcal{Q}$ outside C. We consider $\mathcal{P}$, since the same process applies to $\mathcal{Q}$. Since no two pieces in $\mathcal{P}$ are in conflict, we can order the pieces of $\mathcal{P}$ as $P_1, \ldots, P_k$ and write the circuit C as a concatenation of paths (possibly reduced to a single vertex) $p_1, \ldots, p_k$, such that P_k has all its attachment points on p_k. We draw disjoint simple paths $q_1, \ldots, q_k$ with the same endpoints as $p_1, \ldots, p_k$ respectively, inside C, such that the zones Z_i with boundary $p_i \cup q_i$ are disjoint (except perhaps at the endpoints of p_i). Now, using Condition (ii), we embed each piece P_i of $\mathcal{P}$ inside Z_i. □

At a high level, the algorithm first applies Lemma 2.3 to compute a circuit C with at least two pieces (unless G is planar, which concludes). Then it uses the characterization of Lemma 2.4: If the conflict graph of G with respect to C is non-bipartite, it returns that G is non-planar; otherwise, it recursively checks that $C \cup P$ is planar, for each piece P of G (such graphs are clearly connected and bridgeless). The correctness is clear.

To get an efficient algorithm, however, we need to be slightly more specific. The algorithm takes as input a connected bridgeless graph G, and a circuit C of G with at least two pieces.

1. Compute the pieces of G with respect to C.
2. For each piece P of G that is not a path:
   (a) let G' be the graph obtained by adding P to C;
   (b) let C' be the circuit of G' obtained from C by replacing the portion of C between two consecutive attachments with a path of P between them;
   (c) apply the algorithm recursively to graph G' and circuit C'. If G' is non-planar, return "non-planar".
3. Compute the conflict graph of the pieces.
4. Return "planar" or "non-planar" according to whether the conflict graph is bipartite or not.

The correctness follows from the proof of Lemma 2.3 and from the fact that each graph considered is connected and bridgeless.

Now, we study the complexity of Step 3:

Lemma 2.5. *Given a circuit C, we can determine the conflict graph of G with respect to C in quadratic time.*

Proof. Let P be a piece of C, with attachments $v_1, \ldots, v_k$ in cyclic order around C. Then another piece Q does not conflict with P if and only if all its attachments are in some interval $[v_i, v_{i+1}]$, in cyclic order around C (indices are taken modulo k). This suggests the following approach: Mark each vertex of C according to which interval(s) $[v_i, v_{i+1}]$ it belongs to; for each piece $Q \neq P$, determine if all its attachments belong to a single interval using this marking. This takes linear time plus a time linear in the number of attachment points of all the pieces, which is also linear. Iterating for every piece P, we obtain the conflict graph of G in quadratic time. □

Since testing whether a graph is bipartite can be done in linear time, this shows that each recursive invocation of the algorithm takes quadratic time. Furthermore:

Lemma 2.6. *The number of recursive invocations is linear in the complexity of the input graph.*

Proof. We associate a different edge of G to each invocation of the recursive algorithm. Namely, for a given invocation on graph G and circuit C, we select a *witness* edge e of C that does not belong to the circuit of the parent invocation. That edge e does not appear in the siblings' graphs, so it will not show up as a witness edge in any sibling invocation nor in any descendent of a sibling. There remains to prove that e does not appear as the witness edge of a descendent invocation. Consider a path of recursive invocations towards that descendent. If e belongs to the circuit of the current invocation, it cannot be chosen as the witness of a child's invocation, so we are safe; and if e ceases to belong to the circuit of the current invocation, then by choice of the new circuit C', e now belongs to a piece of C' that is a path, and therefore is absent from any descendent invocation. □

This proves Theorem 2.1.

2.2 Graph drawing

Now we consider the following problem: Given a planar graph G, given in the form of a combinatorial map (for example, obtained by the algorithm in the previous section), how can we build an actual embedding of G in the plane?

To be more specific, we need some definitions. A *simple* graph is a graph without loops or multiple edges. A planar graph is *triangulated* if every face of G, including the outer face, has degree three. A graph emebdding in the plane is *straight-line* if every edge is a straight-line segment (such an embedding is thus uniquely determined by the coordinates of its vertices). We shall prove:

Theorem 2.7. *Let G be a simple planar graph, given in the form of a combinatorial map. In $O(n)$ time, we can compute a straight-line embedding of G where the vertices are on a regular $O(n) \times O(n)$-grid.*

The restriction of having a simple graph is legitimate, because non-simple graphs do not have a straight-line embedding. Furthermore, we can remove all loops and multiple edges in a graph in linear time if desired:

Lemma 2.8. *Let G be a graph (not necessarily planar) of complexity n. In $O(n)$ time, we can determine all loop edges and multiple edges of G.*

Proof. Let v be a vertex of G. Mark each neighbor w of v with the list of edges with endpoints v and w, by visiting each edge incident with v in turn. Any list containing more than one edge corresponds to multiple edges; if the list of v is non-empty, it corresponds to one or several loops. This operation takes a time linear in the degree of v. We can iterate the process over all vertices v in turn. □

Reusing the technique, we also obtain:

Lemma 2.9. *Let G be a simple planar graph. In linear time, we can add edges to G to obtain a simple, triangulated, planar graph.*

Proof. It is easy to add edges so that the resulting graph is connected, and then triangulated, in linear time. The only problem is that the resulting graph may be non-simple. Let e be an edge of a triangulated graph; removing e yields a degree-four face, which we can triangulate by inserting the unique edge $e' \neq e$; we call this procedure a *flip* of e.

For each vertex v, compute all loops and multiple edges incident with v, using the technique of the previous lemma. Now, we flip all loop edges incident with v (no such edge belongs to the original graph G). Furthermore, for each neighbor u of v, consider the set of edges E_{uv} with both u and v as endpoints. Assume $|E_{uv}| \geq 2$. The original graph G has at most one edge in E_{uv}; if G contains one edge of E_{uv}, we let e be that edge, otherwise we let e be an arbitrary edge of E_{uv}. Now flip all edges in $E_{uv} \setminus \{e\}$.

Figure 2.2. After flipping a multiple edge (left) or a loop (right) in a planar graph, the new edge is not a loop and is not a multiple edge.

The crucial observation is that none of these flips introduce loops or multiple edges, by planarity of the triangulated graph (Figure 2.2).

Iterating this process for each vertex v in turn, we obtain the desired linear-time algorithm. □

The previous lemma implies that, to prove Theorem 2.7, we can assume that G is triangulated. Another key ingredient for the proof of this theorem is the following inductive decomposition of a planar, simple, triangulated graph, depicted in Figure 2.3.

Proposition 2.10. *Let G be a planar, simple, triangulated graph. Let v_1 and v_2 be two vertices on its outer circuit. In linear time, we can order the vertices of G as $v_1, \ldots, v_n$ such that, for each $k \geq 3$, the subgraph G_k of G induced by $v_1, \ldots, v_k$ satisfies:*

- *G_k is connected;*
- *the boundary of G_k is a circuit;*
- *each inner face of G_k has degree three;*
- *v_{k+1} is in the outer face of G_k.*

The proof of this proposition rests on the following lemma.

Lemma 2.11. *Let G be a planar, simple graph; assume that the boundary of the outer face forms a circuit (without repeated vertices) C. Let v_1v_2 be an edge on C. There exists a vertex v on C, different from v_1 and v_2, that has exactly two neighbors on C.*

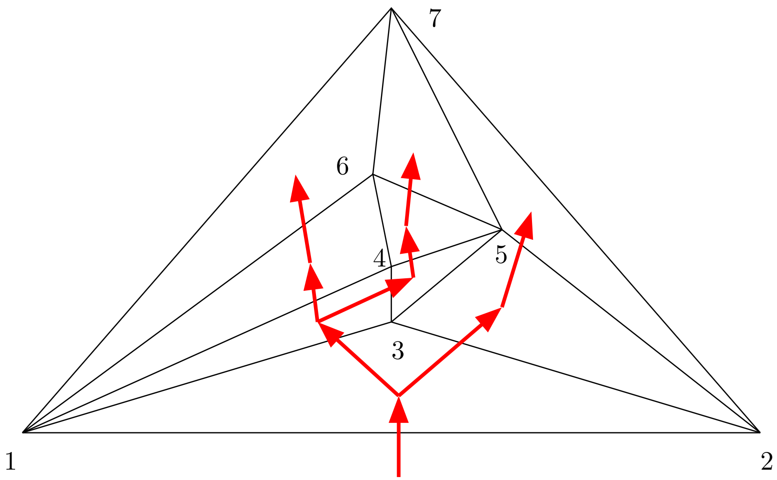

Figure 2.3. Illustration of Proposition 2.10. The directed tree is used later in the proof of Theorem 2.7.

Proof. If every vertex of C has exactly two neighbors of C, we are done. Let the vertices of C be $v_1 = w_1, \dots, w_m = v_2$, in this order. Consider an edge connecting w_i to w_j where $j - i$ is minimal but at least two. Then the only neighbors of w_{i+1} in C are w_i and w_{i+2} (Figure 2.4): None of $w_{i+3}, \dots, w_j$ can be a neighbor of w_{i+1} by minimality of $j - i$, and none of the other vertices on C either, by planarity. □

Proof of Proposition 2.10. We choose $v_n, \dots, v_3$ in this order by repeated applications of Lemma 2.11; the conditions are obviously satisfied.

To do this in linear total time, we maintain the following information on each vertex v of the current graph: Whether v belongs to the outer circuit and, if so, its number of neighbors on the outer circuit. We maintain a list of (pointers to) vertices on the outer circuit that have exactly two neighbors on the outer circuit; by Lemma 2.11, this list is never empty. The algorithm iteratively picks a vertex in the list, updates the data, and iterates until exactly three vertices are left.

This takes linear time, since each edge is considered only if one of the endpoints enters or leaves the circuit. □

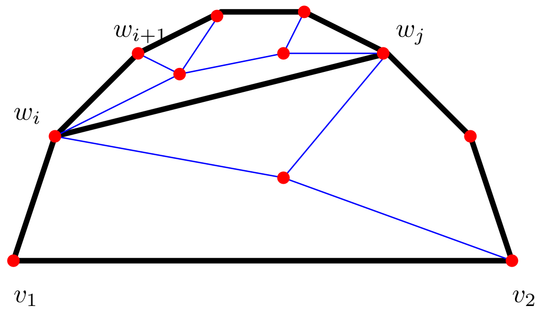

Figure 2.4. Illustration of the proof of Lemma 2.11.

Proof of Theorem 2.7. The algorithm iteratively embeds the subgraph G_k of G induced by $v_1, \dots, v_k$, where k goes from 3 to n. Actually, instead of computing x- and y-coordinates of the vertices, we compute y-coordinates of the vertices and *x-spans* of the edges, namely, the difference between the x-coordinates of their endpoints; trivially, this information is enough to recover the embedding.

Assume inductively that we already embedded G_k ($k \geq 3$) on the grid in such a way that:

1. The y-coordinates of v_1 and v_2 are zero;
2. If $v_1 = w_1, \dots, w_2, \dots, w_m = v_2$ are the vertices on the outer face of G_k, in cyclic order, then the x-spans of each edge $w_i w_{i+1}$ is positive;
3. each edge $w_i w_{i+1}$, $1 \leq i \leq m$, has slope $+1$ or -1.

Vertex v_{k+1} is incident, in G_{k+1}, to a contiguous set of vertices $w_p, \dots, w_q$ on the boundary of the outer face of G_k. Let $P(w_p, w_q)$ be the intersection point of the line of slope $+1$ passing through w_p with the line of slope -1 passing through w_q; Condition (3) implies that $P(w_p, w_q)$ has integer coordinates. Putting v_{k+1} at position $P(w_p, w_q)$ almost yields a planar drawing of G_{k+1}, except that it may fail to see, e.g., w_p. To avoid this problem (Figure 2.5), we shift vertices $w_1, \dots, w_p$ by one unit to the

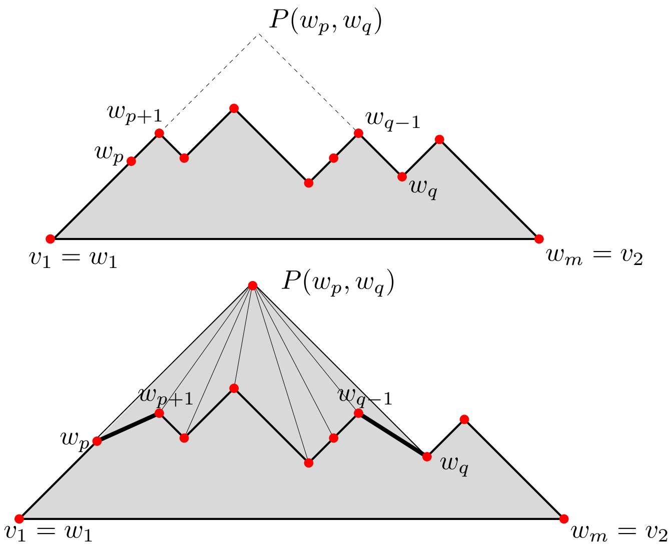

Figure 2.5. Illustration of the proof of Lemma 2.11.

left, so that the slope of $w_p w_{p+1}$ becomes now smaller than $+1$; and similarly we shift $w_q, \dots, w_m$ by one unit to the right. In our choice of representation of points with x-spans and y-coordinates, this takes constant time: Simply increase by one the x-span of $w_p w_{p+1}$ and of $w_{q-1} w_q$. The only problem is that the resulting drawing is inconsistent, so we need an *adjustment phase* to increase the x-spans of some internal edges. We first explain how to do this adjustment of the x-spans of internal edges at each step from G_k to G_{k+1}. However, for the purposes of an efficient algorithm, it will be useful to do these adjustments at once.

We maintain a spanning tree T^* of the dual of G_k, rooted at the outer face and oriented away from the root, as follows (Figure 2.3). Initially (say $k = 3$), there is one edge from the root outer face to the inner face, crossing edge $v_1 v_2$. When we add vertex v_{k+1}, for each newly created internal face of the drawing, we create an edge of T^* arriving to that face by crossing the unique edge incident to that face that belongs to G_k.

When adding edges in G_k to build G_{k+1}, the adjustment phase consists in increasing by one the x-span of the edges crossed by the subpath of T^* from the root to the first vertex incident to $(w_p w_{p+1})^*$, and similarly of the edges crossed by the subpath to the first vertex incident to $(w_{q-1} w_q)^*$. (Edges crossed by both subpaths have thus their x-span increased by two.) Combined with the initial shift of the boundary edges, this results in a shift of a "left" part of the graph to the left and of a "right" part of the graph to the right. To prove that this results in a valid embedding, it suffices to note that the following property is maintained during the algorithm: For every edge e^* of T^* oriented away from the root, the vertex of G_k to the left of e^* has smaller x-coordinate than the vertex of G_k to the right of e^*. It is clear that, at the end, the vertices are on an $O(n) \times O(n)$-grid.

To implement this idea in linear time, we first compute the x-spans and y-coordinates in $G_3, \dots, G_n$ *without* doing the adjustment steps; this takes $O(n)$ time. Omitting this adjustment step does not harm because, at each step, we only need to know that the x-spans and y-coordinates of the vertices on the outer face are correct. Afterwards, we need to increase the x-span of each edge e by the cumulated increase it would have received during all adjustment steps. This amounts to determining how many times e is crossed by the paths of T^* considered during the adjustment steps. For this purpose, during the incremental construction, we record, for each vertex of T^* other from the root, the number of times it appears as an endpoint of such a path. At the end of the incremental construction, we can by a simple search in T^* compute, for each edge of T^*, the number of times it is contained in a path. This takes linear time. □

2.3 Notes

The planarity testing algorithm is taken from [14, Section 3.3]. The graph drawing algorithm is due to de Fraysseix et al. [13], with simplifications from Castelli Aleardi et al. [6].

The fact that every planar graph without loops or multiple edges admits a *straight-line* embedding was shown a few decades before the discovery of the algorithm given above [22, 46, 51]. Actually, if G is a planar graph without loops or multiple edges with n vertices, a straight-line embedding exists where all vertices lie in the $(n-2) \times (n-2)$-grid [23]. Many other representations exist, such as *circle packing* representations: the vertices are mapped to non-overlapping disks in the plane, two of which are tangent if and only if an edge between the corresponding vertices exists (see Mohar and Thomassen [41, Chapter 2] for a proof and references).

Chapter 3

Efficient algorithms for planar graphs

In this chapter, we illustrate the general idea that algorithmic problems on graphs are easier to deal with when the graph is assumed to be planar. By Theorem 2.1, if we are given a planar graph G, we can compute in linear time a combinatorial map of G in the plane; therefore, we can assume that in all algorithms for planar graphs, a combinatorial map of the graph is given. More advanced algorithms will be described in Claire Mathieu's course.

3.1 Minimum spanning tree algorithm

Let $G = (V, E)$ be a cellular graph embedding in $\mathbb{S}^2$, with a weight function $w : E \to \mathbb{R}$ on its edges. Let n be its complexity.

Theorem 3.1. *A minimum spanning tree of G can be computed in $O(n)$ time.*

We note that, by Lemma 1.8, $E' \subseteq E$ is a minimum spanning tree of G if and only if $(E \setminus E')^*$ is a *max*imum spanning tree of G^* (where the weight of a dual edge equals the weight of the corresponding primal edge).

Exercise 3.2. ☆☆☆ Prove that a connected planar graph has either a vertex or a face with degree at most three.

We introduce two operations to transform a cellular graph embedding in $\mathbb{S}^2$ into another one. These operations (together with their reverses) are called *Euler operations*. Let e be an edge of G that is incident with two different faces. Then removing e yields a cellular graph embedding, denoted by $G\backslash e$. The dual operation is *contraction*: let e be an edge of G that is incident with two different vertices (i.e., that is not a loop), then we may contract e by identifying its two incident vertices; the resulting graph embedding is denoted by G/e. Obviously, these two operations preserve the planarity.

Proof of Theorem 3.1. The two following dual rules allow to build inductively the set of edges $T(G)$ of a minimum spanning tree of G:

- Let v be a vertex of G. If all edges incident with v are loops, then G has exactly one vertex, so there is a unique, empty, spanning tree. Otherwise, let e be a minimum-weight edge incident exactly once with v. Necessarily, edge e belongs to a minimum spanning tree of G. Hence $T(G/e) \cup e$ is a minimum spanning tree of G;
- let f be a face of G. If all edges incident with f have f on both sides, then G has exactly one face, so G is a tree, and there is a unique spanning tree, G itself. Otherwise, let e be a maximum-weight edge incident exactly once with f. Then e does not belong to a minimum spanning tree of G (because e^* belongs to a maximum spanning tree of G^*). It follows that $T(G \setminus e)$ is a minimum spanning tree of G.

The number of iterations of this algorithm is $O(n)$. Assuming we can pick a vertex v or a face f with degree $O(1)$ (whose existence is guaranteed by Exercise 3.2) in constant amortized time, we have a linear-time algorithm. Indeed, without loss of generality assume we have a vertex v with degree $O(1)$; the dual case is similar. Determining which edges incident to v are loops takes $O(1)$ time. If all of them are loops, then the recursion stops; otherwise, finding a minimum-weight edge e that is not a loop can clearly be done in $O(1)$ time. Also, contracting e can be done in $O(1)$ time, since there are $O(1)$ flags to update: this uses the fact that one vertex incident with e has degree $O(1)$.

It remains to explain how to compute in $O(1)$ amortized time a vertex or a face with degree at most three. For this purpose, we maintain a bucket B (a list) containing all vertices and faces of degree at most three (and possibly other vertices and faces, possibly some of them being destroyed in the course of the algorithm after they are put in the bucket). Initially, put all vertices and faces in B. When contracting or deleting an edge e, only the degrees of the vertices and faces incident with e can change, so we put them in the bucket before contracting or deleting e. Therefore in total $O(n)$ vertices and faces are put into B.

To find a vertex or face of degree at most three in the current graph, pick an element of B, check in $O(1)$ time whether it still belongs to the current graph and, if so, whether it has degree at most three. If it is not the case, remove it from B and proceed with the next element. Since $O(n)$ elements in total are put in B, also $O(n)$ elements are removed from B, so the total time spent to find vertices and faces with degree at most three is $O(n)$. □

3.2 Graph coloring

Let $G = (V, E)$ be a graph and $k \geq 1$ be an integer. A *coloring* of G with k colors is a map $V \to \{1, \ldots, k\}$ such that adjacent vertices are mapped to different integers ("colors"). If a graph has a coloring with k colors, we say that it is *k-colorable*.

In coloring problems, we can safely ignore graphs with loops (edges incident twice to the same vertex), because such graphs are not k-colorable, for any k. *In this section, we implicitly only consider graphs without loops*, and all subsequent graphs built in the proofs have this property.

Determining whether a graph is k-colorable is NP-hard, except for $k = 1$ (it is equivalent to have no edge in the graph) and $k = 2$ (it is equivalent to have a bipartite graph, a problem easily solvable in linear time). For planar graphs, life seems to be no easier: It is NP-hard to decide whether a planar graph is 3-colorable [27], by reduction from 3-SAT.

However, it is a remarkable fact that every planar graph is 4-colorable [1]; this was proved by Appel and Haken, heavily relying on computer assistance (up to date, no proof is known that does not involve a lot of case distinctions). We shall prove that every graph is 5-colorable, and give an algorithm to color a planar graph in linear time, assuming a combinatorial map is given.

Theorem 3.3. *Every planar graph is 5-colorable.*

Proof. Consider a planar drawing of a graph G in the plane. We can assume without loss of generality that G is connected and has no face (including the outer face) of degree one or two. Let v, e, and f be the number of vertices, edges, and faces of G. Euler's formula $v - e + f = 2$ and double-counting of the edge-face incidences $2e \geq 3f$ implies $e \leq 3v - 6$ and thus the average degree of a vertex, $2e/v$, is strictly less than 6. Thus, G has at least one vertex of degree at most five. This directly implies that G is 6-colorable, since if x is a vertex of degree at most five, we can assume by induction that the graph $G - x$ obtained from G by removing x and its incident vertices is 6-colorable, and then color x with one color not used by any of its neighbors. To prove that every planar graph is 5-colorable, we only need to refine the argument slightly.

If G has a vertex incident with at most four distinct vertices, then by induction we are done. So let x be a vertex of degree exactly five, with distinct neighbors $v_1, \ldots, v_5$ in clockwise order around x. Let a 5-coloring of $G - x$ be given. If $v_1, \ldots, v_5$ do not have distinct colors, then at least one color remains to color x, so we are done. So assume (up to permutation) that v_i bears color i.

Let G_{13} be the subgraph of $G - x$ induced by the vertices colored 1 and 3. Assume first that there is no path in G_{13} connecting v_1 to v_3. We can exchange colors 1 and 3 in the component of G_{13} that contains v_1 (this is clearly valid). Now, both v_1 and v_3 are colored 3, which frees one color for vertex x, and we are done.

On the other hand, if v_1 and v_3 are connected in G_{13}, then we claim that v_2 and v_4 cannot be connected in G_{24} (the subgraph of $G - x$ induced by the vertices colored 2 and 4), which implies that we can use the same trick with v_2 and v_4 in place of v_1 and v_3.

To prove the claim, we assume the contrary: There are two disjoint paths p_{13} and p_{24} in $G - x$ connecting pairs (v_1, v_3) and (v_2, v_4) respectively. We can modify $G - x$ to exhibit a planar graph that is K_5, the complete graph with five vertices (Figure 3.1), which is a contradiction (Exercise 1.10).

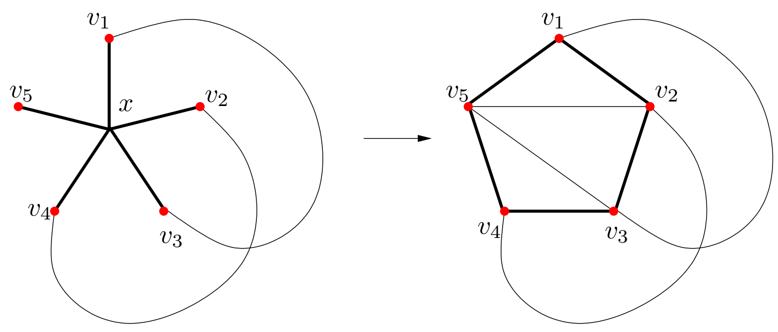

Figure 3.1. The last step in the proof of Theorem 3.3.

To do that, take the graph with vertex set $\{x, v_1, v_2, v_3, v_4\}$ and with the following edges: x is connected to all other vertices via edges drawn like in G; v_1 and v_3 are connected via p_{13}, similarly v_2 and v_4 are connected via p_{24}. Moreover, (v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1), (v_2, v_5), and (v_3, v_5) can be connected together without crossing other edges because they appear in cyclic order around x. This is K_5. □

We can actually obtain an efficient 5-coloring algorithm:

Theorem 3.4. *Every planar graph, given in the form of a combinatorial map, can be 5-colored in linear time.*

We will rely on the following independent proposition.

Proposition 3.5. *Let G be a planar graph with no face of degree one or two. Then G has either a vertex of degree at most four, or a vertex of degree five incident to two vertices of degree at most six.*

(Precisely, this should be understood as follows: among the three vertices involved in the second alternative, some of them can be the same, however the two edges involved should be distinct.)

Proof. It suffices to prove the result assuming G is triangulated. We proceed by contradiction, assuming that such configurations cannot occur.

Let us put a *charge* equal to 6 in each triangle. Now each triangle T sends its charge to all its incident vertices of degree five and six, in a way that the degree-5 incident vertices get all the same charge from T, the degree-6 incident vertices get all the same charge from T, and any degree-5 incident vertex gets twice the charge of a degree-6 incident vertex from T. (If T has no vertex of degree 5 or 6, the charge remains in T.)

After this operation, each degree-6 vertex v gets charge at least 2 from each of its incident triangles. Indeed, excepting v, such a triangle can be incident to at most two other degree-6 vertices, or to one degree-5 vertex and to a vertex of degree at least 7. Therefore, each degree-6 vertex gets charge at least 12. Not that the reasoning is also valid if some triangles around v are identified.

Also, each degree-5 vertex v gets charge at least 24. Indeed,

- If it is not adjacent to any degree-5 or degree-6 vertex, except possibly itself, it gets all the charge from its incident triangles, which is 30;
- if it is adjacent to a degree-5 vertex $w \neq v$, then it has no other vertex of degree 5 or 6 around, so it gets all the charge from the 3 triangles not incident to w and half the charge from the 2 triangles also incident to w, and thus 24;
- if it is incident to a degree-6 vertex w, similarly, it gets all the charge from the 3 triangles not incident to w and 2/3 of the charge from the 2 triangles also incident to w, which is 26.

Let t be the number of triangles in G, and n_i be the number of vertices of degree i in G. The previous discussion implies $6t \geq 24n_5 + 12n_6$, or equivalently $t \geq 4n_5 + 2n_6$ $(*)$.

On the other hand, Euler's formula can be rewritten in the two following ways:

- $\sum_i (2-i)n_i = 4 - 2t$, since $2e = \sum_i in_i$;
- $\sum_i 2n_i = t + 4$, since $2e = 3t$.

Eliminating n_7 in these two linear equations yields $\sum_i (14-2i)n_i = t + 28$.

Since $n_i = 0$ for $i \leq 4$, this implies $4n_5 + 2n_6 - 2n_8 - 4n_9 - \ldots = t + 28$. In particular, $t < 4n_5 + 2n_6$, contradicting $(*)$. □

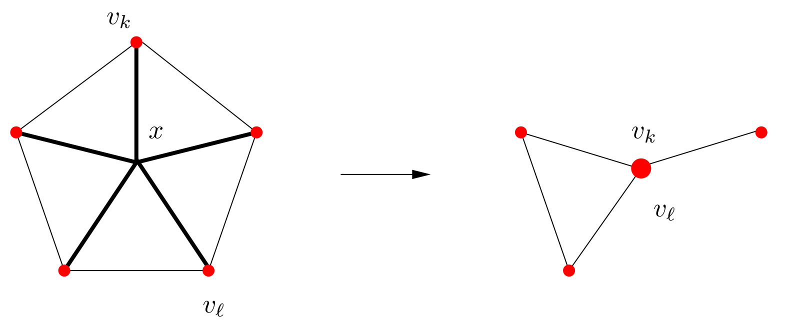

Figure 3.2. Illustration of the inductive construction in the proof of Theorem 3.4.

Proof of Theorem 3.4. We can assume that our input planar graph G is triangulated. Indeed, we can without harm remove any edge bounding a degree-one face, and, for any set of parallel edges forming adjacent faces of degree two, we can remove all but one such edges; finally, we may triangulate the remaining edges without adding loops (as in the proof of Lemma 2.9). This can be done in linear time.

We first describe the high-level approach, without worrying about complexity. We also proceed by induction, like in the proof of Theorem 3.3. Let x be a vertex of G obtained by Proposition 3.5. If x has at most four distinct neighbors, then we are done by applying induction to $G - x$ (after triangulating the new face). Otherwise, x has degree five, and five distinct neighbors $v_1, \ldots, v_5$ in this cyclic order around x, two of which, say v_1 and v_i $(i \in \{2, 3\})$, have degree at most six.

By planarity, if $i = 2$, then either v_1 and v_3 are not adjacent, or v_2 and v_4 are not adjacent. If $i = 3$, then either v_1 and v_4 are not adjacent, or v_3 and v_5 are not adjacent. So let $k, \ell \in \{1, \ldots, 5\}$ such that v_k and v_ℓ are not adjacent and v_k has degree at most six.

Let H be the graph obtained from $G - x$ by identifying v_k and v_ℓ, as in Figure 3.2 (this operation preserves planarity since v_k and v_ℓ belong to the same face of $G - x$, and no loop is created since v_k and v_ℓ are not adjacent in G). We apply induction to H. After H is 5-colored, this corresponds to a coloring of $G - x$ where v_k and v_ℓ have the same color, which leaves one color free to color x.

If we omit the time taken to find vertex x, then this algorithm can be implemented in linear time. Indeed, finding which vertices adjacent to x have degree at most six takes constant time. Using the fields vu described in Section 1.2.1, we can test whether the neighbors of x are distinct in constant time. We can also test whether two given vertices are adjacent in constant time if one of the vertices has bounded degree; this allows to determine v_k and v_ℓ in constant time. Then identifying v_k to v_ℓ takes constant time because v_k has bounded degree (this latter property is needed since updating the fields vu takes time linear in the smaller degree of v_k and v_ℓ).

There remains to compute vertex x efficiently. To do this, we maintain, during the algorithm, a stack (a linked list, for example) containing the vertices of degree at most four, and the vertices of degree five adjacent to two vertices of degree at most six. At each step, a constant number of vertices can enter or leave the stack, which can be updated in constant time. □

3.3 Minimum cut algorithm

We now give an efficient algorithm for computing minimum cuts in planar graphs.

Before that, we need to state without proof a result on shortest paths in planar graphs. Let $G = (V, E)$ be a connected graph where each edge has a non-negative *length* (also called *weight*), and let s be a vertex of G. A *shortest path tree* is a spanning tree of G rooted at s that contains a shortest path from s to each vertex in G. Dijkstra's algorithm (with the appropriate data structure for the priority queue, for example Fibonacci heaps) allows to compute a shortest path tree in $O(|E| + |V| \log|V|)$ time. The following result, which is (fortunately) admitted, improves the result for planar graphs.

Theorem 3.6. *Given a graph cellularly embedded in $\mathbb{S}^2$, a shortest path tree from a given vertex can be computed in time linear in the complexity of the graph.*

We shall use this result to prove the following theorem.

Theorem 3.7. *Let $G = (V, E)$ be a weighted planar graph of complexity n, cellularly embedded in $\mathbb{S}^2$. Let s and t be two vertices of G. The problem of computing a minimum-weight (s,t)-cut of G can be solved in $O(n \log n)$ time.*

To prove Theorem 3.7, we first dualize the problem in the following proposition, which is rather intuitive but not so easy to prove formally. Henceforth, let $G = (V, E)$ be a weighted planar graph, and let F be the faces of G.

Proposition 3.8. *$X \subseteq E$ is an (s,t)-cut in G if and only if X^* contains the edge set of some circuit of G^* separating s and t.*

Proof. If X^* contains (the edge set of) a circuit in G^* separating s and t, then any (s,t)-path in G must cross an edge in X^*, and thus contain an edge in X, so X is an (s,t)-cut.

Conversely, let X be an (s,t)-cut; we will prove that X^* contains the edge set of a circuit in G^* separating s and t. Without loss of generality, we may assume that X is *inclusionwise minimal* among all (s,t)-cuts.

First, label "S" each face v^* of G^* such that there is, in G, an (s,v)-path avoiding X. Similarly, label "T" each face v^* of G^* such that there is, in G, a (v,t)-path avoiding X. Since X is a cut, no face of G^* is both labeled "S" and "T". Any edge of G^* incident to faces labeled differently must be in X^*. Therefore, by minimality of X^*, each face of G^* is labeled either "S" or (exclusive) "T", and X^* is the set of edges incident to faces with different labels.

Let S be the subset of the plane made of the faces of G^* labeled "S", together with the *open* edges of G^* whose incident faces are both labeled "S". Define similarly T. Thus S and T are disjoint, connected subsets of the plane. Let f^* be a vertex of G^*; we claim that there cannot be four faces incident to f^* that belong respectively, in cyclic order around the vertex, to S, T, S, and T. Indeed, if the opposite assertion holds, then by connectivity of S, there is a closed curve in $S \cup \{f^*\}$ that goes through f^* and has faces of T on both sides of it, which contradicts the connectedness of T by the Jordan curve theorem.

Thus, X^* is a union of vertex-disjoint circuits in G^*; let γ be one such circuit. Since each edge of γ is incident to one face labeled "S" and one face labeled "T", $\gamma \subseteq X^*$ separates s from t. □

We now reformulate the problem in terms of curves crossing G. More precisely, we consider closed curves in *general position* with respect to G, which do not meet any vertex of G and intersect the edges of G at finitely many points, where they *cross*. The *length* of such a closed curve is the sum of the weights of the edges of G crossed by that curve, counted with multiplicity. Computing shortest paths between two points in this setting can be done in $O(n)$ time by applying Theorem 3.6 in the dual graph.

Proposition 3.9. *Let γ be a simple closed curve in general position with respect to G; assume that γ has minimum length among all such curves that separate s from t. Then the set of edges of G crossed by γ is a minimum-weight (s,t)-cut in G.*

Proof. The set of edges of G crossed by γ is an (s,t)-cut, by the Jordan curve theorem. Conversely, if we have a minimum-weight (s,t)-cut in G, Proposition 3.8 implies that its dual contains a circuit separating s and t, which corresponds to a simple closed curve γ separating s and t whose length is the same as the weight of the cut. □

Now, we view G as embedded on the sphere, and we remove two small disks around s and t. We now have an embedding of G on an annulus A, and by Proposition 3.9 it suffices to compute a shortest simple closed curve in general position with respect to G that goes "around" the annulus. The general idea of the algorithm is depicted in Figure 3.3. Let p be some shortest path from an arbitrary point on one boundary to an arbitrary point on the other boundary (again, where the length is measured by the sum of the weights of G crossed by p). Let D be the disk obtained by cutting the annulus along p; let p' and p'' be the pieces of its boundary corresponding to p.

3.3.1 Naïve algorithm

The following lemma implies that some shortest simple closed curve separating the two boundaries of A corresponds, in D, to a shortest path between a pair of "twin" points of p' and p''.

Lemma 3.10. *Some shortest closed curve separating the two boundaries of A is simple and crosses p exactly once.*

Proof. Let γ be a shortest closed curve separating the two boundaries. The image of γ in D (after cutting along p) must contain a simple path q from p' to p'', for otherwise γ would not separate the boundaries of A.

Let γ' be a closed curve obtained by connecting the endpoints of q with a shortest path running along p. This closed curve is simple, separates the boundaries of A, crosses p exactly once, and is no longer than γ, since γ has at least the length of q plus the length necessary to connect the endpoints of q. □

This allows a naïve $O(n^2)$-time algorithm: Let $k \geq 0$ be the number of edges of G crossed by p; let $v_0, \ldots, v_k$ be points on p, in this order on p, such that the subpath between v_i and v_{i+1} crosses exactly one edge of G. Compute all shortest paths between v_i' and v_i'' (the twin points corresponding, in D, to v_i), and take a shortest such path. The running-time follows since $k = O(n)$ and since shortest paths can be computed in linear time in planar graphs (Theorem 3.6).

3.3.2 Divide-and-conquer algorithm

To beat this quadratic bound, we use a "divide-and-conquer" strategy based on the following lemma, illustrated in Figure 3.4.

Lemma 3.11. *Let x, y, and z be points on p, in this order, and (x', x''), (y', y''), and (z', z'') be the corresponding twin points on D. Let p_x and p_z be disjoint simple shortest paths in D between the corresponding twin pairs (x', x'') and (z', z''). Then some shortest path p_y between the twin pairs (y', y'') crosses neither p_x nor p_z, and is simple.*

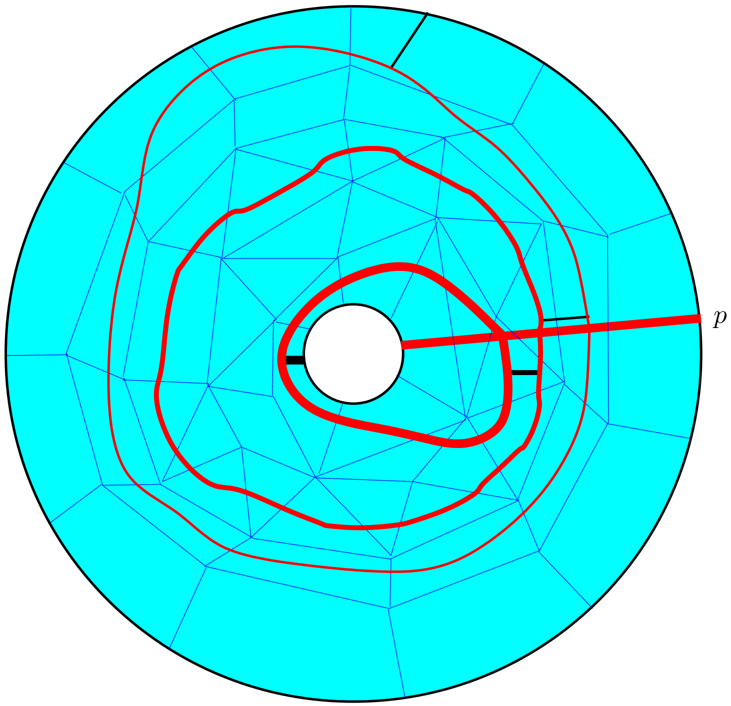

Figure 3.3. Overview of the algorithm of Theorem 3.7. The initial annulus is recursively cut into smaller annuli, until one of the two conditions for stopping the recursion happens; then computing one or two new shortest paths (not shown here) concludes.

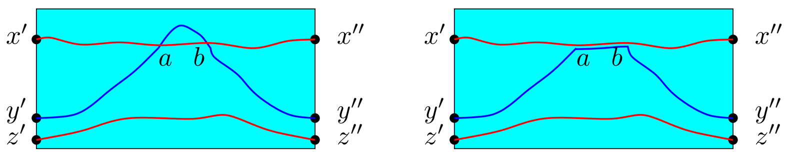

Figure 3.4. Illustration of Lemma 3.11.

Proof. Let p_y be an arbitrary shortest path between y' and y''. It crosses p_x an even number of times, because y' and y'' are not separated by p_x in D. If p_y crosses p_x at least twice, at points a and b, we may replace the part of p_y between a and b by a path running along p_x, removing two crossings between p_x and p_y; this does not decrease the length of p_y, since p_y is a shortest path; and this does not introduce additional crossings between p_y and p_z, since p_y and p_z are disjoint.

So by induction, we may assume that p_y is disjoint from p_x. Similarly, we may assume that p_y is also disjoint from p_z. Finally, we may remove the loops in p_y to make it simple. □

We first describe the two base cases of the recursion, which can be solved in linear time:

1. If $k = O(1)$ (for example, if $k \leq 1$), we may conclude by computing all shortest paths, in D, between each pair of twin vertices v_i' and v_i'', and taking the shortest of these paths;
2. similarly, if there is a face f of the graph incident with both boundaries of A, then the shortest closed curve has to go through this face; we can conclude by cutting the annulus A into a disk along a path entirely contained in f and computing a shortest path, in this disk, between the two copies of the path.

Otherwise, we consider vertex $v := v_{\lfloor \frac{k}{2} \rfloor}$ and compute a shortest path in D between the points v' and v'' corresponding to v on p' and p'', respectively; this is thus a shortest closed curve γ passing through v and crossing p exactly once. Let A_1 and A_2 be the two annuli obtained by cutting A along γ. The previous lemma implies that it suffices to recursively compute the shortest closed curve separating the two boundaries of A_1 and of A_2 (using the pieces of p within A_1 and A_2 as new shortest paths), and to take the shortest of these closed curves. This concludes the description of the algorithm.

3.3.3 Correctness and complexity analysis

Proof of Theorem 3.7. The execution of the algorithm can be represented with a binary tree, where each node corresponds to an annulus. The root corresponds to A; internal nodes always have two children; leaves correspond to the base case of the recursion.

The algorithm terminates, since the path p crosses at most $\lceil k/2^i \rceil$ edges at the ith level in the recursion tree, and by base case (1). In fact, this proves that there are at most $\lceil \log k \rceil = O(\log n)$ levels in the recursion tree. The correctness follows from Proposition 3.9 and from the above considerations. There remains to show the $O(n \log n)$ complexity.

Consider a given edge e of G. At some level r of the recursion tree, that edge is cut by some closed curves into a number of subedges $e_1, \ldots, e_j$ ($j \geq 1$), all belonging to distinct annuli at level r. However, only the subedges e_1 and e_j can belong to an annulus that is an internal node of the recursion tree: the other ones end in base case (2). Therefore, e occurs at most twice in total in the annuli that are internal nodes at level r, and thus at most four times in total in the annuli at level $r+1$. Hence, the total number of non-boundary edges of the annuli at a given level is at most $4n$.

Furthermore, every boundary edge of an annulus can be charged to an adjacent non-boundary edge of that annulus, in a way that every non-boundary edge is charged at most twice. Thus, the total number of boundary edges of the annuli at a given level is at most $8n$.

Bottom line: the total number of edges of all annuli at a given level is $O(n)$; by Euler's formula, this is also a bound on the sum of the complexities of all annuli at a given level. Since, at each node, all the operations (cutting and shortest paths computations) take linear time in the complexity of the annulus, the overall complexity of the algorithm is proportional to the total complexity of the annuli appearing in the recursion tree, which is made of $O(\log n)$ levels, each containing annuli of total complexity $O(n)$. $\square$

3.4 Notes

The minimum spanning tree algorithm described above is based on Matsui [40] (see also Cheriton and Tarjan [9] for a more complicated, but more general, algorithm). Actually, the same technique shows that a minimum spanning tree of a graph cellularly embedded on a surface of genus g can be computed in $O(gn)$ time. (See Chapter 4 for more on surfaces.) On arbitrary graphs, things are more complicated: there is a randomized algorithm with linear time [32], and a deterministic algorithm with almost linear time (where "almost" means up to a factor involving the inverse Ackermann function) [8].

Proposition 3.5 is due to Franklin [25]. The linear-time 5-coloring algorithm is a variant of an algorithm sketched by Robertson et al. [44], which seems to have a subtle flaw. In that paper, a weaker version of Proposition 3.5 is used; the algorithm still needs to identify two vertices v_k and v_ℓ, but with that weaker version, none of these vertices can be assumed to have bounded degree. Thus updating the vu fields requires linear time. Such vu fields are needed because we must be able to test whether two vertices are adjacent in constant time, assuming (only) one of these vertices has bounded degree.

The algorithm for finding a minimum cut in a planar graph was found by Reif [43]. The presentation above differs slightly, by using closed curves in general position with respect to G; this concept will be refined when we introduce the notion of cross-metric surface in Chapter 5.1. Frederickson [26] provides a different method. The proof of Proposition 3.8 is often neglected, and the proof used here is a variant of the one found in Colin de Verdière and Schrijver [12, Lemma 7.2].

A shortest circuit in a graph separating two given faces translates, in the dual, to a minimum cut separating the two dual vertices. By the max-flow min-cut theorem, a maximum flow yields immediately a minimum cut, but not conversely. A very recent paper shows that both the minimum cut and maximum flow problems can be solved in $O(n \log \log n)$ in planar graphs [31].

Chapter 4

Topology of surfaces

4.1 Definition and examples

A *surface* is a topological space in which each point has a neighborhood homeomorphic to the unit open disk $\{(x,y)\in\mathbb{R}^2 \mid x^2+y^2<1\}$. We only consider *compact* surfaces in this chapter (and even later, unless specifically noted).

Examples of surfaces are the sphere, the torus, and the double torus: these are compact, connected, orientable (to be defined later) surfaces with zero, one, and two handles, respectively (see Figure 4.1). The classification of surfaces (Theorem 4.5) asserts that two compact, connected, and orientable surfaces are homeomorphic if and only if they have the same number of "handles".

Despite the figures, note that a surface is "abstract": the only knowledge we have of it is the neighborhoods of each point. A surface is not necessarily embedded in $\mathbb{R}^3$. Actually, the non-orientable surfaces cannot be embedded in $\mathbb{R}^3$.

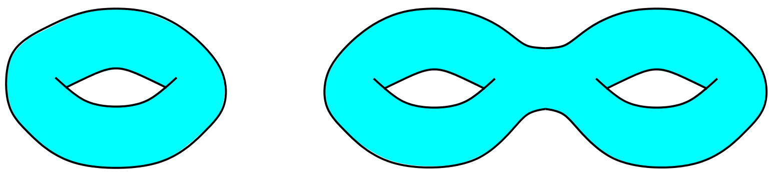

Figure 4.1. A torus and a double-torus.

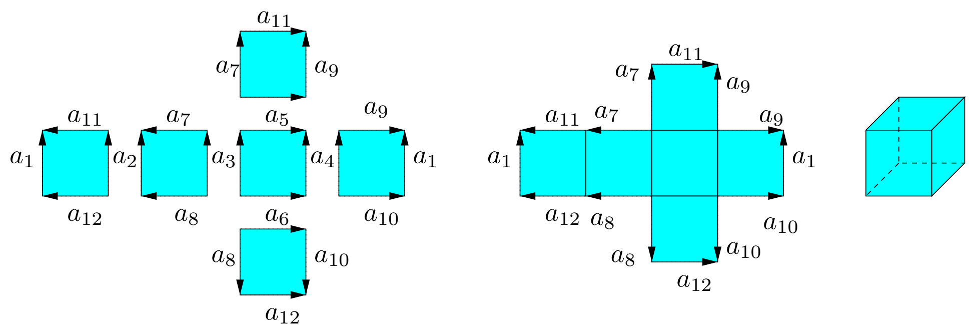

Figure 4.2. A polygonal schema of a graph embedded on a sphere (the graph of the cube) is: $a_2a_{11}\bar{a}_1\bar{a}_{12}$, $a_3a_7\bar{a}_2\bar{a}_8$, $a_4\bar{a}_5\bar{a}_3a_6$, $a_1\bar{a}_9\bar{a}_4a_{10}$, $a_9\bar{a}_{11}\bar{a}_7a_5$, and $a_{12}\bar{a}_{10}\bar{a}_6a_8$.

4.2 Surface (de)construction

4.2.1 Surface deconstruction

A graph embedded on a surface is *cellularly embedded* if all its faces are topological disks. As in the case of the plane, we may consider the combinatorial map of a graph cellularly embedded on a surface; the data structures are identical. The dual graph is defined similarly.

The *polygonal schema* associated with a cellular graph embedding is defined as follows: assign an arbitrary orientation to each edge; for each face, record the cyclic list of edges around the face, with a bar if and only if it appears in reverse orientation around the face. See Figure 4.2.

4.2.2 Surface construction

Conversely, the data of a polygonal schema allows to build up a surface and the cellular graph embedding. More precisely, let S be a finite set of *symbols* and let $\bar{S}=\{\bar{s}\mid s\in S\}$. Let R be a finite set of *relations*, each relation being a non-empty word in the alphabet $S\cup\bar{S}$, so that for every

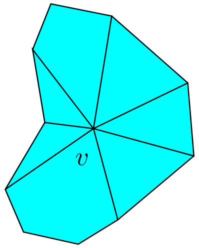

Figure 4.3. The "corners" incident to some vertex v can be ordered cyclically.

$s \in S$, the total number of occurrences of s plus the number of occurrences of $\bar{s}$ in R is exactly two.

For each relation of size n, build an n-gon; label its edges by the elements of R, in order, the presence of a bar indicating the orientation of the edge (see Figure 4.2). (Polygons with one or two sides are also allowed.) Now, identify the "twin" edges of the polygons corresponding to the same symbol in S, taking the orientation into account. (As a consequence, vertices get identified, too.)

Lemma 4.1. *The topological space obtained by the above process is a compact surface.*

Proof. Let X be the resulting topological space; X is certainly compact. We have to show that every point of X has a neighborhood homeomorphic to the unit disk. The only non-obvious case is that of a *vertex* v in X, that is, a point corresponding to a vertex of some polygons. But it is not hard to prove that a neighborhood of v is an *umbrella*: the "corners" (vertices) of the polygons corresponding to v can be arranged into a cyclic order; see Figure 4.3. □

We admit the following converse:

Theorem 4.2 (Kerékjártó-Radó; see Thomassen [49] or Doyle and Moran [15])**.** *Any compact surface is homeomorphic to a surface obtained by the gluing process above.*

This amounts to saying that, on any compact surface, there exists a cellular embedding of a graph. Equivalently, every surface can be triangulated.

Figure 4.4. (a) The orientations of these two faces (triangles) are compatible. (b) Two non-compatible orientations of the faces. A surface is orientable if there exist orientations of all faces that are compatible.

4.3 Classification of surfaces

4.3.1 Euler characteristic and orientability character

Let G be a graph cellularly embedded on a compact surface $\mathscr{S}$. The *Euler characteristic* of G equals $v - e + f$, where v is the number of vertices, e is the number of edges, and f is the number of faces of the graph.

Proposition 4.3. *The Euler characteristic is a* topological invariant*: it only depends on the surface $\mathscr{S}$, not on the cellular embedding.*

Sketch of proof. The Euler characteristic is easily seen to be invariant under Euler operations. The result is then implied by the following claim: any two cellular embeddings on a given surface can be transformed one into the other via a finite sequence of Euler operations. Proving this is not very difficult but requires some work; a key property is that one can assume both embeddings to be piecewise linear with respect to a given triangulation of the surface (using for example the method by Epstein [17, Appendix]). □

G is *orientable* if the boundary of its faces can be oriented so that each edge gets two opposite orientations by its incident faces (Figure 4.4). The orientability character is a topological invariant as well; the same proof as that of Proposition 4.3 works, but it can also be proven directly:

Exercise 4.4. ☆☆ G is orientable if and only if no subset of $\mathscr{S}$ is a Möbius strip.

4.3.2 Classification theorem

Theorem 4.5. *Every compact, connected surface $\mathscr{S}$ is homeomorphic to a surface given by the following polygonal schemata, called* canonical *(each made of a single relation):*

i. $a\bar{a}$ *(the sphere; Euler characteristic* 2*, orientable);*

ii. $a_1b_1\bar{a}_1\bar{b}_1\dots a_gb_g\bar{a}_g\bar{b}_g$*, for* $g \geq 1$ *(Euler characteristic* $2-2g$*, orientable);*

iii. $a_1a_1\dots a_ga_g$*, for* $g \geq 1$ *(Euler characteristic* $2-g$*, non-orientable).*

Furthermore, the surfaces having these polygonal schemata are pairwise non-homeomorphic. In particular, two connected surfaces are homeomorphic if and only if they have the same Euler characteristic and the same orientability character.

In the above theorem, g is called the *genus* of the surface; by convention $g = 0$ for the sphere. The orientable surface of genus g is obtained from the sphere by cutting g disks and attaching g "handles" in place of them. Similarly, the non-orientable surface of genus g is obtained from the sphere by cutting g disks and attaching g Möbius strips (since a Möbius strip has exactly one boundary component). See Figure 4.5. See also Figure 4.6 for a representation of a double-torus in canonical form.

Proof. Let $\mathscr{S}$ be a compact, connected surface, and G be a graph embedded on $\mathscr{S}$ (by Theorem 4.2). By iteratively removing edges incident with different faces, we may assume that G has only one face.[1] By iteratively contracting edges incident with different vertices, we may assume that G has only one vertex and one face[2] (unless this yields a sphere, so the polygonal schema is $a\bar{a}$ — actually, we could say that the polygonal schema made of the empty relation is a degenerate polygonal schema for the sphere). The surface $\mathscr{S}$ cut along G is therefore a topological disk; we use cut-and-paste operations on this polygonal schema to obtain a standard form.

[1]This amounts to removing all primal edges of a spanning tree in the dual graph.

[2]This amounts to contracting the edges of a spanning tree in the primal graph.

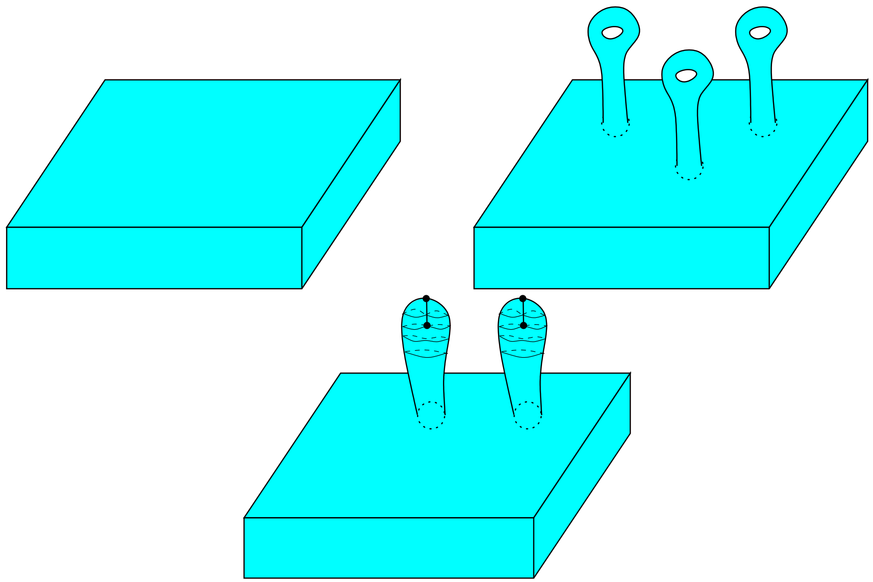

Figure 4.5. Every compact, connected surface is obtained from a sphere by removing disjoint disks and attaching handles (orientable case) or Möbius strips (non-orientable case). However, the non-orientable surfaces are not embeddable in $\mathbb{R}^3$.

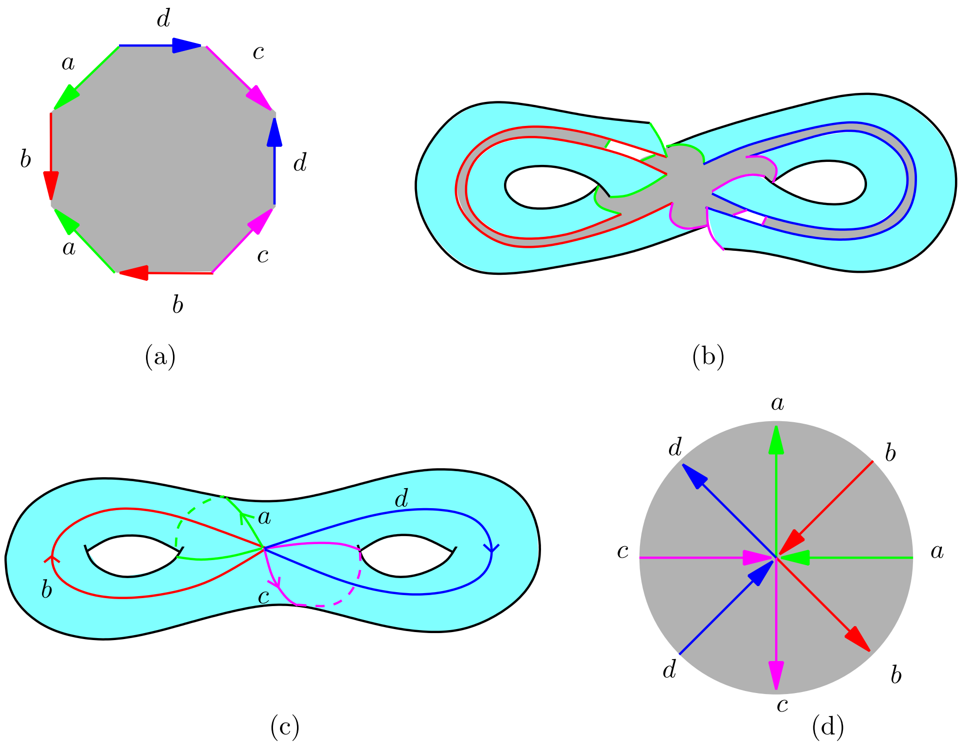

Figure 4.6. (a) A canonical polygonal schema of the double torus. (b) The identification of the edges of the schema. (c) The actual graph embedded on the double torus. (d) Closeup on the order of the loops around the basepoint of the surface, as seen from below; it can be derived directly from (a).

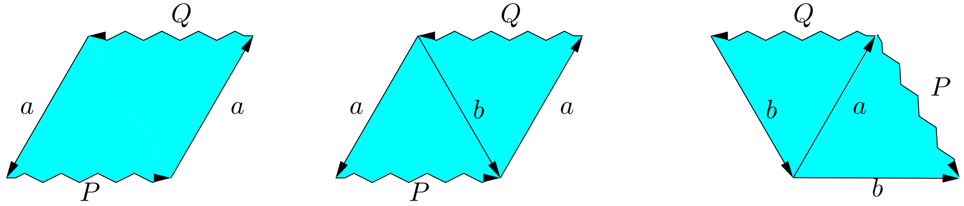

Figure 4.7. The classification of surfaces: grouping the twin edges appearing with the same orientation.

If the polygonal schema has the form $aPaQ$ (where P and Q are possibly empty sequences of symbols), then we can transform it into $bb\bar{P}Q$ (Figure 4.7)—$\bar{Q}$ denotes the symbols of Q in reverse order, inverting also the presence or absence of a bar above each letter. So inductively, we may assume that each pair of symbols appearing in the polygonal schema with the same orientation is made of two consecutive symbols. We still have one face and one vertex.

Assume some edge appears twice in the polygonal schema with opposite orientations: $aP\bar{a}Q$. Then P and Q must share an edge b, because otherwise the endpoints of a would not be identified on the surface. By the preceding step, b must appear in opposite orientations in P and Q, so we may assume that the polygonal schema has the form $aPbQ\bar{a}R\bar{b}S$. Then, by further cut-and-paste operations, we may transform the polygonal schema into $dc\bar{d}\bar{c}RQPS$ (Figure 4.8). We still have one face and one vertex, and can iterate the process. After this stage, the polygonal schema is the concatenation of blocks of the form aa and $ab\bar{a}\bar{b}$.

If there are no blocks of the form aa, or no blocks of the form $ab\bar{a}\bar{b}$, then we are in form (ii) or (iii), respectively. Otherwise, onea part of the boundary of the polygonal schema has the form $aabc\bar{b}\bar{c}$. We may transform it to $\bar{d}\bar{c}\bar{b}\bar{d}\bar{b}\bar{c}$ (Figure 4.9), and, applying the method of Figure 4.7 to b, c, and d in order, we obtain that we replaced the part of the boundary we considered into $eeffgg$; the other part of the boundary is unchanged. So iterating, we may convert the polygonal schema into form (iii).

The Euler characteristics and orientability characters of the surfaces are readily computed, since the canonical polygonal schemata have exactly one

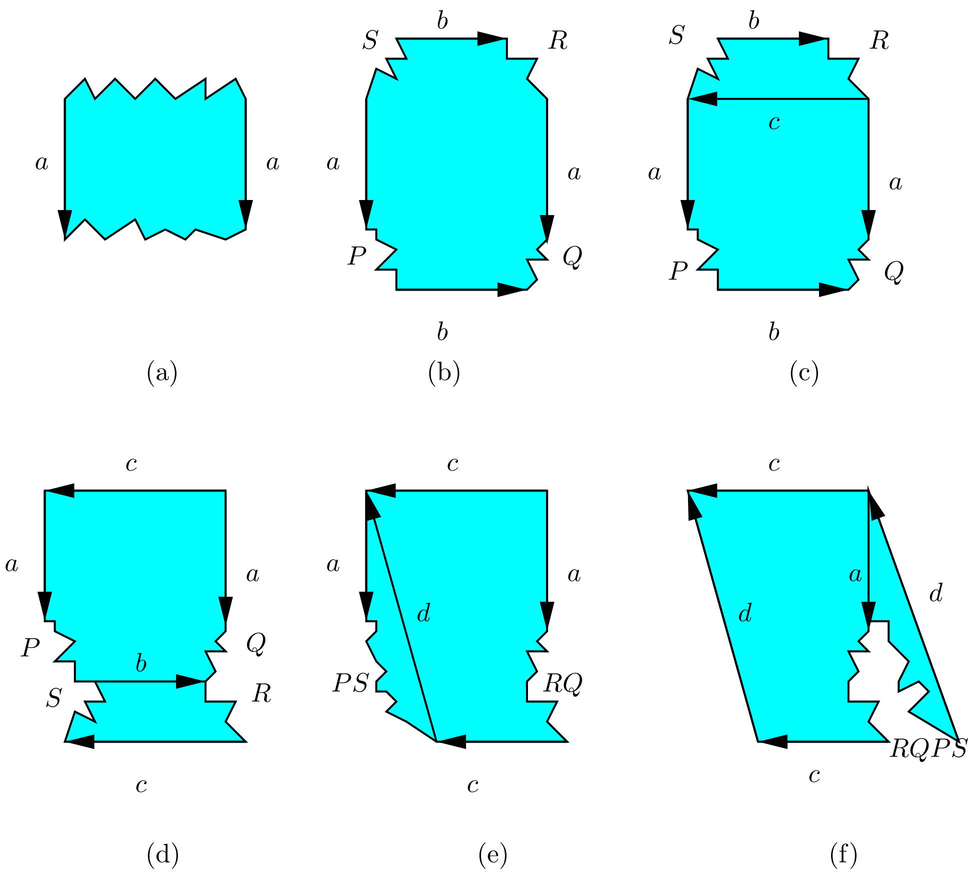

Figure 4.8. The classification of surfaces: grouping pairs of twin edges appearing with different orientations.

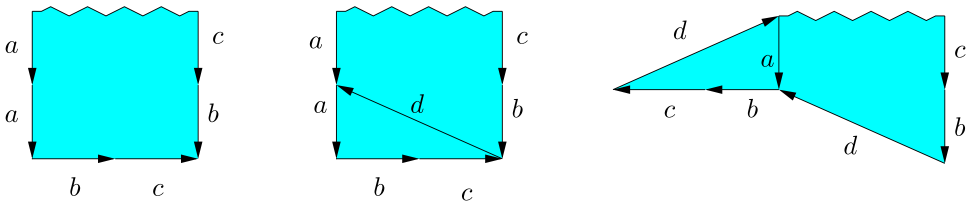

Figure 4.9. The classification of surfaces: transforming one form into the other.

vertex and one face. Since two distinct canonical polygonal schemata do not have the same Euler characteristic and the same orientability character, they cannot be homeomorphic, by Proposition 4.3 and Exercise 4.4. □

Example 4.6.

- The orientable surface with genus 1 is a *torus*; the orientable surface with genus 2 is the *double torus*; and so on.
- The non-orientable surface with genus 1 is a *projective plane*; with genus 2 it is the *Klein bottle.*

Exercise 4.7. ☆☆ Identify the surfaces with the following schemata:

1. $a\bar{a}b\bar{b}$;
2. $abab$;
3. $aba\bar{b}$;
4. $a_1a_2\dots a_n\bar{a}_1\bar{a}_2\dots\bar{a}_n$;
5. $a_1a_2\dots a_{n-1}a_n\bar{a}_1\bar{a}_2\dots\bar{a}_{n-1}a_n$.

4.4 Surfaces with boundary

A *surface (possibly) with boundary* $\mathscr{S}$ is a topological space in which each point has a neighborhood homeomorphic to the unit open disk $\{(x,y) \in \mathbb{R}^2 \mid x^2+y^2 < 1\}$ or to the unit half disk $\{(x,y) \in \mathbb{R}^2 \mid x^2+y^2 < 1 \text{ and } x \geq 0\}$.

The *boundary* of $\mathscr{S}$, denoted by $\partial\mathscr{S}$, comprise the points of this surface that have no neighborhood homeomorphic to the unit disk. The *interior* of $\mathscr{S}$ is the complementary part of its boundary.

A *cellular embedding* on a surface with boundary is defined as in the case of surfaces without boundary. In particular, since each face must be an open disk, the boundary of the surface must be the union of some edges of the graph. The classification theorem (Theorem 4.5) can be extended for surfaces with boundary: Given a surface with boundary $\mathscr{S}$, we may attach a disk to each of its boundary components, obtaining a surface without boundary $\bar{\mathscr{S}}$, and apply the previous classification theorem. Furthermore, the number of boundary components is a topological invariant.

The Euler characteristic and the orientability character of a cellular embedding on a surface with boundary $\mathscr{S}$ are defined as in the case of surfaces without boundary; they are also topological invariants. The Euler characteristic of $\mathscr{S}$ equals that of $\bar{\mathscr{S}}$ minus the number of boundary components of $\mathscr{S}$. So two surfaces with boundary $\mathscr{S}$ and $\mathscr{S}'$ are homeomorphic if and only if they have the same Euler characteristic, orientability character, and number of boundary components.

If we have a graph embedding G without isolated vertex on a surface $\mathscr{S}$, then *cutting* $\mathscr{S}$ along G is a well-defined operation that yields a surface with boundary, denoted by $\mathscr{S}\backslash\!\backslash G$.[3] This fact is not trivial, and follows from the fact that every graph embedding on a surface $\mathscr{S}$ can be mapped by a homeomorphism of $\mathscr{S}$ (actually, an isotopy) to a piecewise-linear embedding with respect to a fixed triangulation of $\mathscr{S}$, using, e.g., the method by Epstein [17, Appendix].

4.5 Notes

The classification theorem is due to Brahana, Dehn, and Heegaard; the present proof is inspired from Stillwell [47]. For another, more visual proof, see Francis and Weeks [24].

The proofs of the classification theorem usually involve two steps, the first one being topological (Theorem 4.2, Proposition 4.3, Exercise 4.4), the second one being combinatorial. In the same vein, the *Hauptvermutung* ("main conjecture") says that any two embeddings of a graph on a surface are subdivisions of graph embeddings that are combinatorially identical. This is true, but some higher-dimensional analogs do not hold.

Let G and M be simple graphs (that is, without loops or multiple edges). M is a *minor* of G if M can be obtained from G by iteratively contracting edges, deleting edges, and deleting isolated vertices (at each step, the graph should be made simple by removing loops and identifying multiple edges). Let $\mathscr{S}$ be a fixed surface. Clearly, if G is embeddable on $\mathscr{S}$, then every minor of G is also embeddable on $\mathscr{S}$. Let $\mathcal{F}$ be the set of minor-minimal graphs *not* embeddable on $\mathscr{S}$; thus G is embeddable on $\mathscr{S}$ if and only if no graph in $\mathcal{F}$ is a minor of G. Kuratowski's theorem asserts that G is planar if and only if it does not have K_5 or $K_{3,3}$ as a minor; in other words, if $\mathscr{S}$ is the sphere, the family $\mathcal{F}$ is finite. This actually holds for every surface $\mathscr{S}$; however, no algorithm is known to enumerate the family $\mathcal{F}$.

More generally, this property is implied by a deep result by Robertson and Seymour [45] (whose proof needed no less than 20 papers and several hundreds of pages): In any infinite family of graphs, at least one is a minor of another.

[3] This notation is not standard (yet).

Chapter 5

Computing shortest graphs with cut loci

In this chapter, we describe algorithms to compute shortest curves and graphs that "cut" a given surface into simpler pieces.

5.1 Combinatorial and cross-metric surfaces

We aim at computing "short" graphs and curves on surfaces. For this, we need to define a metric on a surface that is both accurate in the applications and simple enough so as to be handled algorithmically. We shall introduce two ways of doing this, which are dual of each other. Depending on the context, some results and algorithms are more easily described using one setting or the other.

In this chapter, all surfaces are compact, connected, and orientable. They do not have boundaries.

5.1.1 More types of curves

We already defined paths on surfaces; we need to introduce more types of curves.

An *arc* on a surface with boundary $\mathscr{S}$ is a path $p : [0,1] \to \mathscr{S}$ such that $p(t)$ belongs to $\partial\mathscr{S}$ if and only if $t \in \{0,1\}$. A *loop* ℓ is a path with the same endpoints; $\ell(0) = \ell(1)$ is called the *basepoint* of the loop. A path is *simple* if it is one-to-one. A loop is *simple* if its restriction to $[0,1)$ is one-to-one (of course, due to the identified endpoints, it cannot be one-to-one on $[0,1]$).

The *concatenation* of p and q, denoted by $p \cdot q$, is the path defined by:

- $(p \cdot q)(t) = p(2t)$, if $0 \le t \le 1/2$;
- $(p \cdot q)(t) = q(2t-1)$, if $1/2 \le t \le 1$.

A *reparameterization* of a path p is a path of the form $p \circ \varphi$, where $\varphi : [0,1] \to [0,1]$ is bijective and increasing. If the paths are considered up to reparameterization, the concatenation is associative. The *inverse* of a path p, denoted by $\bar{p}$, is the map $t \mapsto p(1-t)$.

5.1.2 Combinatorial surfaces

A *combinatorial surface* $(\mathscr{S}, M)$ is the data of a surface $\mathscr{S}$ (possibly with boundary), together with a cellular embedding M of a weighted graph. The weights must be non-negative. In this model, the only allowed curves are walks in M; the length of a curve is the sum of the weights of the edges traversed by the curve, counted with multiplicity.

5.1.3 Cross-metric surfaces

We will, however, use a dual formulation of this model, which allows to define *crossings* between curves: this turns out to be helpful both for stating the results and as intermediate steps. A *cross-metric surface* $(\mathscr{S}, M^*)$ is a surface $\mathscr{S}$ together with a cellular embedding of a weighted graph M^*. If $\mathscr{S}$ has a boundary, we require in particular that each boundary of $\mathscr{S}$ be the union of some edges in M^*, with infinite crossing weight. We consider only *regular* paths on $\mathscr{S}$, which intersect the edges of M^* only transversely and away from the vertices. The *length* $\text{length}(\gamma)$ of a regular curve γ is defined to be the sum of the weights of the dual edges that γ *crosses*, counted with multiplicity. The length of a regular arc is defined similarly, excluding the endpoints of the arc (which belong to an edge of M^* with infinite crossing weight). To emphasize this usage, we sometimes refer to the weight of a dual edge as its *crossing weight*.

To any combinatorial surface $(\mathscr{S}, M)$ without boundary, we associate by duality a cross-metric surface $(\mathscr{S}, M^*)$, where M^* is (as notation suggests) the dual graph of M. To any curve on a combinatorial surface, traversing edges $e_1, \ldots, e_p$, we can associate a curve in the corresponding cross-metric surface, crossing edges $e_1^*, \ldots, e_p^*$, and conversely. This transformation preserves the lengths of the curves. So far, the notions of combinatorial and of cross-metric surfaces (without boundary) are thus essentially the same, up to duality. We can easily construct shortest paths on a cross-metric surface by restating the usual algorithms (for example, Dijkstra's algorithm) on M in terms of the dual graph M^*.

5.1.4 Curves on cross-metric surfaces, algorithmically

We can represent an arbitrary set of possibly (self-)intersecting curves on a cross-metric surface $(\mathscr{S}, M^*)$ by maintaining the *arrangement* of M^* and of the curves, i.e., the combinatorial embedding associated with the union of the curves (assuming this union forms a cellular embedding, which will always be the case). Contrary to combinatorial surfaces, this data structure also encodes the crossings between curves. The initial arrangement is just the graph M^*, without any additional curve. We embed each new curve *regularly*: every crossing point of the new curve and the existing arrangement, and every self-crossing of the new curve, creates a vertex of degree four.

Whenever we split an edge e^* of M^* to insert a new curve, we give both sub-edges the same crossing weight as e^*. Each segment of the curve between two intersection points becomes a new edge, which is, unless noted otherwise, assigned weight zero. However, it is sometimes desirable to assign a non-zero weight to the edges of a curve. For example, the cross-metric surface $\mathscr{S} \backslash\backslash \alpha$ obtained by cutting $\mathscr{S}$ along an embedded curve α can be represented simply by assigning infinite crossing weights to the edges that comprise α, indicating that these edges cannot be crossed by other curves.

5.1.5 Complexity

The *complexity* of a combinatorial surface $(\mathscr{S}, M)$ is the total number of vertices, edges, and faces of M; similarly, the *complexity* of a cross-metric surface $(\mathscr{S}, M^*)$ is the total number of vertices, edges, and faces of M^*. The *complexity* of a set of curves is the number of times it crosses edges of M^*.

5.2 Cut loci

Let us fix the notations for the remaining part of this chapter. Unless otherwise noted, $(\mathscr{S}, M^*)$ is a cross-metric surface (connected, compact, orientable, without boundary) of genus g and complexity n. Furthermore, b is a point inside a face of M^* and is the basepoint of all loops considered in this chapter (we omit the precision that the basepoint is b in the sequel).

Let T be the shortest path tree from b to a point in each face of M^*.[1] The *cut locus* C of $(\mathscr{S}, M^*)$ with respect to b is the subgraph of M^* obtained by removing all edges of M^* crossed by T. It is therefore a graph embedded on $\mathscr{S}$. See Figure 5.1.

Lemma 5.1. *$\mathscr{S} \backslash\backslash C$ is a disk.*

Proof. At some stage of the growth of the shortest path tree T, consider the union of all open faces of M^* visited by T, and of all edges of M^* crossed by T. This is an open disk; at the end, it contains all faces of M^*, and its complement is C. In particular, $\mathscr{S} \backslash\backslash C$ is a disk. □

Intuitively, we are inflating a disk around b progressively, without allowing self-intersections, until it occupies the whole surface; the cut locus C is the set of points of the surface where the boundary of the disk touches itself.

[1] Strictly speaking, the shortest path tree is not always unique: there may be several shortest paths between two given points. However, uniqueness holds for generic choices of the weights; in other words, it can be enforced using an arbitrarily small perturbation of the lengths. By a slight abuse of language, we will therefore use the article "the" in such cases, since it does not harm (and may help the reader) to think that uniqueness holds. Nevertheless, no algorithm or result presented here requires uniqueness of shortest paths.

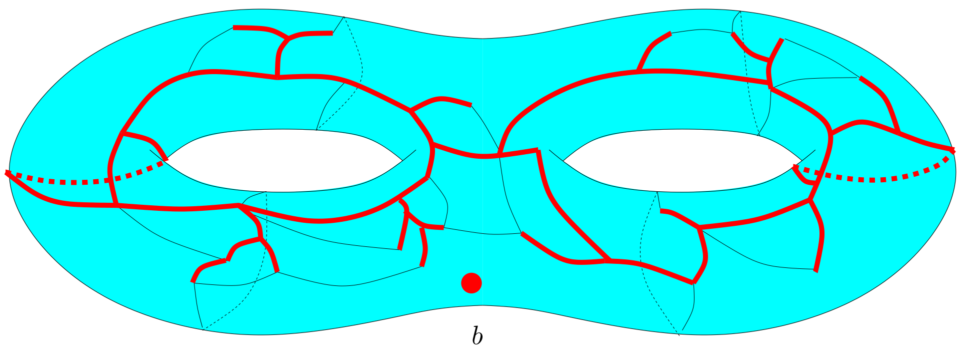

Figure 5.1. The cut locus C of a double torus (in bold lines) and the remaining edges of M^* (in thin lines).

Dijkstra's algorithm implies that we can compute C in $O(n \log n)$ time.

Exercise 5.2 (Complexity of the reduced cut locus)**.** ☆ Let C' be the graph obtained from the cut locus C' by repeatedly removing every degree-one vertex, together with its incident edge, and replacing every degree-two vertex v and its incident edges with an edge connecting the two neighbors of v. Prove that C' has complexity $O(g)$.

Given an edge $e \in C$, the loop $\sigma(e)$ is defined as a loop with basepoint b that follows the shortest path tree to go from its root b to a face incident with e, crosses e, and goes back from the other face incident with e to the root. This can be done so that all the loops $\sigma(e)$ are simple and disjoint (except, of course, at their basepoint b—we shall omit this triviality in the sequel). See Figure 5.2.

Define the *weight* of an edge e of C to be the length of the corresponding loop $\sigma(e)$ (this is not the same as the crossing weight, defined for every edge of M^*!); these weights can be computed with no time overhead during the cut locus computation.

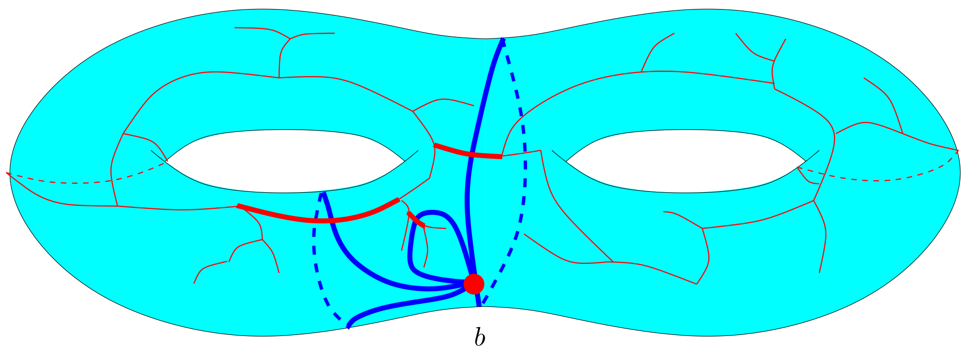

Figure 5.2. The loops $\sigma(e)$, for three edges $e \in C$.

5.3 Shortest non-contractible loop

A (possibly non-simple) loop is *contractible* if it can be continuously deformed into a point.

Exercise 5.3. ☆☆☆ Prove that, on a disk or a sphere, every loop is contractible.

Lemma 5.4. *A simple loop is contractible if and only if it bounds a disk.*

Proof. If a loop bounds a disk, it is certainly contractible. The proof of the converse is more difficult, and we admit it. □

Our goal now is to give an algorithm to compute the shortest non-contractible loop.

5.3.1 3-path condition

A set L of loops satisfies the *3-path condition* if, for any point $a \neq b$ and any three paths p, q, and r from b to a, if $p \cdot \bar{q}$ and $q \cdot \bar{r}$ belong to L, then $p \cdot \bar{r}$ belongs to L.

Lemma 5.5. *The set of contractible loops satisfies the 3-path condition.*

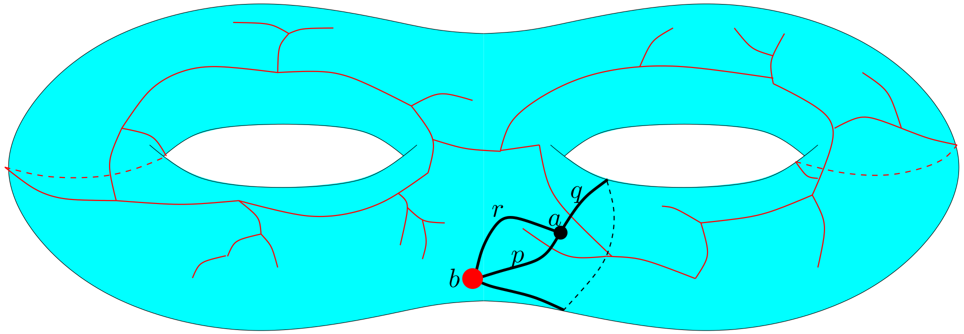

Figure 5.3. Illustration of Lemma 5.6.

Proof. If $p \cdot \bar{q}$ and $q \cdot \bar{r}$ are contractible, then so is their concatenation, $(p \cdot \bar{q}) \cdot (q \cdot \bar{r})$, which deforms continuously to $p \cdot \bar{r}$. □

Lemma 5.6. *Let L be a set of loops satisfying the 3-path condition. Some shortest loop not in L crosses the cut locus C at most once.*

Proof. See Figure 5.3 for an illustration of the proof. Let ℓ be a shortest loop not in L; without loss of generality, we can choose ℓ such that it crosses C as few times as possible. Assume, for the sake of a contradiction, that ℓ crosses C at least twice; let a be a point on ℓ not on M^* between its first and last crossing with C. This point a splits ℓ into two paths p and q, both from b to a, and we have $\ell = p \cdot \bar{q}$. Furthermore, let r be the shortest path from b to a; this path does not cross C.

The 3-path condition applied to p, q, and r implies that $p \cdot \bar{r}$ or $q \cdot \bar{r}$ does not belong to L. Both paths are no longer than $\ell = p \cdot \bar{q}$ and cross C fewer times than ℓ, implying the desired contradiction. □

5.3.2 Structural lemmas

Lemma 5.7. *Some shortest non-contractible loop has the form $\sigma(e)$.*

Proof. Let ℓ be a shortest non-contractible loop. By Lemmas 5.5 and 5.6, some shortest non-contractible loop crosses the cut locus at most once. On the other hand, every non-contractible loop has to cross C at least once (since $\mathscr{S} \setminus C$ is a disk). Hence some shortest non-contractible loop crosses the cut locus exactly once, at some edge e. This loop deforms continuously to $\sigma(e)$, which cannot be longer. The result follows. □

Lemma 5.8. *Let e be an edge of C. Then $\sigma(e)$ is contractible if and only if some component of $C - e$ is a tree.*

Proof. Assume first that one component of $C - e$ is a tree. One can them move $\sigma(e)$ continuously over the tree to make it disjoint from C; the resulting loop is contractible.

Conversely, if $\sigma(e)$ is contractible, it bounds a disk D by Lemma 5.4. We want to prove that the part of C inside D is a tree. But if it is not the case, this part contains a circuit, which further bounds a disk $D' \subset D$, and therefore C cuts $\mathscr{S}$ into at least two pieces, one of which is D'; this is impossible (Lemma 5.1). □

5.3.3 Algorithm

Theorem 5.9. *Finding a shortest non-contractible loop can be done in $O(n \log n)$ time. The loop computed is simple.*

Proof of Theorem 5.9. We first compute the cut locus C, and assign to every edge e of C a weight that is the length of $\sigma(e)$, in $O(n \log n)$ time. We show how to eliminate the edges e such that at least one component of $C - e$ is a tree. This concludes, since it then suffices to select the minimum-weight remaining edge of C (by Lemmas 5.7 and 5.8).

This graph pruning can be done in $O(n)$ time: put all edges incident with a degree-one vertex in a list. Then, while the list is non-empty, remove an edge e from it; remove it from C (unless it was already removed); if one or both of its endpoints have now degree one in C, put the corresponding edge(s) in the list. Clearly, this removes only edges e such that no component of $C - e$ is a tree. All them must eventually be removed, because a tree has a degree-one vertex (a leaf). □

Corollary 5.10. *Finding a shortest non-contractible loop without specified basepoint can be done in $O(n^2 \log n)$ time.*

Proof. For every face of M^*, run the algorithm in Theorem 5.9 with the basepoint in that face, and return the shortest loop. □

5.4 Shortest non-separating loop

5.4.1 Types of simple loops

A simple loop ℓ is *separating* if $\mathscr{S}\setminus\!\!\setminus\ell$ is not connected. A simple contractible loop bounds a disk, hence is separating; the converse is false. So there are (essentially) three kinds of simple loops: contractible, separating but not contractible, and non-separating. These three types are illustrated in Figure 5.2.

Exercise 5.11. ☆

1. Give an algorithm that determines whether a given simple loop is separating.
2. Give an algorithm that determines whether a given simple loop is contractible. Indication: use Lemma 5.4.

Our present goal is to compute the shortest non-separating (simple) loop. We need first to define the notion of homology boundary, which generalizes the notion of separating loop to possibly non-simple loops. To anticipate, we introduce a bit more technicalities than those needed for this sole purpose.

5.4.2 Preliminaries on homology

We introduce *1-dimensional homology for graphs embedded on surfaces, over* $\mathbb{Z}/2\mathbb{Z}$.

To simplify matters, we assume here (and in Section 5.5) that all curves considered are drawn on a very dense graph $G = (V, E)$ embedded on $\mathscr{S}$, transversely to M^*.[2] We consider *chains*: subsets of E. It is a natural $\mathbb{Z}/2\mathbb{Z}$*-vector space*: the addition of two subsets of E is the symmetric difference, multiplication by 0 gives the empty subset of E, and multiplication by 1 is the identity.

A chain $E' \subseteq E$ is a *homology cycle* if every vertex of V is incident with an even number of edges of E'. A chain $E' \subseteq E$ is a *homology boundary* if the faces of G can be colored black and white so that E' is the set of edges of E with exactly one black and one white incident face. Equivalently, if we consider the "dual" graph of (V, E'), which has one vertex inside each face of (V, E') and one edge crossing each edge of (V, E'), then E' is a homology boundary if and only if this dual graph is bipartite.

Exercise 5.12. ☆☆☆

1. Prove that the set of homology cycles (resp. homology boundaries) forms a vector space, and that every homology boundary is a homology cycle.
2. Assume $\mathscr{S}$ is a sphere. Prove that every homology cycle is a homology boundary.

Lemma 5.13. *A simple loop ℓ in G disconnects $\mathscr{S}$ if and only if its edge set forms a homology boundary.*

Proof. Let E' be the set of edges of ℓ. Either the graph (V, E') has one face, in which case the only boundary is the empty set, or it has two faces, in which case coloring one face in black and the other one in white yields a non-zero boundary formed by the edge set of ℓ. □

So the notion of homology boundary extends the notion of being separating.

As shown in Exercise 5.12, the set of homology boundaries, B, is included in the set of homology cycles, Z. The reverse inclusion does not hold in general. Homology measures the "difference" between Z and B; formally, it

[2] This would not be needed if we introduced *singular homology*, but it seems preferable to avoid doing so. The assumption above is actually not needed: we only require G to be dense enough so that the loops $\sigma(e)$ are disjoint walks on G and so that G contains some shortest non-separating loop (or some shortest system of loops, in Section 5.5). The existence of such a graph G is clear, and it is never used in the algorithms, only in proofs, so its complexity does not matter.

is Z/B, the $\mathbb{Z}/2\mathbb{Z}$-vector space that is the quotient of the two $\mathbb{Z}/2\mathbb{Z}$-vector spaces Z and B.

Given a loop ℓ in G, its *mod 2 image* is the set of edges of G used an odd number of times by ℓ. (We sometimes identify a loop with its mod 2 image.)

5.4.3 Algorithm

We prove here:

Theorem 5.14. *Finding a shortest loop that is not a homology boundary can be done in $O(n \log n)$ time. The loop computed is simple, and is (therefore) also a shortest simple non-separating loop.*

Corollary 5.15. *Finding a shortest loop without specified basepoint that is not a homology boundary (or a shortest simple non-separating closed curve) can be done in $O(n^2 \log n)$ time.*

Lemma 5.16. *A subset A of the edges of C disconnects C if and only if the set of loops $\sigma(A)$ disconnects $\mathscr{S}$.*

Proof. We may certainly assume $A \neq \emptyset$. Let D be the disk $\mathscr{S} \backslash\backslash C$; the basepoint b belongs to the interior of D. Each loop $\sigma(e)$ in $\sigma(A)$ corresponds, in D, to two paths from b to the boundary of D, connecting twins of e. See Figure 5.4.

Therefore, if we let $\tau(e)$ be the intersection of e with $\sigma(e)$, any path in $\mathscr{S} \setminus \sigma(A)$ continuously retracts to a path in $C \setminus \tau(A)$, without moving the endpoints if they already belong to C. This implies that $\mathscr{S} \setminus \sigma(A)$ is connected if and only if $C \setminus \tau(A)$ is connected; this is in turn equivalent to having $C - A$ connected. □

Proof of Theorem 5.14. The general strategy is very similar to the proof of Theorem 5.9. The set of all loops in G whose mod 2 images are homology boundaries satisfies the 3-path condition. Hence, by Lemma 5.6, some shortest loop in G whose mod 2 image is *not* a homology boundary crosses the cut locus at most once, hence exactly once, at some edge e, by

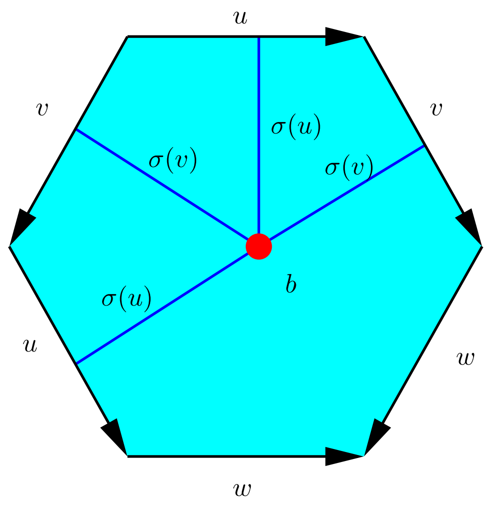

Figure 5.4. A view of the disk $\mathscr{S} \backslash\backslash C$, whose polygonal schema is $uvw\bar{w}\bar{u}\bar{v}$. The loops $\sigma(u)$ and $\sigma(v)$ are cut into two paths connecting the basepoint to twin points.

Exercise 5.12. A slight extension of that exercise implies that $\sigma(e)$ is in the same homology class, and it is no longer. Hence some shortest loop whose mod 2 image is not a homology boundary has the form $\sigma(e)$.

In particular, it is simple, and is therefore a non-separating loop (Lemma 5.13). It must be a shortest non-separating loop in G because every separating loop is a homology boundary. It is therefore a shortest non-separating loop, because we can (retroactively) assume that G contains some shortest non-separating loop.

By Lemma 5.16, we are thus looking for a minimum-weight edge e of C such that $C - e$ is connected; such edges are called *non-bridge* edges. Recall from Lemma 2.2 that we can determine all non-bridge edges in linear time. Alternately, note that any minimum-weight edge not in a maximum spanning tree of C is such an edge. □

5.5 Shortest system of loops

In this section, we describe an algorithm to compute a shortest topological decomposition of the surface. Namely, a *system of loops* L is a set of simple loops meeting pairwise only at their common basepoint b, such that $\mathscr{S} \setminus\!\!\setminus L$ is a disk (refer to Figure 4.6(c) for an example). We give an algorithm to compute the shortest system of loops of a given surface.

5.5.1 Algorithm

Define a *homology basis of loops* to be a set of loops whose homology classes (of their mod 2 images) form a basis of the homology vector space. There exist homology bases of loops:

Exercise 5.17. ☆☆☆ Prove that every homology cycle is the mod 2 image of a loop.

Recall that a *system of loops* L is a set of simple loops meeting pairwise only at their common basepoint, such that $\mathscr{S} \setminus\!\!\setminus L$ is a disk. Denote by $[\ell]$ the homology class of a loop ℓ, and by $[L]$ the set of homology classes of a set of loops L.

Lemma 5.18. *Some shortest homology basis is made of loops of the form $\sigma(e)$. In particular, the loops in that basis are simple and disjoint.*

Proof. Let ℓ be a loop in the shortest homology basis. Let $e_1, \ldots, e_k$ be the edges of the cut locus crossed by ℓ. Then it is not too hard, using Exercise 5.12(2) , to prove that $[\ell] = [\sigma(e_1)] + \ldots + [\sigma(e_k)]$.

In particular, ℓ crosses at least one edge of the cut locus. Furthermore, since $[\ell]$ is linearly independent from the homology classes of the other loops in the basis, one of the $[\sigma(e_i)]$ must be linearly independent from the homology classes of the other loops in the basis. Replacing ℓ with $\sigma(e_i)$ still yields a homology basis, which is no longer than the original one because $\sigma(e_i)$ is a shortest loop with basepoint b among the loops that cross e_i, and ℓ indeed crosses e_i. Iterating, we obtain that some shortest homology basis is made of loops of the form $\sigma(e)$. □

Exercise 5.19. ☆☆☆ Let L be a set of simple, disjoint loops in G. Prove that L disconnects $\mathscr{S}$ if and only if the homology classes of the loops in L are linearly dependent.

Theorem 5.20. *We can compute a a shortest homology basis of loops in $O(gn + n \log n)$ time. Furthermore, there are $2g$ loops, each of the form $\sigma(e)$.*

Proof. By Lemma 5.18, computing a shortest homology basis of loops boils down to computing a shortest inclusionwise maximal set of loops $\sigma(e_1), \ldots, \sigma(e_k)$ with linearly independent homology classes, or, equivalently, that does not disconnect $\mathscr{S}$ (Exercise 5.19). This is equivalent to computing an inclusionwise maximal set S of edges of C such that $C - S$ is connected, with minimal sum of weights (Lemma 5.16). This precisely means computing the complement of a maximum-weight spanning tree of C.

Recall that C is cellularly embedded on $\mathscr{S}$ with one face (Lemma 5.1). Therefore, by Euler's formula, the number of vertices, v, and edges, e, of C satisfy $v - e = 2 - 2g - 1 = 1 - 2g$. A spanning tree always contains $v - 1$ edges (Lemma 1.7), so the complement of a spanning tree of C has exactly $2g$ edges; we conclude that there are $2g$ loops in L.

Computing the cut locus C takes $O(n \log n)$ time. A maximum spanning tree can be computed in $O(n \log n)$ time using any textbook algorithm. The actual loops may each have $O(n)$ size, and there are $2g$ of these. □

Proposition 5.21. *The shortest homology basis of loops L computed in Theorem 5.20 is actually a shortest system of loops.*

Proof. Every system of loops is made of $2g$ loops by Euler's formula. The homology classes of a system of loops are linearly independent (Exercise 5.19), and there are $2g$ of these, so they form a basis. So any system of loops is a homology basis. It therefore suffices to prove that L is a system of loops.

L is a set of $2g$ simple, disjoint loops that does not disconnect $\mathscr{S}$. Cutting along it yields a (connected) surface of Euler characteristic 1 (because cutting along the first loop keeps the Euler characteristic unchanged and cutting along each subsequent loop increases it by one), hence a disk. □

5.6 Extensions

5.6.1 Shortest loops on surfaces with boundary

Let $(\mathscr{S}, M^*)$ be a cross-metric surface with genus g and $b \geq 1$ boundary components, and complexity n. We briefly indicate how the results of Sections 5.3 and 5.4 generalize to surfaces with boundary.

Let $\hat{\mathscr{S}}$ be the surface $\mathscr{S}$ where a handle is attached to each boundary component. Thus $\hat{\mathscr{S}}$ is a surface without boundary. The graph M^* is not cellularly embedded in $\mathscr{S}$, but we can make it cellular by adding two edges per attached handle. We assign infinite weights to these new edges. Let $\hat{M}^*$ be the resulting graph.

A loop in $\mathscr{S}$ is contractible if and only if it is contractible in $\hat{\mathscr{S}}$. Therefore, to compute the shortest non-contractible loop in $(\mathscr{S}, M^*)$ with a given basepoint, it suffices to compute the shortest non-contractible loop in $(\hat{\mathscr{S}}, \hat{M}^*)$ with that basepoint. Similarly, a loop in the interior of $\mathscr{S}$ is separating if and only if it is separating in $\hat{\mathscr{S}}$. So, to compute the shortest non-separating loop in $(\mathscr{S}, M^*)$, it suffices to compute the shortest non-separating loop in $(\hat{\mathscr{S}}, \hat{M}^*)$.[3]

There is no direct analog of the notion of system of loops for surfaces with boundary, because no set of disjoint simple loops in the interior of the surface can cut it into a disk. We shall see a replacement of this notion in Section 5.6.3.

5.6.2 Shortest paths relatively to a set of points

We now extend the result of Section 5.5 to the case where there are "more than one basepoint". Specifically, let $(\mathscr{S}, M^*)$ be a cross-metric surface without boundary, with a finite set P of k points, each in a different face of M^*. Let us call a *P-path* any path with endpoints in P. A *system of P-paths* is a set of simple P-paths that are pairwise disjoint, except possibly at their endpoints, cutting $\mathscr{S}$ into a topological disk, and meeting every point in P; equivalently, it is a graph embedded on $\mathscr{S}$ with vertex set *exactly* P whose removal leaves a disk. Our goal is to compute a shortest system of P-paths.

[3] Here, instead of attaching a handle to every boundary component of $\mathscr{S}$, attaching a disk would also work.

Let F be a shortest path forest in M^*, growing simultaneously from each point of P, and connecting every face of M^*. The *cut locus C* of $(\mathscr{S}, M^*)$ with respect to P is defined as the subgraph of M^* obtained by removing all edges of M^* crossed by F. It cuts $\mathscr{S}$ into k topological disks, each containing exactly one point in P. Given an edge e of the cut locus, the arc $\sigma(e)$ is the concatenation of two shortest paths following the shortest path forest starting on both sides of e until each of them reaches a root.

The *3-path condition* generalizes as follows to P-paths: a set S of P-paths satisfies this condition if and only if, for every point a and any three paths p, q, and r from points in P to a, if $p \cdot \bar{q}$ and $q \cdot \bar{r}$ belong to S, then so does $p \cdot \bar{r}$.

Homology is defined almost as in Section 5.4.2: we assume $G = (V, E)$ is a very dense graph with $P \subseteq V$. A chain $E' \subseteq E$ is a *homology cycle* if every vertex *not in* P is incident with an even number of edges of E'. A chain $E' \subseteq E$ is a *homology boundary* if, as usual, the faces of (V, E') can be colored in black and white such that every edge of E' is incident with one black and one white face. A *homology basis of P-paths* is a set of P-paths whose homology classes form a basis of the homology vector space.

Then as in Section 5.4.2, we have:

Theorem 5.22. *Let $(\mathscr{S}, M^*)$ be a cross-metric surface with genus g and without boundary; let n be its complexity. Let P be a set of k points on $\mathscr{S}$. Finding a shortest homology basis of P-paths can be done in $O(n \log n + (g+k)n)$ time. The basis computed is actually a shortest system of P-paths. The number of P-paths in every system of P-paths, and every homology basis of P-paths, is $2g + k - 1$.*

5.6.3 Shortest arcs on a surface with boundary

We come back to the case of surfaces with boundary: Assume $(\mathscr{S}, M^*)$ has $b \geq 1$ boundary components. A *system of arcs* on $\mathscr{S}$ is a set of disjoint simple arcs cutting $\mathscr{S}$ into a disk (this is an appropriate analog of a system of loops for surfaces with boundary).

Let $\bar{\mathscr{S}}$ be the cross-metric surface without boundary obtained by attaching disks to each boundary component of $\mathscr{S}$. Let $\bar{M}^*$ be the graph M^* where all edges on the boundary of $\mathscr{S}$ are assigned a fixed large enough crossing weight W; thus $(\bar{\mathscr{S}}, \bar{M}^*)$ is a cross-metric surface without boundary. Furthermore, let P be a set of points, one inside every disk glued to the boundaries of $\mathscr{S}$.

Call a P-path on $\bar{\mathscr{S}}$ *admissible* if it intersects the boundary of $\mathscr{S}$ in exactly two points. Admissible P-paths in $\bar{\mathscr{S}}$ precisely correspond to arcs in $\mathscr{S}$; this correspondence preserves the lengths, up to a shift of $2W$. Furthermore, a system of admissible P-paths in $\bar{\mathscr{S}}$ corresponds to a system of arcs in $\mathscr{S}$. Since the algorithm of Theorem 5.22 only computes admissible P-paths (provided W is chosen large enough), the above considerations yield:

Theorem 5.23. *Let $(\mathscr{S}, M^*)$ be a cross-metric surface with genus g and $b \geq 1$ boundary components, and complexity n. Finding a system of arcs can be done in $O(n \log n + (g+b)n)$ time. Every system of arcs is made of $2g + b - 1$ arcs.*

5.7 Notes

5.7.1 Discrete vs. continuous setting

Most of the combinatorial and cross-metric surface model is taken from Colin de Verdière and Erickson [11]. Several tools of this section were described in a combinatorial setting for simplicity of exposition, but they have well-studied continuous counterparts.

In general, the cut locus of a point x in a metric space S is the set of points in S for which there exist at least two distinct shortest paths to x. It is closely related to the notion of the *medial axis* of a compact set $K \subset S$: it is the set of points of $S \setminus K$ whose distance to K is realized by at least two points of K. If K is finite, the medial axis contains in particular the Voronoi diagram of K.

The main topological property of a cut locus we have used (in Lemmas 5.8 and 5.16) can be stated as follows for a surface with boundary: for any subset A of the edges of C, $\mathscr{S} \setminus \sigma(A)$ deformation retracts to $C - A$. In particular, they have the same number of connected components, and one of the components of $\mathscr{S} \setminus \sigma(A)$ is a disk if and only if the corresponding component of $C - A$ (is connected and) contains no non-contractible loop, i.e., is a tree.

As mentioned earlier, homology can be defined in a continuous setting (*singular homology*), which vastly generalizes the ad-hoc route we took. Let S be any topological space. Let Δ_n be the n-dimensional simplex. The set of *n-chains* C_n is the vector space (say over $\mathbb{Z}/2\mathbb{Z}$, but this generalizes to arbitrary fields, and even rings) generated by all continuous maps $\Delta_n \to S$. There is a *boundary operator* ∂_n taking C_n to C_{n-1}: the boundary of $\Delta_n \to S$ is a sum of $n+1$ maps $\Delta_{n-1} \to S$, one for each face of Δ_n. One checks the important property that $\partial_{n-1} \circ \partial_n = 0$, so $\operatorname{Im} \partial_n \subseteq \operatorname{Ker} \partial_{n-1}$. The set of *homology cycles* is $Z_n := \operatorname{Ker} \partial_{n-1}$ and the set of *homology boundaries* is $B_n := \operatorname{Im} \partial_n$. These vector spaces have infinite dimension (except in trivial cases), but their quotient $H_n := Z_n/B_n$, the *homology vector space*, is usually of finite dimension; it is non-trivial to prove that, under reasonable conditions, H_1 is isomorphic to the homology vector space as introduced in Section 5.4.2.

The appropriate machinery for the generalization to the shortest system of B-paths (and arcs) is *relative homology*. See any textbook on algebraic topology for more details on homology [28, 29].

5.7.2 Algorithms

Erickson and Har-Peled [20] gave the first algorithms to compute the shortest non-contractible or non-separating loop, relying on the idea of "wavefront propagation". The method presented here is different; the idea of considering the edges of the cut locus is borrowed from Erickson and Whittlesey [21]. The 3-path condition is a variation on Mohar and Thomassen [41, p. 110].

If the genus is small, then our $O(n^2 \log n)$ algorithm is not very efficient; after successive improvements [5, 35], the best algorithm up to date has running-time $O(g^3 n \log n)$ [4]. In contrast, computing the shortest separating but non-contractible simple loop (without specified basepoint) is NP-hard [7].

Erickson and Whittlesey [21] described the algorithm of Section 5.5; the algorithm was further generalized, and the proof was simplified, by Colin de Verdière [10], which was in turn simplified by Erickson [19].

Note that there are systems of loops whose polygonal schema is not in canonical form (for example $abcd\bar{a}\bar{b}\bar{c}\bar{d}$). The shortest system of loops is not necessarily in canonical form. There is an $O(gn)$ time algorithm to compute a system of loops in canonical form [36, 50], but computing the shortest such system is likely to be NP-hard. There are other kinds of topological decompositions of surfaces, such as *pants decompositions*: sets of disjoint simple closed curves that cut the surface into spheres with three boundary components. The status of computing the shortest pants decomposition is open [42].

Bibliography

[1] Kenneth Appel and Wolfgang Haken. *Every planar map is four-colorable.* AMS, Providence, Rhode Island, 1989. [p. 16]

[2] Mark Anthony Armstrong. *Basic topology.* Undergraduate Texts in Mathematics. Springer-Verlag, 1983. [pp. 3 and 7]

[3] Erik Brisson. Representing geometric structures in d dimensions: topology and order. *Discrete & Computational Geometry*, 9:387–426, 1993. [p. 8]

[4] Sergio Cabello and Erin W. Chambers. Multiple source shortest paths in a genus g graph. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 89–97, 2007. [p. 37]

[5] Sergio Cabello and Bojan Mohar. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. *Discrete & Computational Geometry*, 37(2):213–235, 2007. [p. 37]

[6] Luca Castelli-Aleardi, Olivier Devillers, and Éric Fusy. Canonical ordering for triangulations on the cylinder, with applications to periodic straight-line drawings. In *Proceedings of the 21st International Symposium on Graph Drawing (GD)*, 2012. To appear. [p. 14]

[7] Erin W. Chambers, Éric Colin de Verdière, Jeff Erickson, Francis Lazarus, and Kim Whittlesey. Splitting (complicated) surfaces is hard. *Computational Geometry: Theory and Applications*, 41(1–2):94–110, 2008. [p. 37]

[8] Bernard Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000. [p. 22]

[9] David R. Cheriton and Robert Endre Tarjan. Finding minimum spanning trees. *SIAM Journal on Computing*, 5(4):724–742, 1976. [p. 22]

[10] Éric Colin de Verdière. Shortest cut graph of a surface with prescribed vertex set. In *Proceedings of the 18th European Symposium on Algorithms (ESA), part 2*, number 6347 in Lecture Notes in Computer Science, pages 100–111, 2010. [p. 37]

[11] Éric Colin de Verdière and Jeff Erickson. Tightening nonsimple paths and cycles on surfaces. *SIAM Journal on Computing*, 39(8):3784–3813, 2010. [p. 37]

[12] Éric Colin de Verdière and Alexander Schrijver. Shortest vertex-disjoint two-face paths in planar graphs. *ACM Transactions on Algorithms*, 7(2):Article 19, 2011. [p. 22]

[13] Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990. [p. 14]

[14] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph drawing*. Prentice Hall, Upper Saddle River, NJ, 1999. [p. 14]

[15] P. H. Doyle and D. A. Moran. A short proof that compact 2-manifolds can be triangulated. *Inventiones Mathematicae*, 5:160–162, 1968. [p. 24]

[16] David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 599–608, 2003. [p. 8]

[17] David B. A. Epstein. Curves on 2-manifolds and isotopies. *Acta Mathematica*, 115:83–107, 1966. [pp. 24 and 28]

[18] Jeff Erickson. Computational topology, 2009. Course notes available at `http://compgeom.cs.uiuc.edu/~jeffe/teaching/comptop/`. [p. 3]

[19] Jeff Erickson. Combinatorial optimization of cycles and bases. In Afra Zomorodian, editor, *Computational topology*, Proceedings of Symposia in Applied Mathematics. AMS, 2012. [p. 37]

[20] Jeff Erickson and Sariel Har-Peled. Optimally cutting a surface into a disk. *Discrete & Computational Geometry*, 31(1):37–59, 2004. [p. 37]

[21] Jeff Erickson and Kim Whittlesey. Greedy optimal homotopy and homology generators. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1038–1046, 2005. [p. 37]

[22] István Fáry. On straight line representations of planar graphs. *Acta scientiarum mathematicarum (Szeged)*, 11:229–233, 1948. [p. 14]

[23] Stefan Felsner. Convex drawings of planar graphs and the order dimension of 3-polytopes. *Order*, 18(1):19–37, 2001. [p. 14]

[24] George K. Francis and Jeffrey R. Weeks. Conway's ZIP proof. *American Mathematical Monthly*, 106(5):393–399, 1999. [p. 28]

[25] Philip Franklin. The four-color problem. *American Journal of Mathematics*, 44(3):225–236, 1922. [p. 22]

[26] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987. [p. 22]

[27] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976. [p. 16]

[28] Allen Hatcher. *Algebraic topology*. Cambridge University Press, 2002. Available at `http://www.math.cornell.edu/~hatcher/`. [p. 37]

[29] Michael Henle. *A combinatorial introduction to topology*. Dover Publications, 1994. [pp. 7 and 37]

[30] John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974. [p. 9]

[31] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for Min Cut and Max Flow in undirected planar graphs. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 313–322, 2011. [p. 22]

[32] David R. Karger, Philip N. Klein, and Robert E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995. [p. 22]

[33] Lutz Kettner. Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry: Theory and Applications*, 13:65–90, 1999. [p. 8]

[34] Casimir Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930. [p. 8]

[35] Martin Kutz. Computing shortest non-trivial cycles on orientable surfaces of bounded genus in almost linear time. In *Proceedings of the 22nd Annual Symposium on Computational Geometry (SOCG)*, pages 430–438. ACM, 2006. [p. 37]

[36] Francis Lazarus, Michel Pocchiola, Gert Vegter, and Anne Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *Proceedings of the 17th Annual Symposium on Computational Geometry (SOCG)*, pages 80–89. ACM, 2001. [p. 38]

[37] Bruno Lévy. *Topologie algorithmique: combinatoire et plongement*. PhD thesis, Institut National Polytechnique de Lorraine, 1999. [p. 8]

[38] Pascal Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry & Applications*, 4(3):275–324, 1994. [p. 8]

[39] Yuri Makarychev. A short proof of Kuratowski's graph planarity criterion. *Journal of Graph Theory*, 25:129–131, 1997. [p. 8]

[40] Tomomi Matsui. The minimum spanning tree problem on a planar graph. *Discrete Applied Mathematics*, 58(1):91–94, 1995. [p. 22]

[41] Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2001. [pp. 3, 7, 15, and 37]

[42] Sheung-Hung Poon and Shripad Thite. Pants decomposition of the punctured plane. In *Proceedings of the 22nd European Workshop on Computational Geometry (EWCG)*, pages 99–102, 2006. [p. 38]

[43] John H. Reif. Minimum $s-t$ cut of a planar undirected network in $O(n \log^2(n))$ time. *SIAM Journal on Computing*, 12(1):71–81, 1983. [p. 22]

[44] Neil Robertson, Daniel P. Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, 1996. [p. 22]

[45] Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92:325–357, 2004. [p. 28]

[46] S. K. Stein. Convex maps. *Proceedings of the AMS*, 2:464–466, 1951. [p. 14]

[47] John Stillwell. *Classical topology and combinatorial group theory*. Springer-Verlag, New York, second edition, 1993. [pp. 3, 7, and 28]

[48] Carsten Thomassen. Kuratowski's theorem. *Journal of Graph Theory*, 5(3):225–241, 1981. [p. 8]

[49] Carsten Thomassen. The Jordan-Schönflies theorem and the classification of surfaces. *American Mathematical Monthly*, 99(2):116–130, 1992. [pp. 5 and 24]

[50] Gert Vegter and Chee K. Yap. Computational complexity of combinatorial surfaces. In *Proceedings of the 6th Annual Symposium on Computational Geometry (SOCG)*, pages 102–111. ACM, 1990. [p. 38]

[51] K. Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936. [p. 14]

Contents