COMBINATORIAL OPTIMIZATION ON EMBEDDED CURVES

BY

AMIR NAYYERI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Professor Jeff Erickson, Chair
Professor David Forsyth
Associate Professor Sariel Har-Peled
Professor Tamal Dey, The Ohio State University

Abstract

We describe several algorithms for classifying, comparing and optimizing curves on surfaces. We give algorithms to compute the minimum member of a given homology class, particularly computing the maximum flow and minimum cuts, in surface embedded graphs. We describe approximation algorithms to compute certain similarity measures for embedded curves on a surface. Finally, we present algorithms to solve computational problems for compactly presented curves.

We describe the first algorithms to compute the shortest representative of a $\mathbb{Z}_2$-homology class. Given a directed graph embedded on a surface of genus g with b boundary cycles, we can compute the shortest single cycle $\mathbb{Z}_2$-homologous to a given even subgraph in $2^{O(g+b)}n \log n$ time. As a consequence we obtain an algorithm to compute the shortest directed non-separating cycle in $2^{O(g)}n \log n$ time, which improves the previous best algorithm by a factor of $O(\sqrt{n})$ if the genus is a constant. Further, we can compute the shortest even subgraph in a given $\mathbb{Z}_2$-homology class if the input graph is undirected in the same asymptotic running time. As a consequence, we obtain the first near linear time algorithm to compute minimum (s,t)-cuts in surface embedded graphs of constant genus. We also prove that computing the shortest even subgraph in a $\mathbb{Z}_2$-homology class is in general NP-hard, which explains the exponential dependence on g.

We also consider the corresponding optimization problem under $\mathbb{Z}$-homology. Given an integer circulation Φ in a directed graph embedded on a surface of genus g, we describe algorithms to compute the minimum cost circulation that is $\mathbb{Z}$-homologous to Φ in $O(g^8 n \log^2 n \log^2 C)$ time if the capacities are integers whose sum is C or in $g^{O(g)}n^{3/2}$ time for arbitrary capacities. In particular, our algorithm improves the best known algorithm for computing the maximum (s,t)-flow on surface embedded graph after 20 years. The previous best algorithm, except for planar graphs, follow from general maximum flow algorithms for sparse graphs.

Next, we consider two closely related similarity measures of curves on piecewise linear surfaces embedded in $\mathbb{R}^3$, called homotopy height and homotopic Frechét distance. These similarity measures capture the longest curve that appears and the longest length that any point travels in the best morph between two given curves, respectively. We describe the first polynomial-time $O(\log n)$-approximation algorithms for both problems. Prior to our work no algorithms were known for the homotopy height problem. For the homotopic Frechét distance, algorithms were known only

for curves on Euclidean plane with polygonal obstacles. Surprisingly, it is not even known if deciding if either the homotopy height or the homotopic Frechét distance is smaller that a given value is in NP.

Finally, we consider normal curves on abstract triangulated surfaces. A curve is normal if it intersects any triangle in a finite set of arcs, each crossing between two different edges of the triangle. Given a triangulated surface of complexity n and a curve that crosses the triangulation X times, we can build another cell decomposition of the input surface of complexity $O(n)$, in $O(\min(X, n^2 \log X))$ time, whose 1-skeleton contains the input curve. We emphasize the the cell decomposition algorithm takes polynomial time even if X is exponential. The main ingredient of our cell decomposing algorithm is a technique to trace a curve in a triangulated surface. We apply our abstract tracing strategy to solve well-known problems about normal curves including computing the number of components, computing the number of isotopy classes and computing the algebraic intersection number between two curves. Our normal-coordinate algorithms are competitive with and conceptually simpler than earlier algorithms.

To Sorayya, Hossein, Ghazal and Parisa

Acknowledgements

First, I would offer my sincerest gratitude to my advisor, Jeff Erickson, for the true research spirit that he showed me with patience. Without his guidance, his constant support and his confidence in my research abilities I would never be able to finish this journey. I have been extremely lucky to have the chance to work under his supervision, I owe most of my abilities as a researcher to his method of advising.

The research presented in this thesis is a result of collaboration with Erin Chambers, Jeff Erickson, Sariel Har-peled, Mohammad Salavatipour and Anastasios Sidiropoulos. My other collaborators are Kyle Fox and Yury Makarychev. Working with such smart and energetic people was a unique chance that I have had in my professional life. I would like to specially thank my former adviser, Tarek Abdelzaher, for all I have learned from him, and for his understanding and support when I decided to switch my research area.

My research has been supported by NSF grant CCF 09-15519. Also, part of the research has been done while I was visiting Toyota Technological Institute in Chicago. I would like to thank both.

Thanks to the fellows in theory group, students and faculty, for all supports that I received in every single steps, for all helpful discussions and for being so friendly. I would like to thank Chandra Chekuri, Alina Ene, Nitish Koula, Nirman Kumar, Sungjin Im, Hemanta Maji, Benjamin Moseley, Manoj Prabhakaran, Benjamin Raichel, and Daniel Schreiber. I also would like to express my gratitude toward professors who accepted to serve as my thesis committee members, Tamal Dey, David Forsyth and Sariel Har-peled as well as Leo Guibas.

Thanks to all my friends that supported me from the first moment I entered United States, in particular, Ali Farhadi for giving me the first ride in this country and for being such a special friend. Thanks to all people that played soccer with me; without playing soccer I would never be able to solve a research problem. I would like to thank my best roommate ever, Behrouz Touri, for all he taught me and for being so patient for three years! I have learned a lot from Hossein Ahmadi and Reza Zamani, both academic and non-academic.

I wish to thank my mother, Sorayya, for being so devoted, dedicated and concerned about my future, my father, Hossein, for being extremely patient and supportive and my sister, Ghazal, for being so kind and energetic. Without their sincere support I would not be able to make this journey.

Last but certainly not the least, Parisa has been the greatest source of inspiration and support in the last couple of years. She has proven to me that life is a beautiful journey and has given me love and energy to continue.

Table of Contents

Chapter 1

Introduction

Curves have been the subject of study in many different branches of mathematics, particularly in geometry and topology, because of their own interesting nature and as a way toward understanding their containing spaces. Computational problems about curves show up in different branches of computer science including computer graphics, computer vision, machine learning, and biological computing

The curves in any space can be classified by equivalence relations. Perhaps the most natural equivalence relation is homotopy, where two curves are equivalent (homotopic) if one can be continuously deformed to the other within the space. Homotopy classes compose the set of the fundamental group of the underlying space, which is extensively used to study and classify spaces. In particular, the isomorphism of the fundamental groups of two spaces is a necessary (but not sufficient) condition for their homeomorphism [108], and this conditions leads to simple proofs of many important theorems, including the fundamental theorem of algebra, the Brouwer fixed point theorem, the Borsuk-Ulam theorem and the existence of non-trivial knots [108]. Further, the fact that any group is isomorphic to the fundamental group of some topological space [108] enlightens an intimate relation between the study of embedded curves and abstract algebra.

Another relation that Poincaré [153] defined between curves is homology. Roughly, two families of cycles are homologous if one can be transformed into the other, using a continuous transformation that allows cycles to split and rejoin at intersection points. Poincaré studied homology with the immediate objective of generalizing a duality observed by Betti and finding a completely general version of Euler's formula [180]. The first order homology group of a surface is, in fact, the abelianization of the fundamental group. Thus, homology as a relation is coarser than homotopy, meaning that any pair of homotopic curves are also homologous but not the other way around. However, homology groups still provide an accurate classification of surfaces with interesting mathematical implications such as Brouwer fixed point theorem and the invariance of dimension [108]). On the other hand, because homology groups are abelian, homology spaces are essentially vector spaces, which makes them much easier to deal with. Since homology groups provide valuable information about the underlying space and it is easier to compute with them, they have found a wide range of applications including curve and surface reconstruction [63], image data analysis [32], coverage in sensor networks [176], shape

description [50]. See [58, 68, 92, 191] for more extensive surveys of applications.

We can think of any equivalence relation as a crude similarity measure, in which two curves are at distance 0 if they are equivalent and distance ∞ if they are not. More refined similarity measures can be defined by considering the geometry of the underlying space. Measures of similarity like Hausdorff distance, earth mover distance and the Frechét distance have found several applications in computer science [120, 123, 148, 178]. For embedded curves on a surface, and particulary for homotopic curves, a natural measure of similarity is the minimum "cost" of a deformation of one curve to the other. Different costs of deformation has been considered in the literature [19, 40, 69, 187].

Different algorithms are required to classify or compare curves in different representations. Curves can be combinatorially presented as walks in the primal or dual cell complex of the underlying surface, or equivalently by recording the sequence of triangles they intersect [45, 47, 75]. However, this simple method can be far from being efficient. More compact methods include weighted train tracks [11, 12], Dehn-Thurston coordinates (with respect to a fixed pants decomposition of the surface) [55, 82], compressed intersection sequence [166, 179] and normal coordinates [4, 163].

Normal coordinates were originally invented by Haken and Kneser [102, 127] to study problem about embedded surfaces in three manifolds. Because of their compactness they are used to build polynomial size certificates for several problems; examples are recognizing a trivial knot [4, 107], recognizing a three-sphere [169] and recognizing string graphs [164]. On the other hand, the compressed presentation of curves make their algorithmic problems more challenging; therefore, most of the algorithmic results about compressed curves use highly non trivial techniques like grammar based compression and word equations [163–165, 179].

1.1 History

One of the oldest algorithms in this area is Dehn's algorithm [54] to test whether two given curves on a surface are homotopic, and specifically whether a curve can be continuously contracted (it is null-homotopic). There are essentially two main approaches to attack this problem. Building an appropriate covering space was originally suggested by Schwartz, and further developed by Poincaré [154] and Dehn [53]. On the other hand, Dehn observed that any contractible cycle can be contracted through a set of local greedy moves. For a fixed surface Dehn's algorithm running time is linear in the complexity of the input curves, with a constant that depends on the genus of the surface [13].

Several authors have considered the following problem. Given a surface of complexity n and one embedded curve (resp. two embedded curves) with total complexity ℓ, decide wether it is null-homotopic (resp. they are homotopic). Dey and Schipper [62, 168] describe an algorithm that tests the contractibility in $O(n+$

$\ell \log g$), observing that only a small portion of the universal cover is required to be investigated. Later Dey and Guha [59] reconsidered Dehn's method of greedy contracting to obtain the first truly linear ($O(n+\ell)$) time algorithm. Later, Lazarus and Rivaud [131] reported a subtle flaw in their algorithm and describe another linear time algorithm that constructs a small portion of a certain covering space. Very recently, Erickson and Whittlesey [80] recast and simplify Lazarus and Rivaud's algorithm using the language of small cancelation theory [91] to avoid searching any covering space.

Testing if two polygonal paths on the plane minus points are homotopic is particularly interesting to many applications including circuit routing [136], motion planning [96] and map simplification [9, 25]. Cabello *et al.* [30] describe an $O(n \log n)$ time algorithm for this Euclidean version of the problem, and they proved their running time is asymptotically optimal.

As a generalization to the classical shortest path problems (either geometric or graph theoretic), many authors have considered the problem of computing the shortest curve in a given homotopy class. Hershberger and Snoeyink [110] first used the covering space concept to compute the shortest path (under two different metrics) homotopic to a given path, in a boundary triangulated piecewise linear surface of genus 0. Borrowing ideas from the decision algorithm of Cabello *et al.* [30], Efrat *et al.* [70] and Bespamyatnikh [10] describe algorithms for Euclidean plane minus a finite set of points.

Colin de Verdière and Lazarus [48, 49] describe an algorithm to find the shortest simple cycle homotopic to a given *simple* cycle in a surface embedded graph. Their algorithm runs in polynomial time with respect to the complexity of the surface and the input curve. Colin de Verdière and Erickson [46] give algorithms to compute the shortest path (resp. cycle) homotopic to an arbitrary path (resp. cycle) of length k in $O(gnk)$ (resp. $O(gnk \log(nk))$) time.

Testing whether two given cycles are homologous or not is essentially equivalent to solving a system of linear equations; more precisely to check whether it has a valid solution. Erickson and Whittlesey [79] describe an algorithm to compute the homology class of any cycle in constant time per edge, after $O(gn)$ preprocessing time. However, the optimization problem of finding the shortest cycle in a given homology class is more difficult. An argument of Chambers *et al.* [35] implies that finding the shortest cycle (either simple or not) in a given homology class in a surface graph is NP-hard; Chen and Friedman [42, 43] proved that the corresponding problem in simplicial complexes is NP-hard to approximate within any constant factor. In this thesis we prove that computing the minimum element of a $\mathbb{Z}_2$-homology class is NP-hard, yet fixed parameter tractable on surfaces. On the positive side, we describe an efficient algorithm to compute the minimum element of a $\mathbb{Z}$-homology class in a given surface. Dunfield and Hirani [66] show that computing the minimum element of a $\mathbb{Z}$-homology class is NP-hard in general.

Computing shortest cycles in an equivalence class (homotopy or homology) is intimately related to other well known problems like computing the minimum

cut [39], computing the minimum non-separating cycle, computing the minimum non-contractible cycle [76, 183], computing a tight system of loops and computing a short cut graph [79]. See Chapter 3 for more explanation.

Given two homotopic cycles, one can consider infinitely many homotopies to transform one to the other. However, depending on the application different homotopies may have different costs [19, 40, 187]; it is natural to ask how to find an optimal homotopy with respect to a given cost function. Brightwell and Winkler [19] and Chambers and Letscher [40] define the homotopy height to be the longest curve within the homotopy. Surprisingly, almost nothing is known about the homotopy height problem beside the fact that it can be solved in polynomial space for unweighted graphs.

The problem of finding the minimum cost homotopy is closely related to the homotopic Frechét distance problem. Frechét distance has been used to compare different objects in a various range of applications, including dynamic time warping [120], time series matching in databases [123], melody comparing in music information retrieval [173], map-matching of vehicle tracking [18, 190] and moving object analysis [20, 21].

Chambers *et al.* [36] observed that, although Frechét distance is natural for Euclidean ambient spaces it ignores important features of more general ambient spaces. As an alternative, they suggested the homotopic Frechét distance as a measure of simularity for curves; the homotopic Frechét distance and the classic Frechét distance are identical in Euclidean ambient spaces. The homotopic Frechét distance of two curves f and g is intuitively defined as follows. Imagine that a person and a dog are simultaneously moving on f and g with no backward step. Then, the homotopic Frechét distance is the shortest possible leash that they need to be connected during the entire move. The leash should be always embedded on the surface and it only deforms continuously. More formally, the leash is a curve in the surface that deforms continuously as the man and dog move. Similar to the homotopy height problem, we only know few trivial facts about the complexity of computing homotopic Frechét distance in general. Chambers *et al.* [36] described polynomial time algorithms for curves in the Eculidean plane with polygonal punctures. In this thesis we provide the first $O(\log n)$ approximation algorithms for both homotopy height and homotopic Frechét distance in arbitrary discrete and piecewise linear surfaces embedded in $\mathbb{R}^3$.

Classification problems can be asked about curves (or surfaces) presented compactly in normal coordinates. Schaefer *et al.* [163, 166, 179] consider several algorithmic questions about normal curves, such as computing the number of components of a curve, deciding whether two given curves are isotopic, and computing algebraic and geometric intersection numbers of pairs of curves. Classical algorithms for these problems require explicit traversal or crossing sequences as input. By connecting normal coordinates with grammar-based text compression [132, 133, 141, 161] and word equations [65, 152, 159, 160], Schaefer *et al.* developed algorithms whose running times are polynomial in the bit complexity of the normal coordinate vector,

which they call the *normal complexity* of the curve. These algorithms rely on a complex algorithm of Plandowski and Rytter [152] to compute compressed solutions of word equations. We are unaware of any precise time analysis, but as Plandowski and Rytter's algorithm uses a nested sequence of quadratic- and cubic-time reductions, its running time is quite high. Štefankovic [179] described simpler algorithms for some of these problems in time *linear* in the normal complexity, or $O(n \log(X/n))$ time, by reducing them to an elegant algorithm of Robson and Deikert [159, 160] to solve word equations with a certain special structure. Some of the problems considered by Schaefer *et al.* can also be solved in polynomial time using the polynomial-time *orbit-counting* algorithm of Agol, Hass, and Thurston [4]. Dynnikov and Wiest [67] later developed a special case of the orbit-counting algorithm to reconstruct braids from their planar curve diagrams; Dehornoy *et al.* [56] refer to this variant as the *transmission-relaxation method*.

1.2 New results

In this thesis we consider several computational problems about embedded curves on surfaces. We mostly focus on problems that are related to classification of curves and measuring their similarity. In Chapter 2 we briefly overview some required background.

In Chapter 3 we concentrate on the problem of finding the minimum element of a $\mathbb{Z}_2$-homology class; the results in this chapter are joint work with Erin Chambers and Jeff Erickson [39, 77]. The elements of $\mathbb{Z}_2$-homology classes are even subgraphs (or equivalently, collections of cycles). An even subgraph is null-homologous if it is the boundary of a subset of faces; two even subgraphs are homologous of their union is null-homologous.

Since $\mathbb{Z}_2$-homology groups are vector spaces, each homology class can be represented by a bit vector, called its homology signature, whose length is linear in the genus of the graph. This bit vector is dependent to the choice of the basis of the homology vector space. The basis can be represented by a set of paths, called a system of arcs, whose removal leaves a topological disc. Let $[H]$ denote the signature of the even subgraph H, with respect to a certain basis. Then $[H]$ encodes the parity of the number of times that H crosses the paths of the homology basis. To compute the shortest even subgraph homologous to H, we observe that such an even subgraph can be decomposed to a set of simple cycles each as short as possible in its homology class, and whose sum of signatures is $[H]$. To this end, we compute the shortest cycle in every homology class and then use a dynamic programming to merge them.

The high level idea of our algorithm to compute the shortest cycle in a given homology class is as follows. Assume that we know a vertex v of the graph that is on the cycle that we are looking for. Now, we want to compute a cycle through v that has a certain signature and it is as short as possible. We build a new larger

space that encodes both the destination and the signature of paths. Each vertex in the covering space is a pair composed of a vertex of the graph and a signature (bit vector) (u, δ). Cycles that contain v with signature δ are projections of paths in the covering space from $(v, 0)$ to (v, δ), thus the problem is reduced to computing shortest paths in a larger space. Fortunately, we can prove that the larger space is not too large, and we can speedup our algorithm using fast multiple source shortest path finders [27, 125]. Overall, our algorithm runs in $2^{O(g)} n \log n$ time. In an earlier result [39] we described an $g^{O(g)} n \log n$ algorithm to solve the same problem; Italiano *et al.* [114] recently improved the running time of our algorithm to $g^{O(g)} n \log \log n$.

Given a graph embedded on a surface and two vertices s and t, we prove that the minimum (s, t)-cut is dual to shortest even subgraph in a certain $\mathbb{Z}_2$-homology class. We immediately a near linear time algorithm to compute the minimum cut in an undirected surface embedded graph of constant genus. On the other hand, the minimum non-separating cycle is the shortest cycle that is not null-homologous. Since our algorithm finds the shortest directed cycle in every homology class, in particular, it finds the shortest directed non-separating cycle.

The running time of our algorithm depends exponentially on the genus of the graph. We show that this exponential dependency is necessary by proving that computing the minimum $\mathbb{Z}_2$-homologous even subgraph is NP-hard.

In Chapter 4 we consider the maximum flow problem for surface embedded graphs; the results in this chapter are joint work with Erin Chambers and Jeff Erickson [37, 38]. The maximum flow problem is the linear programming dual of the minimum cut problem; however, our solution requires significantly different techniques. To solve the maximum flow problem, we observe that homology can be used to partition the space of all flows in a surface embedded graph. A flow is feasible (with respect to a capacity function on the edges) if it does not oversaturate any edge of the graph. We say that a homology class is feasible if it contains a feasible flow. We describe a canonical way to represent each homology class by computing what we call a flow homology basis, which resembles a system of arcs. Generalizing a technique used in planar maximum flow algorithms [186], we obtain an oracle to compute a feasible flow within a given flow homology class if one exists or recognize that the given homology class is not feasible. We use the membership separation oracle to obtain a $O(g^8 n \log^2 n \log^2 C)$ time algorithm if the inputs are integers whose sum is at most C, and a $g^{O(g)} n^{3/2}$ time algorithm in general to compute the maximum flow on a surface embedded graph of genus g.

In Chapter 5 we describe efficient $O(\log n)$-approximation algorithms for two closely related problems, namely the homotopy height and the homotopic Frechét distance; the results in this chapter are joint work with Sariel Har-Peled, Mohammad Salavatipour and Anastasios Sidiropoulos [103]. Both of our algorithms work with the assumption that the input curves are on the boundary of a triangulated disk. Our algorithm to compute the homotopy height exploits a simple divide an conquer strategy. Roughly speaking, we find a short intermediate path that splits the disk into

two smaller disks with roughly equal complexity; we then solve the problem within each subdisk recursively. The paths we find at each level of recursion are longer than the paths found in the previous level by at most a constant times the actual homotopy height; the length of the first splitting path is at most a constant times the actual homotopy height; and the recursion tree has $O(\log n)$ levels. Thus the length of the longest path computed by our algorithm is a $O(\log n)$-approximation of homotopy height.

We use our homotopy height algorithm as an ingredient for an approximation algorithm for the homotopic Frechét distance problem. At high level our algorithm performs a search over possible values for the homotopic Frechét distance. To determine whether our current guess is too small or too large, we classify regions of the disk that are far from both curves as obstacles. Using a greedy algorithm, we compute a subset of the disk that avoid all obstacles, such that the homotopic Frechét distance within that subset is a constant-factor approximation of the true homotopic Frechét distance. We then combine a constant factor approximation algorithm of regular Frechét distance and our $O(\log n)$ approximation algorithm for the homotopy height to obtain a leash sequence whose longest leash is $O(\log n)$ times longer than the homotopic Frechét distance.

The $O(\log n)$ factor shows up in the homotopic Frechét distance algorithm only because it uses the homotopy height as a subroutine. Thus, any constant factor approximation algorithm for the homotopy height problem implies a constant factor approximation algorithm for the homotopic Frechét distance.

Finally, in Chapter 6 we propose an efficient strategy to compute with curves represented by normal coordinates; the results in this chapter are joint work with Jeff Erickson [78]. Instead of using complex compression techniques to avoid unpacking the crossing sequence of the input curve, our algorithms *modify the underlying cellular decomposition* of the surface so that the curve has a small *explicit* description with respect to the new decomposition. Specifically, given the normal coordinates of a curve γ on a triangulated surface with n edges, we compute a new cellular decomposition of the surface with complexity $O(n)$, called a *street complex*, such that γ is a simple path or cycle in the 1-skeleton. See Section 6.4 for the formal definition and Figure 6.2 for an example.

At a high level, our algorithm simply *traces* the curve, continuously updating the street complex to reflect the portion of the curve traced so far. A naïve implementation of our tracing strategy runs in $O(X)$ time, where X is the total number of edge crossings; each time the curve enters a triangle by crossing an edge, we can easily determine in $O(1)$ time which of the other two edges of the triangle to cross next. We describe a tracing algorithm that runs in $O(n^2 \log X)$ time, an exponential improvement over the naïve algorithm for any fixed surface triangulation.

Our new algorithm relies on two simple ideas. First, we observe that for typical curves, most of the decisions made by the brute-force tracing algorithm are redundant. If a curve enters a triangle Δ between two older elementary segments that leave Δ through the same edge, the new elementary segment must also leave Δ

through that edge; see Figure 1.1. The street complex allows us to skip these redundant decisions automatically.

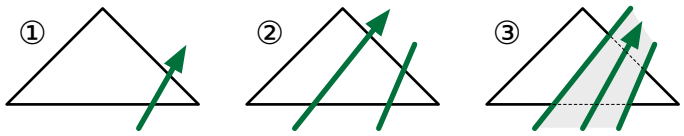


Figure 1.1. Tracing three segments of a curve through a triangle. Tracing the third segment does not require any decisions.

Second, even with redundant decisions filtered out, the naïve algorithm may repeat the same series of crossings many times when the input curve contains a *spiral* [67, 146, 165, 167]. Our algorithm detects spirals as they occur, quickly determines the depth of the spiral (the number of repetitions), and then skips ahead to the first crossing after the spiral. As a consequence of our tracing algorithm, we obtain efficient algorithms for several problems about normal curves such as computing the number of components of a curve, deciding whether two given curves are isotopic, and computing algebraic and geometric intersection numbers of pairs of curves. Classical algorithms for these problems require explicit traversal or crossing sequences as input.

Chapter 2

Background

In this chapter, we review several fundamental definitions, which are necessary throughout the thesis, related to surfaces and surface embedded graphs. For more comprehensive treatments, we refer the interested reader to Gross and Tucker [100] and Mohar and Thomassen [143] for topological graph theory, and to Hatcher [108] and Stillwell [180] for topology.

2.1 Graphs

Let $G = (V, E)$ be a directed graph. A directed ***(u, v)-walk*** in G is an alternating sequence of vertices and directed edges $W = (u = w_0, e_1, w_1, e_2, \cdots, e_k, w_k = v)$, where $e_i = w_{i-1} \rightarrow w_i$ for all $1 \leq i \leq k$. A (u, v)-walk is ***closed*** if $u = v$, a ***path*** if it has no repeated vertices, and a ***cycle*** if it is a path with $u = v$ its only repeated vertex. Let W_1 be a (u, v)-walk and W_2 be a (v, w)-walk; their ***concatenation*** $W_1 \cdot W_2$ is defined as the concatenation of their corresponding vertex sequences.

A ***cut*** (S, T) in G is a partition of vertices into two disjoint subsets S and T; (S, T) is an ***(s, t)-cut*** if and only if $s \in S$ and $t \in T$, for any two vertices s and t.

A ***spanning tree*** T of a connected graph $G = (V, E)$ is a maximal subgraph of G that contains no cycles.

2.2 Surfaces

A ***surface*** Σ (or a 2-manifold) is a topological space where every point has a neighborhood homeomorphic to either a 2-dimensional Euclidean plane or a closed half-plane. The union of all the points in the space that are homeomorphic to a half-plane is called the ***boundary*** of Σ and it is denoted by $\partial\Sigma$. The boundary of a surface is homeomorphic to the union of a finite set of disjoint circles. A surface is ***non-orientable*** if it contains a subspace homeomorphic to the Möbius band, and it is ***orientable*** otherwise.

A ***path*** in a surface Σ is a continuous function $\sigma : [0, 1] \rightarrow \Sigma$. A ***loop*** is a path whose endpoints $p(0)$ and $p(1)$ coincide; we refer to this common endpoint as the ***basepoint*** of the loop. An ***arc*** is a path whose endpoints lie on the boundary of Σ. A ***cycle*** is a continuous function $\gamma : S^1 \rightarrow \Sigma$. We refer to any union of paths, loops, arcs, and cycles as ***curves***; indeed a curve may be disconnected. We will usually

not distinguish between a path/cycle and its image in Σ. A curve is ***simple*** if the function that defines it is injective, except for the basepoint in the case of loops. The ***reversal*** $\overline{p}$ of a path p is defined by setting $\overline{p}(t) = p(1-t)$. The ***concatenation*** $p \cdot q$ of two paths p and q with $p(1) = q(0)$ is the path created by setting $(p \cdot q)(t) = p(2t)$ for all $t \leq 1/2$ and $(p \cdot q)(t) = q(2t-1)$ for all $t \geq 1/2$.

A simple arc is ***properly embedded*** if it intersects $\partial\Sigma$ only at its endpoints; similarly, a simple cycle is properly embedded if it avoids $\partial\Sigma$ entirely. A ***properly embedded curve*** is a finite collection of disjoint, properly embedded arcs and cycles. We emphasize that curves may have multiple components.

A cycle γ is ***separating*** if $\Sigma \backslash \gamma$ is not connected. The ***genus*** of a surface Σ is the maximum number of disjoint cycles whose removal leaves Σ connected.

2.3 Auxiliary directed graph

For an undirected graph $G = (V, E)$, for the sake of argument we define an auxiliary directed graph $\vec{G} = (V, \vec{E})$ by conceptually replacing each undirected edge with a pair of antisymmetric directed edges. Following Borradaile and Klein [16] we refer to the directed edges as ***darts***. We emphasize that the darts are really conceptual directed edges that do not change the embedding of G, and we do not introduce any new faces. Let $\vec{e} = u \to v$ be a dart that leaves u and enters v. We call u and v the ***tail*** and the ***head*** of $\vec{e}$, respectively. The ***reversal*** of $\vec{e}$, denoted by $v \to u$, is the dart that leaves v and enters u. The definition of darts can be extended to cover graphs with loops and parallel edges, where different darts correspond to different parallel edges and a dart and its reversal are identical if and only if they correspond to a loop.

2.4 Graph embedding

An ***embedding*** of a graph $G = (V, E)$ on a surface Σ is composed of a mapping from V to distinct points of Σ and a collection of mappings from E to paths in Σ that are disjoint except at common endpoints. A ***face*** of an embedding is a maximal connected subset that avoids the image of V and E. An embedding is ***cellular*** if its faces are homeomorphic to open discs. In any cellular embedding, each connected component of a the boundary of Σ as well as the boundary of each face of the embedding is covered by a closed walk of G. Any cellularly embedded graph can be presented by a ***rotation system***, which is a permutation π of the darts, where $\pi(\vec{e})$ is the dart that appears after $\vec{e}$ in the counterclockwise ordering of darts leaving $tail(\vec{e})$.

Suppose G is a n-vertex graph cellularly embedded on a surface Σ of genus g with b boundaries. According to Euler's formula $|V| - |E| + |F| = 2 - 2g - b$ if Σ is orientable and $|V| - |E| + |F| = 2 - g - b$ if Σ is non-orientable. In either case, Euler's formula implies that there are $O(n + g + b)$ edges and faces if G is simple.

Given a cellular embedding of G, every dart separates two (possibly equal) faces of G, called the ***left shore*** and the ***right shore***. We use $f \uparrow f'$ to denote a dart whose left shore is f and whose right shore is f'; thus, $rev(f \uparrow f') = f' \uparrow f$.

Two paths in a combinatorial surface ***cross*** if no continuous infinitesimal perturbation makes them disjoint; if such a perturbation exists, then the paths are ***non-crossing***. We say that a cycle γ is ***non-self-crossing*** if no two sub-paths of γ cross, ***weakly simple*** if γ is non-self-crossing and traverses each edge at most once, and ***(strictly) simple*** if γ visits each vertex at most once.

Cutting a surface along a cycle or an arc modifies both the surface and the embedded graph. Let G be a graph embedded on a surface Σ and γ be any cycle G, we define a new embedded graph $G \not\downarrow \gamma$ and a new surface $\Sigma \not\downarrow \gamma$ by taking the topological closure of $\Sigma \setminus \gamma$ as the new underlying surface; the new embedded graph contains two copies of each vertex and edge of γ both on the border of its boundary.

2.5 Duality

Let $G = (V, E)$ be a graph embedded on a surface Σ of genus g with no boundary, and let F be the set of faces in the embedding. The ***dual*** graph G^* is defined as an embedded graph on Σ that has a vertex f^* for each face $f \in F$. There is an edge $v \rightarrow u$ in G if and only if $v^* \uparrow u^*$ is an edge in G^*.

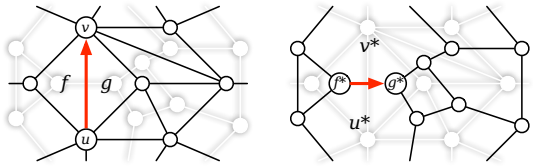


Figure 2.1. Graph duality. One edge $u \rightarrow v$ and its dual $(u \rightarrow v)^* = f^* \uparrow g^*$ are emphasized.

The ***tree-cotree decomposition*** (T, C, L) of G is composed three disjoint sets: T is a spanning tree of G, C^* is a spanning tree of G^*, and L is a set of extra edges. Euler's formula implies that L contains $2g$ or g edges if Σ is orientable or non-orientable, respectively.

If Σ is a surface with boundary, G^* has ***boundary vertices*** that correspond to boundary faces of G. In this case, we can obtain a ***tree-coforest decomposition*** by computing the tree-cotree decomposition (T, C, L), make the dual spanning tree C^* a rooted spanning tree R^* by picking an arbitrary boundary vertex to be the root, and remove all boundary vertices to obtain the forest F^*. Each vertex of F^* is assigned to its lowest ancestor in R^* that is a boundary vertex. Since the root of R^* is a boundary vertex each connected component of F^* is assigned to a single boundary vertex. Equivalently, the coforest F has one component per boundary.

2.6 Homotopy and isotopy

Homotopy is an equivalence relation between curves that captures the notion of continuous deformation. Two paths p and p' are ***homotopic*** if there is a continuous map $h\colon [0,1]\times[0,1]\to\Sigma$ such that $h(0,t)=p(t)$ and $h(1,t)=p'(t)$ for all t, and $h(\cdot,0)$ and $h(\cdot,1)$ are constant maps. Two cycles γ and γ' are ***(freely) homotopic*** if there is a continuous map $h\colon [0,1]\times S^1\to\Sigma$ such that $h(0,t)=\gamma(t)$ and $h(1,t)=\gamma'(t)$ for all t. A loop or cycle is ***contractible*** if it is homotopic to a constant map; an arc is contractible if it is homotopic to a subpath of a boundary cycle.

A ***(proper) isotopy*** between two cycles γ and γ' is a continuous map $h\colon [0,1]\times S^1\to\Sigma$ such that $h(0,\cdot)=\gamma$ and $h(1,\cdot)=\gamma'$, and $h(t,\cdot)$ is a properly embedded cycle for all $t\in[0,1]$. Similarly, a (proper) isotopy between two arcs α and α' is a continuous map $h\colon [0,1]\times[0,1]\to\Sigma$ such that $h(0,\cdot)=\alpha$ and $h(1,\cdot)=\alpha'$, and $h(t,\cdot)$ is a properly embedded arc for all $t\in[0,1]$. The definition of isotopy extends naturally to properly embedded curves with multiple components. Two curves are ***isotopic***, or in the same ***isotopy class***, if there is a isotopy between them.

2.7 Chains, circulations, and flows

Let $G=(V,E)$ be a graph cellularly embedded on an orientable surface Σ, and let F be the set of faces of the embedding. A 2-chain $\alpha : F\to\mathbb{R}$ assigns weights to the faces of the graph; a 1-chain $\phi : E\to\mathbb{R}$ assign weights to the edges of the graph; and a 0-chain $\omega : V\to\mathbb{R}$ assigns weights to the vertices of the graph. It is helpful to think about a 1-chain as a function that assigns numbers to each dart such that $\phi(\vec{e})=-\phi(rev(\vec{e}))$.

The boundary of a 1-chain ϕ is defined as

$$\partial\phi(v)=\sum_{u:u\to v\in\vec{E}}\phi(u\to v).$$

Similarly, the boundary of a 2-chain α is defined as

$$\partial\alpha(f\uparrow g)=\alpha(f)-\alpha(g).$$

A ***circulation*** is a 1-chain whose boundary is trivial; that is the ***conservation constraint*** $\partial\phi(v)=0$ holds for every vertex v. For any two vertices s and t a 1-chain whose boundary is 0 everywhere except possibly at s and t is called an ***(s,t)-flow***. In this case $|\partial\phi(s)|=|\partial\phi(t)|$ is called the ***value*** of the flow ϕ, denoted $|\phi|$.

The (first) ***chain space $C(G)$*** of an embedded graph is the vector space of all possible 1-chains, which is clearly isomorphic to $\mathbb{R}^{|E|}$. The cycle space $Z(G)$ is the vector space of all possible circulations, which is isomorphic to $\mathbb{R}^{|E|-|V|+1}$. Finally, the ***flow space $Z(G;st)$*** is the space of all (s,t)-flows, which is isomorphic

to $\mathbb{R}^{|E|-|V|+2}$. Intuitively, one can think of a flow space as the cycle space when s and t are identified.

2.8 Homology

The boundary of any 2-chain $\alpha : F \to \mathbb{R}$ is defined to be $\partial\alpha : E \to \mathbb{R}$, where $\partial\alpha(f \uparrow g) = \alpha(g) - \alpha(f)$ for any edge $f \uparrow g \in E$. A ***boundary circulation*** is the boundary of some 2-chain, which is indeed a circulation. In planar graphs, any circulation is a boundary circulation as well, which is not true for higher genus surfaces. The ***boundary space $B(G)$*** is a vector space of all possible boundary circulations, which is isomorphic to $\mathbb{R}^{|F|-1}$.

Two flows or circulation (any two 1-chains, in general) ϕ and ψ are ***homologous*** (or in the same *homology class*) if and only if $\phi - \psi$ is a boundary circulation. Thus the ***homology space $H(G)$***, the space of all homology classes, is the quotient space $Z(G)/B(G)$, which is by definition homeomorphic to $\mathbb{R}^{|E|-|V|-|F|+2}$ which is $\mathbb{R}^{2g}$ by Euler's formula. The ***(s,t)-flow homology space $H(G; s\ t)$*** is similarly defined to be the space of all homology classes of (s,t)-flows, which is isomorphic to $\mathbb{R}^{2g+1}$.

2.9 Piecewise linear surfaces

A ***piecewise linear*** surface is composed of a finite number of Euclidean polygons by identifying pairs of equal length edges. The interiors of the constituent polygons are called the *faces* of the surface. The *vertices* and *edges* of the surface are the equivalence classes of the vertices and edges of the polygons. A piecewise linear is a ***triangulation*** if all its faces are triangles.

An embedding of a triangulation Σ to $\mathbb{R}^d$ is an injective map $\Phi : \Sigma \to \mathbb{R}^d$, such that each triangle maps to the convex hull of three points in $\mathbb{R}^d$. We say that a triangulation can be embedded in $\mathbb{R}^d$ if such an embedding exists. Most piecewise-linear surfaces cannot be embedded in *any* Euclidean space; consider, for example, the *flat torus* obtained by identifying opposite sides of the unit square.

A ***geodesic*** is a path that is *locally* as short as possible; for any point x in a geodesic γ, a sufficiently small neighborhood of γ around x is a shortest path; in particular any shortest path is a geodesic. If γ is a geodesic in a piecewise-linear surface Σ, any subpath of γ that lies entirely within a face of Σ is a straight line segment. Similarly, a subpath of γ that crosses an edge of Σ from one face A to another face B is a line segment in the polygon obtained by *unfolding* A and B into a common planar coordinate system [57, 135]. A geodesic is ***simple*** if it does not self-intersect. We emphasize that a simple geodesic may cross each face of a piecewise-linear surface arbitrarily many times, or even *infinitely* many times; again, consider the flat torus.

2.10 Curve similarity

Comparing embedded curves is a challenging task with many applications, which is elaborated in Chapter 5. In this section, we provide some preliminary definitions, which are mostly used in that chapter. The subject of the definitions, and in fact our algorithms, are paths instead of curves.

2.10.1 Frechét distance

Let Σ be a topological space with a metric $d : \Sigma^2 \to \mathbb{R}$, and let f and g be two paths in Σ. A parametrization $\phi : [0,1] \to [0,1]$ is a bijective continuous function. The ***width*** of a parametrization with respect to f and g is

$$\text{width}(\phi) = \max_{x \in [0,1]} d\Big(f(x), g(\phi(x))\Big)$$

The Frechét distance between f and g is defined to be

$$d_{\mathcal{F}}(f,g) = \inf_{\phi:[0,1]\to[0,1]} \text{width}(\phi),$$

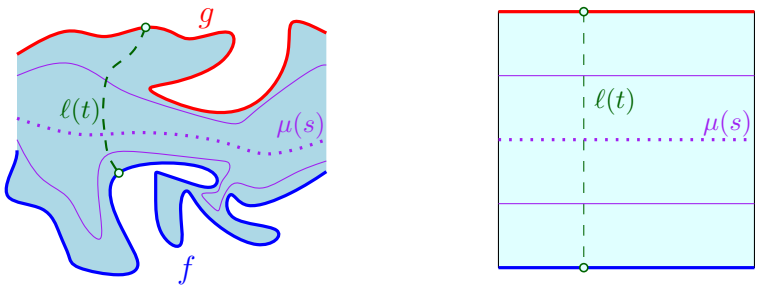
where ϕ ranges over all orientation-preserving homeomorphisms.

While this distance makes sense when the underlying metric is Euclidean, it becomes less natural if the distance function is more interesting. For example, imagine walking a dog in the woods. The leash might get tangled as the dog and the person walk on two different sides of a tree. Since the Frechét distance cares only about the distance between the two moving points, the leash would "magically" jump over the tree.

2.10.2 Homotopic Frechét distance and homotopy height

To address this shortcoming, a natural extension called ***homotopic Frechét distance*** was introduced by Chambers *et al.* [34]. Informally, revisiting the above person-dog analogy, we consider the infimum over all possible traversals of the paths, but this time, we require that the person is connected to the dog via a leash. The homotopic Frechét distance is the minimum length of a leash that allows the dog and its owner to simultaneously traverse the curves.

More formally, consider a homotopy $h : [0,1]^2 \to \Sigma$, and four paths f, r, g, ℓ whose concatenation in this order is a closed walk. For fixed parameters s and t consider $\ell_t(y) = h(t,y)$ and $\mu_s(x) = h(x,s)$ as functions of y and x, respectively. The functions $\mu(y) \equiv \mu_t(y)$ and $\ell(x) \equiv \ell_s(x)$ are parametrized curves that are the natural restrictions of h to one dimension, by the x and y coordinates, respectively. We require that $\mu(0) = f$, $\mu(1) = g$, $\ell(0) = \ell$ and $\ell(1) = r$. The ***homotopic width*** of h is $\text{width}(h) = \max_{t \in [0,1]} |\ell(t)|$, and the ***homotopic Frechét distance*** between f and



(i) (ii)

Figure 2.2. (i) Two paths f and g, and (ii) the parametrization of their homotopic Frechét distance.

g is

$$d_{\mathcal{H}}(f,g) = \min_{h:[0,1]^2 \to \Sigma} \text{width}(h),$$

Clearly, $d_{\mathcal{H}}(f,g) \geq d_{\mathcal{F}}(f,g)$. Further, $d_{\mathcal{H}}(f,g)$ can be arbitrary larger than $d_{\mathcal{F}}(f,g)$. We remark that $d_{\mathcal{H}}(f,g) = d_{\mathcal{F}}(f,g)$ for any pair of paths in Euclidean space of any dimension, as we can always pick the leash to be a straight line segment between the person and the dog. However, this is not true for general ambient spaces, where the leash might have to pass over obstacles or hills. In particular, in most spaces, usually, the leash is not always a shortest path during the motion.

Efrat *et al.* [69] refer to the homotopic Frechét distance as the ***morphing width*** of f and g, which bounds how far a point on f has to travel to its corresponding point in g under the morphing of h. The length of $\mu(s)$ is the *height of the morph at time* s, and the ***height*** of such a morphing is $\text{height}(\mu) = \max_{s\in[0,1]} |\mu(s)|$. The ***homotopy height*** between f and g bounded by $\ell(0)$ and $\ell(1)$ is

$$hh(f,g,\ell(0),\ell(1)) = \inf_{\mu} \text{height}(\mu),$$

where h varies over all possible maps $h : [0,1]^2 \to \Sigma$ such that $h(0,\cdot) = \ell$, $h(1,\cdot) = r$, $h(\cdot,0) = f$ and $h(\cdot,1) = g$. See Figure 2.2 for an example. Note that if we do not constraint the endpoints of the paths during the homotopy to stay on $\ell(0)$ and $\ell(1)$, the problem of computing the minimum height homotopy is trivial. One can contract f to a point, send it to g from the shortest (f,g)-path, and then expand it to g. To keep the notation simple, we use $hh(f,g)$ when f and g have common endpoints.

Intuitively, the homotopy height measures how long the path has to become as it deforms from f to g, and it was introduced by Chambers and Letscher [40, 41] and Brightwell and Winkler [19]. Observe that if we are given the starting and ending leashes $\ell(0)$ and $\ell(1)$ then the homotopy height of f and g, is the homotopic Frechét distance between $\ell(0)$ and $\ell(1)$.

2.10.3 Discrete problems

Let W_1 be an (s,t)-walk and f be a face in G. Assume that α_1 is a subwalk of W_1 and $\partial f = \alpha_1 \cup \alpha_2$, where α_1 and α_2 are walks that share endpoints u and v, such that u is closer to s on W_1. The ***face flip*** operation is defined as follows. The walk $W_2 = W_1[s,u] \cdot \alpha_2 \cdot W_1[v,t]$ is the result of flipping W_1 over f. In this case, we say that W_1 and W_2 are one face flip operation apart.

Now, let W_1 be an (s,t)-walk and $e = u{\rightarrow}v$ be an edge in G. Assume that $u \in W_1$. We obtain the walk $W_2 = W_1[s,u] \cdot (u{\rightarrow}v) \cdot (v{\rightarrow}u) \cdot W_1[u,t]$ after applying a ***spike*** operation on W_1 along e. In this case, we can obtain W_1 from W_2 by applying a ***reverse spike*** operation along e. We say that W_1 and W_2 are a spike operation apart. In general, we say that W_1 and W_2 are one operation apart if we can transform one to the other using a single face flip, spike, or reverse spike. Letscher and Chambers refer to the same set of operations as: face lengthening, face shortening, spike and reverse spike.

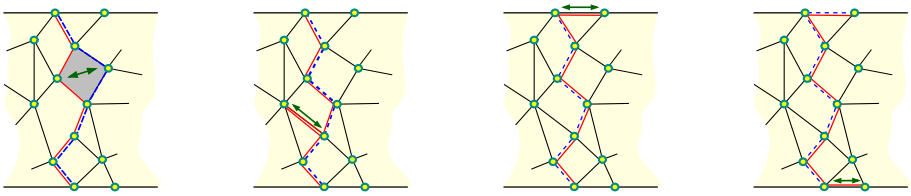


Figure 2.3. From left to right: face-flip, spike/reverse spike, man-move and dog-move.

Let $A = (a_0, a_1, \ldots, a_k)$ and $B = (b_0, b_1, \ldots, b_l)$ be walks of G. An (A,B)-walk is a walk that has one endpoint on A and one endpoint on B. The walk $W_1 = (a_i = w_1, w_2, \ldots, w_k = b_j)$ changes to the walk $W_2 = (a_{i+1}, a_i = w_1, w_2, \ldots, w_k)$ after a ***man move***. Similarly, the walk $W_1 = (a_i = w_1, w_2, \ldots, w_k = b_j)$ changes to the walk $W_2 = (w_1, w_2, \ldots, w_k = b_j, b_{j+1})$ after a ***dog move***. An ***endpoint move*** is either a man move or a dog move. A ***leash operation*** is a man move, a dog move, a face flip, a spike or a reverse spike.

A sequence of (A,B)-walks, $(W_1, W_2, \ldots, W_q)$ is called an (A,B)-***leash sequence*** if W_1 is a (a_0, b_0)-walk, W_q is a (a_k, b_l)-walk and for all $1 \leq i < q$, W_i changes to W_{i+1} by a set of leash operations that contains exactly one endpoint move. The height of a leash sequence is the length of its longest walk. The ***discrete Frechét distance*** of A and B is the height of the minimum height (A,B)-leash sequence. The leash sequence $(W_1, W_2, \ldots, W_q)$ contains no gap if W_i changes to W_{i+1} by exactly one leash operation. The ***discrete homotopic Frechét distance*** of A and B is the height of the minimum height (A,B)-leash sequence that contains no gap.

Let L and R be two (s,t)-walks on the outer face of G with shared endpoints. The sequence of walks $(L = W_0, W_1, \ldots, W_m = R)$ is a ***(L,R)-discrete homotopy*** if for all $1 \leq i \leq m$, W_{i-1} changes to W_i with a single face flip, spike or reverse spike. We may use the word homotopy as a short form of discrete homotopy when it is clear from context. The height of the homotopy is the length of the longest walk in its sequence. The ***discrete homotopy height*** between L and R, is the height of the

shortest possible (L, R)-homotopy.

Chapter 3

$\mathbb{Z}_2$-homologous cycles

In this chapter, we consider two related problems of testing whether two even subgraphs are Z_2-homologous and computing the minimum even subgraph or the minimum cycle in a given $\mathbb{Z}_2$-homology class. In Chapter 4 we consider the related problem of computing the minimum circulation in a $\mathbb{Z}$-homology class, which is intimately related to the problem of computing maximum flow on a surface embedded graph.

In the rest of this chapter we describe results [39,77] about computing minimum homologous cycles and even subgraphs. We focus on $\mathbb{Z}_2$-homology in this chapter; throughout the current chapter when we use the word homology without any prefix we mean $\mathbb{Z}_2$-homology. We start by giving a brief overview of the related results in Section 3.1. In section 3.2 we show that fast algorithms for two well-known problems in topological graph theory, namely computing the minimum cut in an undirected embedded graph and computing the minimum non-separating cycle in a directed graph, are implied by our results . In Section 3.4 we introduce the notion of homology signature, which is a vector representation of homology classes, and use it to design a linear time algorithm for testing homology. Homology signatures prove to be further helpful in Section 3.5 when we introduce the concept of $\mathbb{Z}_2$-homology cover and describe a $2^{O(g)}n\log n$ time algorithm to compute the minimum element of a $\mathbb{Z}_2$-homology class, for an alternative algorithm that runs in $g^{O(g)}n\log\log n$ see our paper [39]. Finally, in Section 3.6, we show that fixed parameter tractable algorithms are the best we can hope for by proving that computing the minimum $\mathbb{Z}_2$-homologous subgraph is in general NP-hard.

3.1 Related results

In this section, we review previously known results; we refer the interested reader to two survey papers by Colin de Verdière and Erickson [45, 75].

3.1.1 Equivalent cycles

Testing whether two subgraphs are equivalent (homotopic or homologous) is a fundamental problem of computational topology. Dehn's algorithm [54] to test whether two loops are homotopic or two cycles are freely homotopic is one of the oldest algorithms in this category. A simpler and linear time algorithm has been

developed to solve the same decision problem as a result of a series of works [59, 80, 131]. In Section 3.4 we describe the first linear time algorithms [77] to test whether two cycles are $\mathbb{Z}_2$-homologous.

Several authors have considered the related question of finding the shortest cycle in a surface graph that is either homotopic to a given cycle. Colin de Verdière and Erickson [46] describe an algorithm to compute the shortest cycle homotopic to a given cycle in a combinatorial surface in $O(gnk \log nk)$ time, where k is the number of edges in the input cycle, following a series of work that consider different versions of this problem [10, 30, 48, 49, 70, 110].

An argument of Chambers *et al.* [35] implies that finding the shortest cycle (either simple or not) in a given $\mathbb{Z}_2$-*homology* class in a surface graph is NP-hard; Chen and Friedman [42, 43] proved that the corresponding problem in simplicial complexes is NP-hard to approximate within any constant factor.

In Section 3.6 we give a similar proof to Chambers et al. [39] for the NP-hardness of computing the minimum element of a $\mathbb{Z}_2$-homology class. In Section 3.5 we present an algorithm to find the shortest $\mathbb{Z}_2$-homologous cycles in $2^{O(\beta)}n \log n$ and $g^{O(\beta)}n \log\log n$ time. In Chapter 4 we describe a result by Chambers *et al.* [38] to find the minimum-cost *circulation* in a given real or integer homology class in a directed surface-embedded graph in polynomial time; Dey *et al.* [60] generalized this result to arbitrary chains of arbitrary dimension in arbitrary simplicial complexes.

3.1.2 Shortest non-trivial cycles

The problem of finding shortest topologically nontrivial cycles in embedded *undirected* graphs has a long history. Itai and Shiloach [113] observed that the minimum (s,t)-cut in an undirected planar graph G is dual to the minimum-cost cycle that separates faces s^* and t^* in the dual graph G^*. Thus, Frederickson's minimum cut algorithm [88] computes the shortest nontrivial cycle in a combinatorial annulus in $O(n \log n)$ time. Thomassen [183] developed the first efficient algorithm for graphs on arbitrary surfaces, which runs in $O(n^3)$ time and exploits the so-called *3-path condition*; see also Mohar and Thomassen [143, Sect. 4.3]. Erickson and Har-Peled described a faster algorithm that runs in $O(n^2 \log n)$ time [76]. This is the fastest algorithm known for arbitrary surface-embedded graphs; however, several faster algorithms are known when the genus g of the underlying surface is small [26, 27, 31, 128].

The history for *directed* embedded graphs is much shorter, in part because neither Thomassen's 3-path condition nor Cabello and Mohar's crossing condition hold. The shortest nontrivial *directed* cycle in an annular graph is dual to either the minimum (s,t)-cut or the minimum (t,s)-cut in the directed planar dual graph, whichever has smaller capacity. Both of these cuts can be computed in $O(n \log n)$ time using planar flow algorithms. It appears that Jeniga and Koubeck's algorithm [117] always correctly computes the smaller of these two cuts. Cabello *et al.* [28] describe an algorithm to find a shortest non-contractible and non-separating cycle in a directed

surface graph in $O(n^2 \log n)$ time and $O(\sqrt{g} n^{3/2} \log n)$ time, respectively, using a subtle generalization of Thomassen's 3-path condition. Erickson and Nayyeri [77] found a $2^{O(g)} n \log n$ time algorithm to compute the shortest non-separating cycle; see Section 3.5. Later Erickson [74] further improved the best known running times by finding a $O(g^2 n \log n)$ time algorithm to compute the shortest non-separating cycle and a $O(g^{O(g)} n \log n)$ time algorithm to compute the shortest non-contractible cycle. Very recently, Fox [85] gave a $O(g^3 n \log n)$ time algorithm to compute the shortest non-contractible cycle.

3.1.3 Flows in sparse graphs

Euler's formula implies that an n-vertex graph embedded on a surface of genus $O(n)$ has at most $O(n)$ edges. The fastest known combinatorial maximum-flow algorithms for sparse graphs, due to Sleator and Tarjan [177] and Goldberg and Tarjan [95], run in time $O(n^2 \log n)$. The *minimum-cost* maximum flow can be computed in $O(n^2 \log^2 n)$ time using an algorithm of Orlin [145]. (For graphs with small separators, the running time of Orlin's algorithm can be improved to $O(n^2 \log n)$ by replacing Dijkstra's algorithm with a linear-time shortest-path algorithm [109, 182].) The fastest algorithm known for integer capacities, due to Goldberg and Rao [94], runs in $O(n^{3/2} \log n \log U)$ time, where U is an upper bound on the edge capacities. A more recent algorithm of Diatch and Spielman [64] computes the minimum-cost maximum flow in $O(n^{3/2} \operatorname{polylog} n \log U)$ time.

3.1.4 Flows and cuts in planar graphs

Maximum flows in planar graphs have received considerable attention for more than 50 years. Weihe [189] and Borradaile and Klein [14, 16] describe the history of planar flow algorithms in detail; we describe only a few important highlights.

Itai and Shiloach exploited the connection between maximum flows in an undirected planar graph and shortest paths in its dual graph to obtain an $O(n \log n)$-time algorithm when the source and sink vertices lie on a common face [113]; see also Hassin [105].

Reif [156] developed a divide-and-conquer algorithm to compute a minimum cut, and thus the maximum flow *value*, in a planar undirected network in $O(n \log^2 n)$ time. Reif's algorithm was extended by Hassin and Johnson to compute the actual maximum flow in $O(n \log n)$ additional time, using a carefully structured dual shortest-path computation [106]. Frederickson subsequently improved Reif's algorithm to $O(n \log n)$ time [88]. Frederickson's improvement can also be obtained more directly using more recent planar shortest-path algorithms [27, 109, 125, 182]. The same improvement can also be obtained using more recent multiple-source shortest path algorithms by Klein [125] and Cabello and Chambers [27]. Very recently, after almost 25 years without progress, Italiano *et al.* [114] described an improved algorithm that runs in $O(n \log \log n)$ time.

Maximum flows in *directed* planar graphs were first investigated by Johnson and Venkatesan [118], who described a divide-and-conquer algorithm, based on recursive separator decompositions, with running time $O(n^{3/2} \log n)$. Venkatesan [186] observed that a feasible flow with a given *value*, if such a flow exists, can be computed in $O(n^{3/2})$ time by computing a single-source shortest path tree in a dual graph with both positive and negative edge weights, using an algorithm of Lipton, Rose, and Tarjan [134]. (Venkatesan's reduction is described in greater detail in Section 4.2.2.) For graphs with integer capacities, binary search over the possible flow values immediately yields a max-flow algorithm that runs in $O(n^{3/2} \log C)$ time, where C is the sum of the capacities. This running time can be improved by more recent planar shortest path algorithms [81, 109, 126]; in particular, the recent algorithm of Mozes and Wulff-Nilsen [144] implies a running time of $O(n \log^2 n \log C / \log\log n)$. Miller and Naor [139] generalized Johnson and Venkatesan's algorithm to planar (single-commodity) flow networks with multiple sources and sinks. Returning to the classical augmenting path technique, Weihe [188, 189] described a planar maximum-flow algorithm that runs in $O(n \log n)$ time, provided the input graph satisfies a certain connectivity condition. Finally, Borradaile and Klein [14, 16] described the first $O(n \log n)$-time algorithm to find maximum flows in arbitrary directed planar graphs. Erickson [73] simplified the presentation and analysis of Borradaile and Klein's algorithm by reformulating it in terms of parametric shortest paths.

This $O(n \log n)$ time algorithm was the fastest known for undirected planar minimum cut for more than a decade until, recently, Italiano et al. [114] found a $O(n \log\log n)$ time algorithm.

3.1.5 Generalizations of planar cuts

Surprisingly little is known about the complexity of computing maximum flows or minimum cuts in generalizations of planar graphs. In particular, we know of no algorithm to compute minimum cuts in non-planar graphs that does not first compute a maximum flow prior to our results.

By combining a technique of Miller and Naor [139] with the planar directed flow algorithm of Borradaile and Klein [14–16], one can compute maximum (single-commodity) flows in a planar graph with k sources and sinks in $O(k^2 n \log n)$ time. A recent algorithm of Hochstein and Weihe [111] computes a maximum flow in a planar graph with k additional edges in $O(k^3 n \log n)$ time, using a clever simulation of Goldberg and Tarjan's push-relabel algorithm [95]. Recently, Borradaile *et al.* [17] found a $O(n \log^3)$ time algorithm to compute maximum flow with multiple sources and sinks in directed planar graphs. In a different work, Lacki *et al.* [129] describe an algorithm to compute the value of all $(n-1)$ flows from a given source to each other sink.

To our knowledge, the only prior max-flow algorithm that applies to graphs of positive genus, but not to arbitrary sparse graphs, is an algorithm of Imai and

Iwano [112] that computes minimum-cost flows in graphs with small balanced separators, using a combination of nested dissection [134, 147], interior-point methods [185], and fast matrix multiplication. Their algorithm can be adapted to compute maximum flows (and therefore minimum cuts) in any graph of constant genus in time $O(n^{1.595} \log C)$, where C is the sum of the capacities. However, this is slower than more recent and more general algorithms [94].

Euler's formula implies that a simple n-vertex graph embedded on a surface of genus $O(n)$ has at most $O(n)$ edges. The fastest known combinatorial maximum-flow algorithms for sparse graphs, due to Sleator and Tarjan [177] and Goldberg and Tarjan [95], run in time $O(n^2 \log n)$. The fastest algorithm known for integer capacities, due to Goldberg and Rao [94], runs in time $O(n^{3/2} \log n \log U)$, where U is an upper bound on the edge capacities. These are also the fastest algorithms previously known for computing maximum flows or minimum cuts in graphs of any positive genus.

For further background on maximum flows, minimum cuts, and related problems, we refer the reader to monographs by Ahuja *et al.* [5] and Schrijver [170].

3.2 Undirected minimum cut and directed non-separating cycle

Before we describe our algorithm, we first show that the minimum-weight homologous subgraph problem includes (the combinatorial dual of) the classical minimum-cut problem as a special case. Thus, our results immediately imply fast algorithm for computing minimum cuts on surfaces with small genus; see Corollary 3.5.12.

Lemma 3.2.1. *Let $G = (V, E)$ be an edge-weighted graph embedded on a surface Σ without boundary, and let s and t be vertices of G. Finally, let X be the minimum-weight (s, t)-cut in G. Then X^* is the minimum-weight even subgraph of G^* homologous with the boundary of s^* in the surface $\Sigma \setminus (s^* \cup t^*)$.*

Proof: Let ∂s^* denote the boundary of s^*, and let Σ' denote the surface $\Sigma \setminus (s^* \cup t^*)$.

Let X be an arbitrary (s, t)-cut in G. This cut partitions the vertices of G into two disjoint subsets, S and T, respectively containing vertices s and t. Thus, the dual subgraph X^* partitions the faces of G^* into two disjoint subsets, S^* and T^*, respectively containing faces s^* and t^*. In particular, X^* is the boundary of the union of the faces in S^*, which implies that X^* is null-homologous in Σ. The subgraph $X^* \oplus \partial s^*$ is the boundary of the union of faces in $S^* \setminus \{s^*\}$, which is a subset of the faces of Σ'. Thus, $X^* \oplus \partial s^*$ is null-homologous in Σ'. We conclude that X^* and ∂s^* are homologous in Σ'.

Conversely, let X^* be an arbitrary even subgraph of G^* homologous to ∂s^* in Σ'. The subgraph $X^* \oplus \partial s^*$ is null-homologous in Σ'. This immediately implies that X^* is null-homologous in Σ; moreover, faces s^* and t^* are on opposite sides of X^*. Any

path from s to t in the original graph G must traverse at least one edge of X. We conclude that X is an (s,t)-cut. □

In addition, computing the minimum homologous cycle in every $\mathbb{Z}_2$-homology classes generalize the problem of computing shortest non-separating cycle in a *directed* embedded graph, see Corollary 3.5.8. In fact, the shortest non-separating cycle is the shortest cycle whose $\mathbb{Z}_2$-homology class is not trivial.

3.3 Forest-cotree construction and greedy system of arcs

For the rest of the chapter, fix a directed graph $G = (V, E)$ cellularly embedded on a surface Σ of genus g with b boundaries. Without loss of generality, we assume that the underlying surface Σ has at least one boundary; otherwise, we can remove an arbitrary face of G from Σ without affecting its homology at all. Let $\delta_1, \ldots, \delta_b$ denote the boundary cycles of Σ, and let $\beta = 2g + b - 1$ denote the the first Betti number of Σ.

Cutting a surface embedded graph into a disc is an algorithmic problem in computational topology, with numerous applications. Here, we discuss a natural generalization of tree-cotree decompositions [72] to surfaces with boundary as a tool to compute a set of paths to cut the graph into a topological disc. We refer the interested reader to our paper [77] to see the other possible generalization.

A ***forest-cotree decomposition*** of G is any partition $(\partial G, F, C, X)$ of the edges of G into *four* edge-disjoint subgraphs with the following properties:

- ∂G is the set of all boundary edges of G.
- F is a spanning forest of G, that is, an acyclic subgraph of G that contains every vertex.
- Each component of F contains a single boundary vertex.
- C^* is a spanning tree of $G^* \setminus (\partial G)^*$, that is, a subtree of G^* that contains every vertex *except* the dual boundary vertices δ_i^*.

Euler's formula implies that there are exactly β edges in X; arbitrarily label these edges $e_1, e_2, \ldots, e_\beta$. For each edge $e_i \in X$, the subgraph $F \cup \{e_i\}$ contains a single nontrivial arc α_i, which is either a simple path between distinct boundary cycles, or a nontrivial loop from a boundary cycle back to itself; in the second case, α_i may traverse some edges of F twice. Cutting along the arcs $\alpha_1, \ldots, \alpha_\beta$ transforms Σ into a topological disk. Thus, every non-null-homologous cycle in G must cross at least one arc α_i. See Figure 3.1.

Lemma 3.3.1. *A forest-cotree decomposition can be computed in O(n) time.*

Proof: First construct a graph H by identifying all boundary vertices in G to a single vertex. Compute a spanning tree of H by whatever-first search; the edges of this

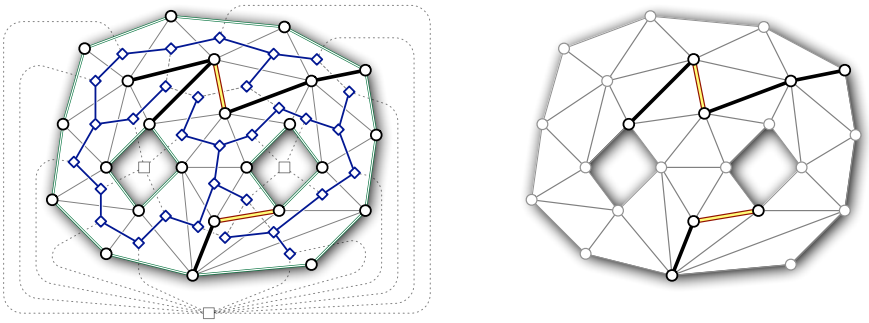


Figure 3.1. Left: A forest-cotree decomposition of a graph; thick doubled lines indicate edges in X. Right: The resulting system of arcs.

spanning tree define an appropriate spanning forest F. Construct the dual subgraph $G^* \setminus F^*$ and compute a dual spanning tree C^* via whatever-first search. Finally, let $X = G \setminus (C \cup F)$. □

Using the forest-cotree decomposition we obtain a set of arcs $\{\alpha_1, \ldots, \alpha_\beta\}$ to cut the surface into a topological disc in linear time. In the following lemma we compute a particular system of arcs with the property that each composing arc is composed of two shortest paths. Following Chambers *et al.* [36] we call such a system of arcs a *greedy system of arcs*. The concept of *greedy system of loops* is closely related [79].

Lemma 3.3.2. *A greedy system of arcs $P = \{p_1, p_2, \ldots, p_\beta\}$ can be computed in linear time.*

Proof: Let $G/\partial G$ denote the graph obtained from G by *contracting* the entire subgraph ∂G—both vertices and edges—to a single vertex x. Using Dijkstra's algorithm, we compute the single-source shortest-path tree T in $G/\partial G$ rooted at x in $O(n + g + b)$ time [109, 182]. Let F be the subgraph of G corresponding to T. Each component of F is a tree of shortest paths from a boundary vertex to a subset of the non-boundary vertices of G. With the shortest-path forest F in hand, we can easily construct the rest of the forest-cotree decomposition in $O(n)$ time.

Finally, for each edge $e_i \in X$, let σ_i and τ_i denote the unique directed paths in F from the boundary of G to the endpoints of e_i, and let $S := \{\sigma_1, \ldots, \sigma_\beta, \tau_1, \ldots, \tau_\beta\}$. By construction of F, every element of S is a (possibly empty) shortest directed path. Moreover, because $p_i = \sigma_i \cdot e_i \cdot rev(\tau_i)$ for each index i, every non-null-homologous cycle in G must intersect at least one path in S. We can easily compute each path in S in $O(n)$ time. □

3.4 Homology signatures and homology test

Recall that the space of homology classes of cycles on a surface is a vector space. It follows that we should be able to present each homology class with a vector. In this section we introduce the notion of homology signature for cycles, which is

essentially a vector that presents the homology class of cycles according to a certain bases. We describe how to build the signatures for $\mathbb{Z}_2$-homology and $\mathbb{Z}$-homology, however they can be easily generalize to cover any $\mathbb{Z}_k$-homology classes.

We define α_i^+ to be the arc obtained by extending α_i to the boundaries by adding vertices of degree 1 to its endpoints. Let $\alpha_i^+ = (w_0, w_1, \cdots, w_k, w_{k+1})$. We say that a dart $\vec{e} = u \rightarrow v$ enters α_i from right if and only if $v = w_j$ for $1 \leq j \leq k$, $u \notin \alpha_i$ and (w_{j-1}, u, w_{j+1}) are in counterclockwise order around $w_j = v$. We say that a dart $\vec{e} = u \rightarrow v$ leaves α_i from right if and only if $u = w_j$ for $1 \leq j \leq k$, $v \notin \alpha_i$ and (w_{j-1}, v, w_{j+1}) are in counterclockwise order around $w_j = u$. For each dart $\vec{e} = u \rightarrow v$ in G, we define its ***integer signature*** $[\vec{e}]$ to be the vector of β integers whose ith integer is equal to 1 if and only if $\vec{e}$ enters α_i from right side, it is -1 if it leaves α_i from right side, and it is 0 otherwise. Then, the signature of a circulation is the weighted vector sum of the integer signatures of its edges. The ***binary signature*** of a an edge e, denoted by $[e]_2$, or an even subgraph η, denoted by $[\eta]_2$, are their integer signature modulo 2.

Let $\boldsymbol{h \oplus h'}$ denote the vector sum of two integer homology signatures h and h', and $\boldsymbol{b \oplus_2 b'}$ is the bitwise exclusive-or (vector sum modulo 2) of the binary signatures b and b'. For circulations η and η', the identity $[\eta \oplus \eta'] = [\eta] \oplus [\eta']$ follow directly from the definitions. In particular, we have $[\eta \oplus_2 \eta']_2 = [\eta]_2 \oplus_2 [\eta']_2$ note that a circulation under $\mathbb{Z}_2$-homology is essentially and even subgraph. When it is clear from the context that we are working with $\mathbb{Z}_2$-homology we drop the subscripts and write $[\cdot]$ instead of $[\cdot]_2$ and $\oplus$ instead of $\oplus_2$.

Lemma 3.4.1. *G can be preprocessed in $O(\beta n)$ time, so that the integer signature of any circulation can be computed in $O(\beta)$ time per edge.*

Proof: A forest-cotree decomposition can be computed in $O(n)$ time using the algorithm of Lemma 3.3.1. With the decomposition in hand, it is straightforward to compute each path α_i in $O(n)$ time, and then compute each edge signature in $O(\beta)$ time. □

Corollary 3.4.2. *We can preprocess G in $O(\beta n)$ time, so that the binary signature of any even subgraph can be computed in $O(\beta)$ time per edge.*

3.4.1 $\mathbb{Z}$-homology

In this subsection we describe an algorithm to test whether two circulations are $\mathbb{Z}$-homologous. This is the only subsection of this chapter, in which we talk about $\mathbb{Z}$-homology. We start with the following structural lemma about integer circulations.

Lemma 3.4.3. *Any circulation η with integer coefficients can be decomposed to a collection of directed simple cycles.*

Proof: Let v be a vertex of G that has at least one outgoing dart with a positive coefficient in η. Our algorithm starts a walk from v, whenever it enters an unvisited

vertex u, it leaves u through an unvisited outgoing dart with positive coefficient. Since η is a circulation the existence of such an outgoing dart is guaranteed upon the first entrance to any vertex. As soon as, the algorithm enters an already visited vertex, it finds a cycle in η and so in G; note that this cycle does not necessarily contain v. Now we can subtract this cycle from η to obtain another circulation η'. Then, the statement of the lemma follows by induction on the sum of the coefficients of η. □

Now, we use the above lemma to show that the integer homology signature captures null-homology.

Lemma 3.4.4. *A circulation η of G is in trivial $\mathbb{Z}$-homology class in Σ if and only if $[\eta] = 0$.*

Proof: Let η be a null-homologous circulation of G. Then by definition, η is the boundary of a 2-chain α. The boundary of any face f is contractible in Σ and therefore has signature 0. It follows immediately that $[\eta] = [\sum_{f\in Y} \partial\alpha(f)] = \sum_{f\in Y}[\partial\alpha(f)] = 0$.

Conversely, suppose $[\eta] = 0$. Lemma 3.4.3 implies that η can be decomposed into a set of cycles. In particular, we can define a set of crossing points between η (in fact, the collection of cycles) and any α_i. For a directed cycle γ and an arc α_i, we say that γ crosses α_i from right to left (resp. from left to right) at x_1 if and only if there is a subpath $(x_0, x_1, \ldots, x_k)$ of γ such that $x_0 \rightarrow x_1$ enters α_i from right (resp. from left), $x_1, x_2, \ldots, x_{k-1} \in \alpha_i$, and $x_{k-1} \rightarrow x_k$ leaves α_i from left (resp. from right). In the above case, we refer to the subpath $x_1, x_1, \ldots, x_{k-1}$ as the crossing interval of x_1.

Since the net number of times that η crosses α_i is 0, there exists intersection points x_1 and y_1 such that η crosses p_i from right to left and left to right at x_1 and y_1, respectively. Let $I_x = (x_1, \ldots, x_{k-1})$ be the crossing interval of x_1 and let $I_y = (y_1, y_1, \ldots, y_{l-1})$ be the crossing interval of y_1. Further let $\alpha_i[x, y]$ and $\alpha_i[y, x]$ be the directed subpaths of α_i from x to y and from y to x respectively. We change η without changing its homology class in order to reduce its total number of crossing points. We change I_x and I_y and so their containing cycles as follows. First we add the trivial cycle $\alpha_i[x_1, y_1] \cdot \alpha_i[y_1, x_1]$ to η. Then we reinterpret the connections to have a (x_0, y_l)-path $(x_0 \rightarrow x_1) \cdot \alpha_i[x_1, y_1] \cdot I_y \cdot (y_{l-1} \rightarrow y_l)$ and a (y_0, x_k)-path $(y_0 \rightarrow y_1) \cdot \alpha_i[y_1, x_1] \cdot I_x \cdot (x_{k-1} \rightarrow x_k)$.

It follows by induction that η is homologous to another circulations (or collection of cycles) η' that does not cross any path α_i at all, and so it is null-homologous. □

The following corollary is now immediate.

Corollary 3.4.5. *Two circulations η and η' of G are $\mathbb{Z}$-homologous in Σ if and only if $[\eta] = [\eta']$.*

Now, we can conclude this section by the following theorem, which gives an algorithm for $\mathbb{Z}$-homology test.

Theorem 3.4.6. *Let $G = (V, E)$ be a graph with n vertices embedded on a surface Σ of genus g with b boundaries. Then, after preprocessing G in $O((g+b)n)$ time, for any two circulations η_1 and η_2, we can test whether η_1 and η_2 are $\mathbb{Z}$-homologous in $O((g+b)(|\eta_1|+|\eta_2|))$ time, where $|\eta|$ denotes the number of vertices of the even subgraph η.*

3.4.2 $\mathbb{Z}_2$-homology

Now, we show how to compute with the $\mathbb{Z}_2$-homology class. Some results of this subsection are used later in building the $\mathbb{Z}_2$-homology cover, which will be used to compute the minimum elements of $\mathbb{Z}_2$ homology classes.

Lemma 3.4.7. *An even subgraph η of G is in trivial $\mathbb{Z}_2$-homology class in Σ if and only if $[\eta]_2 = 0$.*

Proof: The proof is exactly similar to the proof of Lemma 3.4.4, however, we should note that crossing from right to left and from left to right are not distinguishable when we work under $\mathbb{Z}_2$-homology. As a result we can pair any two crossing points, so we can reduce the total number of crossings as long as there are at least two crossings on any arc α_i. It follows that we can reduce the number of crossings to 0 if $[\eta]_2 = 0$. □

The following corollaries are now immediate.

Corollary 3.4.8. *Two even subgraphs η and η' of G are $\mathbb{Z}_2$-homologous in Σ if and only if $[\eta]_2 = [\eta']_2$.*

Corollary 3.4.9. *Two cycles γ and γ' in G are $\mathbb{Z}_2$-homologous in Σ if and only if $[\gamma]_2 = [\gamma']_2$.*

The algorithm for testing whether two even subgraphs are $\mathbb{Z}_2$-homologous follows immediately.

Theorem 3.4.10. *Let $G = (V, E)$ be a graph with n vertices embedded on a surface Σ of genus g with b boundaries. Then, after preprocessing G in $O((g+b)n)$ time, for any two even subgraphs η_1 and η_2, we can test whether η_1 and η_2 are $\mathbb{Z}_2$-homologous in $O((g+b)(|\eta_1|+|\eta_2|))$ time, where $|\eta|$ denotes the number of vertices of the even subgraph η.*

3.5 Minimum homologous subgraph and the $\mathbb{Z}_2$-homology cover

3.5.1 The $\mathbb{Z}_2$-homology cover

In the remaining of this chapter we mainly work with $\mathbb{Z}_2$-homology classes, so we frequently use the word homology instead of $\mathbb{Z}_2$-homology, $[\cdot]$ instead of $[\cdot]_2$ and $\oplus$ instead of $\oplus_2$.

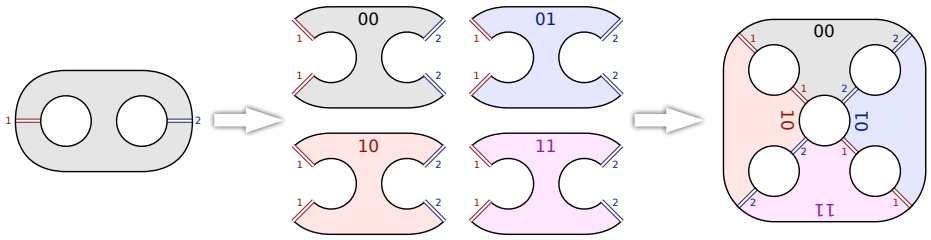


Figure 3.2. Constructing the $\mathbb{Z}_2$-homology cover of a pair of pants (a genus zero surface with three boundaries).

With the homology signatures in hand, the $\mathbb{Z}_2$-homology cover of a combinatorial surface can be defined using a standard *voltage construction* [100, Chapter 4], as follows. Let $\overline{G}$ denote the graph whose vertices are all ordered pairs (v,h) where v is a vertex of G and h is an element of $(\mathbb{Z}_2)^\beta$, and whose edges are the ordered pairs $(u{\to}v,h) := (u,h){\to}(v,h \oplus [u{\to}v])$ for all edges $u{\to}v$ of G and all homology classes $h \in (\mathbb{Z}_2)^\beta$, where $\beta = 2g + b - 1$ is the first betti number. Let $\pi\colon \overline{G} \to G$ denote the covering map $\pi(v,h) = v$; this map projects any cycle in $\overline{G}$ to a cycle in G. To define a cellular embedding of $\overline{G}$, we declare a cycle in $\overline{G}$ to be a face if and only if its projection is a face of G. The combinatorial surface defined by this embedding is the $\mathbb{Z}_2$-homology cover $\overline{\Sigma}$.

Our construction can be interpreted more topologically as follows. Let $\alpha_1, \ldots, \alpha_\beta$ denote the system of arcs used to define the homology signatures $[e]$. The surface $D := \Sigma \setminus (\alpha_1 \cup \cdots \cup \alpha_\beta)$ is a topological disk. Each arc α_i appears on the boundary of D as two segments α_i^+ and α_i^-. For each signature $h \in (\mathbb{Z}_2)^\beta$, we create a disjoint copy (D,h) of D; for each index i, let (α_i^+,h) and (α_i^-,h) denote the copies of α_i^+ and α_i^- in the disk (D,h). For each index i, let b_i denote the β-bit vector whose ith bit is equal 1 and whose other $\beta - 1$ bits are all equal to 0. The $\mathbb{Z}_2$-homology cover $\overline{\Sigma}$ is constructed by gluing the 2^β copies of D together by identifying boundary paths (α_i^+,h) and $(\alpha_i^-,h \oplus b_i)$, for every index i and homology class h. See Figure 3.2 for an example.

Lemma 3.5.1. *The combinatorial surface $\overline{\Sigma}$ has $\overline{n} = 2^\beta n$ vertices, genus $\overline{g} = O(2^\beta \beta)$, and $\overline{b} = O(2^\beta b)$ boundaries, and it can be constructed in $O(2^\beta n)$ time.*

Proof: Let m and f denote the number of edges and faces of Σ, respectively. Recall that the Euler characteristic of Σ is $\chi = n - m + f = 2 - 2g - b = 1 - \beta$. The combinatorial surface $\overline{\Sigma}$ has exactly $\overline{n} = 2^\beta n$ vertices, $2^\beta m$ edges, and $2^\beta f$ faces, so its Euler characteristic is $\overline{\chi} = 2^\beta(1 - \beta)$.

If $b > 1$, then each boundary cycle δ_i has a non-zero homology signature; at least one arc α_j has exactly one endpoint on δ_i. Thus, $\overline{\Sigma}$ has exactly $\overline{b} = 2^{\beta-1} b$ boundary cycles, each of which is a double-cover (in fact, the $\mathbb{Z}_2$-homology cover) of some boundary cycle δ_i. It follows that $\overline{\Sigma}$ has genus $\overline{g} = 1 - (\overline{\chi} + \overline{b})/2 = 2^{\beta-2}(4g + b - 4) + 1$. (Somewhat surprisingly, $\overline{\Sigma}$ may have positive genus even when Σ does not!) On the other hand, when $b = 1$, the boundary cycle δ_1 is null-homologous, so $\overline{\Sigma}$ has $\overline{b} = 2^\beta b$ boundary cycles, and thus $\overline{\Sigma}$ has genus $\overline{g} =$

$1-(\overline{\chi}+\overline{b})/2 = 2^{\beta}(g-1)+1$.

After computing the homology signatures of the edges of Σ in $O(\beta n)$ time, following the definition, it is straightforward to construct $\overline{\Sigma}$ in $O(\overline{n}) = O(2^{\beta}n)$ time. □

If the input graph G is weighted we can compute weights on the edges of the covering space as follows. We assign weights to the directed edges of $\overline{G}$ by setting $\overline{w}(u{\to}v, h) := w(u{\to}v)$ for each edge $u{\to}v$ of G and each homology class h. In other words, each directed edge in $\overline{\Sigma}$ inherits the weight of its projection in Σ.

Now consider an arbitrary path p in G, with (possibly equal) endpoints u and v. A straightforward induction argument implies that for any homology class $h \in (\mathbb{Z}_2)^{\beta}$, the path p lifts to a unique path from (u,h) to $(v, h \oplus [p])$, which we denote $\boldsymbol{(p,h)}$. Moreover, this lifted path has the same length as its projection: $w(p) = \overline{w}(p,h)$. The following lemmas are now immediate.

Lemma 3.5.2. *Every lift of a shortest directed path in G is a shortest directed path in $\overline{G}$.*

We say that a loop ℓ in G with basepoint v is *tight* in its homology class if and only if there is no shorter loop with basepoint v in the same homology class.

Lemma 3.5.3. *A loop ℓ in G with basepoint v is tight its $\mathbb{Z}_2$-homology class if and only if, for some (in fact, every) homology class $h \in (\mathbb{Z}_2)^{\beta}$, the lifted path (ℓ, h) is a shortest directed path in $\overline{G}$ from (v,h) to $(v, h \oplus [\ell])$.*

3.5.2 Computing $\mathbb{Z}_2$-minimal cycles

We now describe our algorithm to compute the shortest directed cycle in a given $\mathbb{Z}_2$-homology class.

Lemma 3.5.4. *In linear time, we can construct a set S of $O(\beta)$ directed shortest paths in G, such that every non-null-homologous cycle in G intersects at least one path in S.*

Proof: Any greedy system of arcs can be decomposed into 2β shortest paths that collectively have the property of the lemma; so the lemma statement is an immediate implication of Lemma 3.3.2. □

Recall that any path σ from u to v in G lifts to a unique path $(\sigma, 0)$ from $(u,0)$ to $(v, [\sigma])$ in $\overline{G}$.

Lemma 3.5.5. *Let γ be a $\mathbb{Z}_2$-minimal cycle in G, and let σ be any shortest path in G that intersects γ. There is a $\mathbb{Z}_2$-minimal cycle γ' homologous to γ, that lifts to a shortest path (γ', h) in $\overline{G}$ that starts with a subpath of $(\sigma, 0)$ but does not otherwise intersect $(\sigma, 0)$.*

Proof: Let v be the vertex of $\sigma \cap \gamma$ closest to the starting vertex of σ, and let (v,h) be the corresponding vertex of the lifted path $(\sigma,0)$. Think of γ as a loop based at v. Lemma 3.5.3 implies that the lifted path (γ,h) is a shortest path from (v,h) to $(v,h \oplus [\gamma])$.

Now let (w,h') be the last vertex along (γ,h) that is also a vertex of $(\sigma,0)$. Let (γ',h) be the path obtained from (γ,h) by replacing the subpath from from (v,h) to (w,h') with the corresponding subpath of $(\sigma,0)$. By construction, (γ',h) starts with a directed subpath of $(\sigma,0)$ but does not otherwise intersect $(\sigma,0)$. Because both (γ,h) and $(\sigma,0)$ are shortest paths in $\overline{\Sigma}$, the new path (γ',h) has the same length as (γ,h). Thus, the projected cycle γ' has the same length and homology class as γ, which implies that γ' is $\mathbb{Z}_2$-minimal. □

We emphasize that the modified cycle γ' may intersect σ arbitrarily many times; however, all such intersections lift to intersections between (γ',h) and lifts of σ other than $(\sigma,0)$. In general, two directed shortest path may cross each other arbitrary many times in directed graphs, where the weight of edges are asymmetric. In undirected graphs and under the assumption of uniqueness of shortest paths, it is easy to show that no pairs of shortest paths cross more than once.

Our algorithm uses a recent generalization of Klein's seminal multiple-source shortest path algorithm [125] to higher-genus embedded graphs:

Lemma 3.5.6 (Chambers *et al.* [33]). *Let G be a directed graph with non-negative edge weights, cellularly embedded on a surface Σ of genus g with $b > 0$ boundaries, and let f be an arbitrary face of G. We can preprocess G in $O(gn \log n)$ time, and $O(n)$ space, so that the length of the shortest path from any vertex incident to f to any other vertex can be retrieved in $O(\log n)$ time.*

Theorem 3.5.7. *Let G be a directed graph with non-negative edge weights, cellularly embedded on a surface Σ with first Betti number β, and let γ be a cycle in G with k edges. A shortest directed cycle in Σ that is $\mathbb{Z}_2$-homologous with γ can be computed in $O(\beta k + 2^{\beta} n \log n)$ time.*

Proof: We begin by computing homology signatures for the edges of G in $O(\beta n)$ time, as described in Section 3.4. In $O(\beta k)$ time, we then compute the homology signature $[\gamma]$. If $[\gamma] = 0$, we return the empty walk and halt.

Next, we construct the $\mathbb{Z}_2$-homology cover $\overline{G}$ in $2^{O(\beta)} n \log n$ time, as described in Section 3.5.1, as well as the set S of directed shortest paths described in Lemma 3.5.4. We then look for the shortest path in $\overline{G}$ of the canonical form described in Lemma 3.5.5, by considering each shortest path $\sigma \in S$ in turn as follows.

Let us write $(\sigma,0) = (v_0,0) \to (v_1,h_1) \to \cdots \to (v_t,h_t)$. We construct the combinatorial surface $\overline{\Sigma} \not\circ (\sigma,0)$ by splitting the path $(\sigma,0)$ into two parallel paths from $(v_0,0)$ to (v_t,h_t), which we denote $(\sigma,0)^+$ and $(\sigma,0)^-$. For each index $1 \le i \le t-1$, let $(v_i,h_i)^+$ and $(v_i,h_i)^-$ denote the copies of vertex (v_i,h_i) on the paths $(\sigma,0)^+$

and $(\sigma,0)^-$, respectively. The paths $(\sigma,0)^+$ and $(\sigma,0)^-$ bound a new common face $f_{(\sigma,0)}$ in $\overline{\Sigma}\not\sim(\sigma,0)$.

Lemma 3.5.5 implies that if any $\mathbb{Z}_2$-minimal cycle homologous to γ intersects σ, then some $\mathbb{Z}_2$-minimal cycle homologous to γ is the projection of a shortest path in $\overline{\Sigma}\not\sim(\sigma,0)$ from some vertex $(v_i,h_i)^\pm$ to the corresponding vertex $(v_i,h_i\oplus[\gamma])$. To compute these shortest paths, we implicitly compute the shortest path in $\overline{\Sigma}\not\sim(\sigma,0)$ from every vertex on the boundary of $f_{(\sigma,0)}$ to *every* vertex of $\overline{\Sigma}\not\sim(\sigma,0)$, using Lemma 3.5.6. The resulting algorithm runs in $O(\overline{g}\,\overline{n}\log\overline{n}) = O(4^\beta\beta^2\,n\log n)$ time, by Lemma 3.5.1. □

By running this algorithm 2^β times, we can compute the shortest directed cycle in Σ in *every* $\mathbb{Z}_2$-homology class, in $2^{O(\beta)}n\log n$ time. In particular, we can compute the shortest directed cycle in Σ that has nontrivial $\mathbb{Z}_2$-homology. When the original surface has no boundary, this is just the shortest *non-separating* cycle in Σ. Very recently, Cabello *et al.* [28] described an algorithm to compute the shortest non-separating cycle in any surface-embedded directed graph in $O(g^{1/2}n^{3/2}\log n)$ time; our new algorithm is faster whenever $g \le (\lg n)/13$.

Corollary 3.5.8. *Given a directed graph G with n vertices with non-negative edge weights, cellularly embedded on a surface with genus g, we can compute the shortest directed cycle in G that is non-separating in Σ in $2^{O(g)}n\log n$ time.*

3.5.3 Computing $\mathbb{Z}_2$-minimal even subgraphs

In this section, we describe an algorithm to compute the minimum-weight even subgraph in *every* $\mathbb{Z}_2$-homology class, using the algorithm of the previous section. Theorem 3.5.7 immediately implies that we can compute a minimum-weight *cycle* in every $\mathbb{Z}_2$-homology class in $2^{O(\beta)}n\log n$ time. However, the minimum weight *even subgraph* in a given homology class may not be a $\mathbb{Z}_2$-minimal cycle. In particular, if a $\mathbb{Z}_2$-minimal cycle γ traverses any edge more than once, then every minimum-weight even subgraph with signature $[\gamma]$ *must* be disconnected.

Call a cycle in G *weakly simple* if it traverses each edge of G at most once and never crosses itself. Any weakly simple cycle in G can be perturbed into a simple cycle in an arbitrarily small neighborhood of G. A ***cycle decomposition*** of an even subgraph η is a set of edge-disjoint, non-crossing, weakly simple cycles whose union is η.

Lemma 3.5.9. *Every even subgraph of an embedded graph has a cycle decomposition.*

Proof: Let H be an even subgraph of G. We can decompose H into cycles by specifying, at each vertex v, which pairs of incident edges of H are consecutive. Any pairing that does not create a crossing at v is sufficient. For example, if $e_1,e_2,\ldots,e_{2d}$ are the edges of H incident to v, indexed in clockwise order around v, we could pair edges e_{2i-1} and e_{2i} for each i. □

The components of any $\mathbb{Z}_2$-minimal even subgraph are themselves $\mathbb{Z}_2$-minimal even subgraphs. Thus, we can assemble a $\mathbb{Z}_2$-minimal even subgraph in any homology class from a subset of the $\mathbb{Z}_2$-minimal cycles we have already computed. The following lemma puts an upper bound on the number of cycles we need.

Lemma 3.5.10. *Every $\mathbb{Z}_2$-minimal even subgraph of G has at most $g + b - 1$ components.*

Proof: Let η be an even subgraph of G with more than $g + b - 1$ components. Then, by Lemma 3.5.9, η has a cycle decomposition with at least $g + b$ elements.

Let $\gamma_1, \ldots, \gamma_{g+b}$ any subset of the cycle decomposition of η, and consider the surface $\Sigma' = \Sigma \setminus (\gamma_1 \cup \cdots \cup \gamma_{g+b})$. The definition of genus implies that Σ' cannot be connected; indeed, Σ' must have at least $b + 1$ components. So the pigeonhole principle implies that some component Σ'' of Σ does not contain any of the boundary cycles of Σ. The boundary of Σ'' is therefore null-homologous. We conclude that η is not $\mathbb{Z}_2$-minimal. □

Theorem 3.5.11. *Let G be an **undirected** graph with non-negative edge weights, cellularly embedded on a surface Σ with first Betti number β. A minimum-weight even subgraph of G in each $\mathbb{Z}_2$-homology class can be computed in $2^{O(\beta)} n \log n$ time.*

Proof: Our algorithm computes a minimum-weight cycle γ_h in every $\mathbb{Z}_2$-homology class h in $2^{O(\beta)} n \log n$) time, via Theorem 3.5.7, and then assemble these $\mathbb{Z}_2$-minimal cycles into $\mathbb{Z}_2$-minimal even subgraphs using dynamic programming.

For each homology class $h \in (\mathbb{Z}_2)^\beta$ and each integer $1 \le k \le g + b - 1$, let $\boldsymbol{C(h, k)}$ denote the minimum total weight of any set of at most k cycles in G whose homology classes sum to h. Lemma 3.5.10 implies that the minimum weight of any even subgraph in homology class h is exactly $C(h, g + b - 1)$. This function obeys the following straightforward recurrence:

$$C(h, k) = \min \left\{ C(h_1, k-1) + C(h_2, 1) \;\middle|\; h_1 \oplus h_2 = h \right\}.$$

This recurrence has two base cases: $C(0, k) = 0$ for any integer k, and for any homology class h, the value $C(h, 1)$ is just the length of γ_h. A standard dynamic programming algorithm computes $C(h, g + b - 1)$ for all 2^β homology classes h in $O(4^\beta \beta)$ time. We can then assemble the actual minimum-weight even subgraphs in each homology class in $O(\beta n)$ time. The total time for this phase of the algorithm is $O(4^\beta \beta + 2^\beta \beta n)$, which is dominated by the time to compute all the $\mathbb{Z}_2$-minimal cycles. □

Lemma 3.2.1 and Theorem 3.5.11 immediately give us the following corollary:

Corollary 3.5.12. *Given an undirected graph H with non-negative edge capacities, embedded on a surface Σ with genus g, and two vertices let s and t, we can compute a minimum (s, t)-cut in H in $2^{O(g)} n \log n$ time.*

3.6 NP-hardness

In this section, we show that finding the minimum-weight even subgraph in a given homology class is NP-hard, even when the underlying surface has no boundary.

Cabello *et al.* [29] show that computing the minimum separating or splitting cycles are NP-complete. Further, Chen and Freedman [43] prove that the problem of computing the minimum cycle in a given $\mathbb{Z}_2$-homology class is hard to approximate within any constant factor. Finally, Dunfield and Hirani [66] show that finding the minimum element of give $\mathbb{Z}$-homology class in complexes is NP-hard. Our proof closely follows a reduction of McCormick *et al.* [138] from MIN2SAT to a special case of MAXCUT.

Theorem 3.6.1. *Computing the minimum-cost even subgraph in a given homology class on a surface without boundary is equivalent to computing a minimum-capacity cut in an embedded edge-weighted graph G whose negative-cost edges are dual to an even subgraph in G^*.*

Proof: Fix a graph G embedded on a surface Σ without boundary, together with a cost function $c : E \to \mathbb{R}$. For any even subgraph H of G, let $c(H) = \sum_{e \in H} c(e)$, and let $\textsc{MinHom}(H, c)$ denote the even subgraph of minimum cost in the homology class of H.

Consider the *residual cost* function $c_H : E \to \mathbb{R}$ defined by setting $c_H(e) = c(e)$ for each edge $e \notin H$, and $c_H(e) = -c(e)$ for each edge $e \in H$. For any subgraph H' of G, we have $c(H') = c_H(H \oplus H') + c(H)$, which immediately implies that $\textsc{MinHom}(H, c) = H \oplus \textsc{MinHom}(\emptyset, c_H)$.

Every null-homologous even subgraph of G is dual to a cut in the dual graph G^*. Thus, we have reduced our problem to computing the minimum cut in G^* with respect to the cost function c_H. Since the empty set is a valid cut with zero cost, the cost of the minimum cut is never positive. In particular, H is the minimum-cost even subgraph in its homology class if and only if the cut in G^* with minimum residual cost is empty.

In fact, our reduction is reversible. Suppose we want to find the minimum cut in an embedded graph $G = (V, E)$ with respect to the cost function $c : E \to \mathbb{R}$, where every face of G is incident to an even number of edges with negative cost. Let $H = \{e \in E \mid c(e) < 0\}$ be the subgraph of negative-cost edges, and let X denote the (possibly empty) set of edges in the minimum cut of G. Consider the *absolute cost* function $|c| : E^* \to \mathbb{R}$ defined as $|c|(e^*) = |c(e)|$. Then $(H \oplus X)^*$ is the even subgraph of G^* of minimum absolute cost that is homologous to H^*. □

We now prove that this special case of the minimum cut problem is NP-hard, by reduction from MINCUT in graphs with negative edges. This problem includes MAXCUT as a special case (when every edge has negative cost), but many other special cases are also NP-hard [138]. The output of our reduction is a simple triangulation; the reduction can be simplified if graphs with loops and parallel edges are allowed.

Suppose we are given an *arbitrary* graph $G = (V, E)$ with n vertices and an *arbitrary* cost function $c : E \to \mathbb{R}$. We begin by computing a cellular embedding of G on some surface. If we don't care whether the surface is orientable, we can simply impose a cyclic order on the edges incident to each vertex. The maximum-genus *orientable* cellular embedding can be computed in polynomial time [89]. Alternately, we can add zero-length edges to make the graph complete and then use classical results of Ringel, Youngs, and others [157, 158] to compute a minimum-genus orientable embedding of K_n in polynomial time. Once we have an embedding, we add vertices and zero-cost edges to obtain a triangulation.

Now, we describe a procedure to further modify G to obtain another cellularly embedded graph G' (with the same vertex set), in which the dual of all negative edges is an even subgraph without changing the cost of any cut; Chambers et al. [39] describe an alternative procedure.

Observe that the dual of all negative edges is a subgraph, which, indeed, has an even number of vertices of odd degree; let $(f_1^*, f_2^*, \ldots, f_{2k}^*)$ be the set of such vertices. For each $1 \le i \le k$ we glue a handle between the faces f_{2i-1} and f_{2i} (after making a puncture in each of them) to obtain an annulus A_i whose boundary components are ∂f_{2i-1} and ∂f_{2i}. Then, to make the embedding cellular, we connect an arbitrary vertex of f_{2i-1} to and arbitrary vertex of f_{2i} through A_i with an edge of zero weight.

Clearly, all faces of G' has an even number of edges with negative weight on their boundary (equivalently, the dual of all negative edges is an even subgraph). Further, the vertex set of G and G' are equal, and G' has only some extra edges of weight zero. It follows that the cost of any cut in G and G' are equal.

Theorem 3.6.2. *Given an even subgraph H of an edge-weighted graph G embedded on a surface without boundary, computing the minimum-weight even subgraph homologous to H is strongly NP-hard.*

Finally, we emphasize that the NP-hardness of this problem relies crucially on the fact that we are using homology with coefficients taken from the finite field $\mathbb{Z}_2$. The corresponding problem for homology with real or integer coefficients is a minimum-cost circulation problem, and thus can be solved in polynomial time; see Chapter 4.

Chapter 4

Flows and $\mathbb{Z}$-homology

4.1 Introduction

Let $G = (V, E)$ be a directed graph embedded on a surface of genus g, $s, t \in V$ known as source and sink, and $c : E \to \mathbb{R}^+$ be a capacity function on edges. In this chapter we describe an algorithm to compute the maximum (s, t)-flow in G. For any fixed genus g and polynomially-bounded integer capacities, our algorithm runs in $O(n \operatorname{polylog} n)$ time (bit operations). We also describe a combinatorial algorithm that runs in $O(n^{3/2})$ time (arithmetic operations) for arbitrary real capacities, for graphs of any fixed genus.

Generalizing a relation between planar flows and shortest paths, first observed by Venkatesan [186], and using homology, we give a reduction of the flow problem to another optimization problem in a lower dimension space. More precisely, we show that the problem of finding the maximum flow is reduced to the problem of finding the homology class of the maximum flow in the flow homology space, which is a vector space of dimension $O(g)$. Since the new optimization problem involves an exponential number of constraints, we perform the optimization implicitly, using an adaptation of central ellipsoid method and multidimensional parametric search.

Following a strategy first suggested by Sullivan [181], we show that a dual formulation of our algorithm finds the minimum-cost circulation in the same homology class as a given circulation, in a graph with non-negative edge costs but no capacities, in roughly the same time as computing a maximum flow; see Section 4.3.

For an overview of related results we refer the interested reader to Section 3.1. Our algorithm to compute the maximum flow in surface embedded graphs follows in Section 4.2. Finally, we describe the dual formulation of our algorithm resulting in a result to compute the minimum circulation in a given $\mathbb{Z}$-homology class in Section 4.3.

4.2 Homology flows

Throughout this section, we fix an undirected graph $G = (V, E)$, a cellular embedding of G on an orientable surface Σ of genus g, a capacity function $c : E \to \mathbb{R}^+$, and two vertices s and t. See our paper [38] for an extension of the result to directed graphs and non-orientable surfaces.

4.2.1 Overview

Our key insight generalizes the relationship between flows and dual shortest paths in planar graphs first observed by Venkatesan [186] using a standard equivalence relation from algebraic topology called *homology*. We prove in Section 4.2.2 that given any flow f, one can find a feasible flow in the same *homology class* in near-linear time, by computing a single-source shortest path tree in the dual of the residual network G_f. Two flows are in the same homology class if their difference is the weighted sum of directed facial cycles. This observation allows us to optimize the homology class of the flow, rather than directly optimizing the flow itself. Instead of optimizing a vector of $O(n)$ flow values, our algorithm optimizes a vector of $2g+1$ homology coefficients, subject to a much larger set of linear constraints; see Section 4.2.4.

We perform this optimization implicitly using two different techniques: central cut ellipsoid method [101] and multidimensional parametric search [1,3]; see our paper [37] for more details. In order to apply the above optimization methods we required a require a membership separation oracle, which we [37] obtained by generalizing a recent algorithm of Mozes and Wulff-Nilsen [144], and a parallel shortest path algorithm, for which we use an algorithm by Cohen [44].

4.2.2 Homologous feasible flows

More than 25 years ago, Venkatesan [186] observed that for any planar graph G, a feasible (s,t)-flow with a given value can be computed, if such a flow exists, by solving a single-source shortest path problem in a dual planar graph G^* with both positive and negative edge lengths. Similar approaches were proposed by Johnson and Venkatesan [118], Hassin and Johnson [106], Khuller *et al.* [121], and Miller and Naor [139]. The following lemma directly generalizes Venkatesan's observation to flow networks of higher genus.

Let $\phi : E \to \mathbb{R}$ be an arbitrary (in particular, not necessarily feasible) (s,t)-flow in G. The ***dual residual network*** G^*_ϕ is the directed dual graph $\vec{G}^*$, where every dual dart $\vec{e}^*$ has a ***cost*** $c_\phi(\vec{e}^*)$ equal to the residual capacity of its corresponding primal dart: $c_\phi(\vec{e}^*) = c_\phi(\vec{e})$. For any directed cocycle λ, let $c(\lambda)$ denote its total capacity, and for any flow ϕ, let $\phi(\lambda)$ denote the total flow through edges in λ:

$$c(\lambda) := \sum_{\vec{e}\in\lambda} c(e) \qquad \text{and} \qquad \phi(\lambda) := \sum_{\vec{e}\in\lambda} \phi(\vec{e}).$$

Lemma 4.2.1. *A (s,t)-flow (circulation) ϕ is null-homologous if and only if for every directed cycle λ^* in G^* we have $\phi(\lambda) = 0$.*

Proof: If ϕ is null-homologous then $\phi = \partial\alpha$ there some 2-chain α. Then, for any directed cycle λ^* of G^* we have:

$$\phi(\lambda) = \sum_{f\uparrow g\in\lambda} \alpha(g) - \alpha(f) = 0$$

The last equality holds because λ^* is a cycle.

On the other hand, suppose $\phi(\lambda) = 0$ for every cocycle λ. We build the 2-chain α such that $\partial\alpha = \phi$ as follows. Let r be an arbitrary face of G and assign $\alpha(r) = 0$. Then for any other face f of G let $\alpha(f) = \phi(\gamma)$, where γ is an arbitrary (r^*, f^*)-directed path in G^*. Observe that the assumption of the lemma implies that $\phi(\gamma) = \phi(\gamma')$ for any pair of (r^*, f^*)-paths γ and γ' in G^*. Now, it is straightforward to check that $\partial\alpha = \phi$. □

The following Corollary is now immediate.

Corollary 4.2.2. *Two (s,t)-flows ϕ and ψ are homologous if and only if for any directed cycle λ^* in G^* we have $\phi(\lambda) = \psi(\lambda)$.*

Let ψ^* be a circulation in G^* (equivalently,ψ is a cocirculation in G), then we define $\phi(\psi)$, call it the amount of flow that ϕ sends through ψ, to be $\sum_{\vec{e}\in G} \phi(\vec{e})\psi(\vec{e})$. The following corollary is immediate from the fact that ψ^* can be specified as a linear combination of cycles (of a circulation basis) in G^*.

Corollary 4.2.3. *Any two homologous flows in G send the equal amount of flow through any cocirculation in G.*

Lemma 4.2.4. *There is a feasible (s,t)-flow in G homologous to a given (s,t)-flow ϕ if and only if the dual residual network G^*_ϕ contains no negative-cost cycles.*

Proof: Let ψ be a feasible flow homologous to ϕ; since ψ is feasible, G^*_ψ does not contain a negative-cost cycle. Then Corollary 4.2.2 implies that the cost of any cycle λ in G^*_ϕ is equal to its cost in G^*_ψ, and so G^*_ϕ cannot contain a negative cocycle.

On the other hand, suppose G^*_ϕ has no negative cycles. Fix an arbitrary source vertex r^* in G^*_ϕ. For any face f of G, let $\alpha(f)$ denote the shortest-path distance from r^* to f^* in G^*_ϕ; these distances are well-defined precisely because G^*_ϕ has no negative cycles. Finally, consider the flow $\psi := \phi + \partial\alpha$, which is clearly homologous to ϕ. Because α is defined by shortest-path distances, we have $c_\phi(f\uparrow g) = c_\phi(f^* \to g^*) \geq \alpha(g) - \alpha(f)$, and therefore

$$\begin{aligned}\psi(f\uparrow g) &= \phi(f\uparrow g) + \alpha(g) - \alpha(f)\\ &\leq \phi(f\uparrow g) + c_\phi(f\uparrow g)\\ &= c(f\uparrow g)\end{aligned}$$

for every dart $f\uparrow g$. In other words, ψ is feasible. □

4.2.3 Shortest paths with negative edges

Lemma 4.2.4 and its proof immediately imply an algorithm to find a feasible flow in a given homology class, if one exists, by directly applying any single-source shortest-path algorithm for embedded directed graphs with both positive- and negative-weight edges. The next two theorems [38] describe the best parallel and

serial algorithms known at present. The serial algorithm is a generalization of the planar shortest path algorithms by Klein *et al.* [126] and Mozes and Wulff-Nilsen [144]. We use a system of shortest non-contractible cycles to planarize the surface embedded graph.

Theorem 4.2.5. *After $O(n)$ preprocessing time, given an (s,t)-flow ϕ in G, we can either find a feasible (s,t)-flow homologous with ϕ, or determine correctly that no homologous feasible flow exists, in $O(\log^3 n)$ time and $O(g^{3/2}n^{3/2})$ work on a EREW PRAM.*

Let $\vec{G} = (V, \vec{E})$ denote the symmetric directed graph associated with some undirected graph $G = (V, E)$; this directed graph inherits a cellular embedding from the embedding of G.

Theorem 4.2.6. *We can compute either a single-source shortest-path tree or a negative cycle in $\vec{G}$, with respect to any given edge weights $w : \vec{E} \to \mathbb{R}$, in $O(g^2 n \log^2 n)$ arithmetic operations.*

Corollary 4.2.7. *Given an (s,t)-flow ϕ in G, we can either find a feasible (s,t)-flow in G that is homologous with ϕ, or find a negative cycle in G^*_ϕ if no homologous feasible flow exists, using $O(g^2 n \log^2 n)$ arithmetic operations.*

4.2.4 Basic flows and optimization

Every (s,t)-flow can be expressed as a weighted sum of simple directed cycles and simple directed paths from s to t. Consequently, every homology class of (s,t)-flows is a weighted sum of homology classes of (s,t)-paths and cycles. It follows immediately that the flow homology space $Z(G,st) \cong \mathbb{R}^{2g+1}$ can be generated by the homology classes of $2g+1$ curves, each of which is a (s,t)-path or a cycle. We call such a collection of curves a ***flow homology basis***. Any flow homology basis includes at least one (s,t)-path; we call a flow homology basis *canonical* if it contains *exactly* one (s,t)-path and $2g$ cycles; these $2g$ cycles necessarily define a basis for the space $H(G)$ of homology classes of circulations.

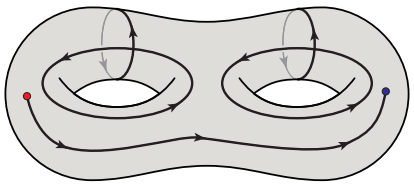


Figure 4.1. A canonical flow homology bases for a surface of genus 2.

Lemma 4.2.8. *We can compute a canonical flow homology basis for G in $O(gn)$ time.*

Proof: Let (T, K, X) be a tree-cotree decomposition, and let $X = \{e_1, e_2, \ldots, e_{2g}\}$.

We define a path p_0 and $2g$ cycles $\gamma_1, \ldots, \gamma_{2g}$ as follows. Let p_0 denote the unique path from s to t in T. For each index i between 1 and $2g$, let γ_i denote the unique cycle in the graph $T \cup e_i$, oriented arbitrarily. We claim that the curves $p_0, \gamma_1, \ldots, \gamma_{2g}$ lie in linearly independent homology classes, and hence comprise a basis for the flow homology space $H(G, st)$.

Any linear combination of cycles is a circulation, but a flow along any path from s to t is not. It follows immediately that the homology class of p_0 is independent from the subspace of $H(G; st)$ generated by homology classes of the cycles $\gamma_1, \ldots, \gamma_{2g}$. It remains only to prove that these $2g$ cycles lie in linearly independent homology classes (and hence define a basis for the homology space $H(G)$).

Suppose to the contrary that $\phi = \sum_{i=1}^{2g} a_i \gamma_i$ is null-homologous for some real coefficients a_i. It follows that $\phi = \partial\alpha$ for a 2-chain α. Observe that, $\phi(e) = 0$ for any $e \in K$. Since K^* is a spanning tree of the dual α assigns the same value to all faces of G, which implies $\phi(e) = 0$ for all edges of G. Thus, all a_i's are zero, which proves the lemma statement.

The tree T and cotree K^* can each be constructed in $O(n)$ time using (for example) depth-first search, after which the path p_0 and each cycle γ_i can be easily constructed in $O(n)$ time. □

Fix a canonical flow homology basis $p_0, \gamma_1, \ldots, \gamma_{2g}$ for G. A ***basic flow*** is any flow ϕ of the form $\phi_0 \cdot p_0 + \sum_{i=1}^{2g} \phi_i \cdot \gamma_i$ for some coefficients $\phi_0, \phi_1, \ldots, \phi_{2g}$. Specifically, we have $\phi_0 = |\phi|$ and $\phi_i = \phi(e_i)$ for each index i. Equivalently, a flow ϕ is basic if and only if $\phi(e) = 0$ for every cotree edge $e \in K$. (Given a flow ϕ that avoids every edge in K, subtracting the basic flow with coefficients $|\phi|, \phi(e_1), \ldots, \phi(e_{2g})$ leaves a circulation that avoids every edge in $K \cup X = G \setminus T$ and is therefore identically zero.) Every flow in G is homologous to exactly one basic flow.

Corollary 4.2.9. *Given the coefficients $\phi_0, \phi_1, \ldots, \phi_{2g}$ of a basic flow ϕ, we can either find a feasible (s,t)-flow in G that is homologous with ϕ, or find a negative cycle in G^*_ϕ if no homologous feasible flow exists, using $O(g^2 n \log^2 n)$ arithmetic operations.*

Corollary 4.2.10. *After $O(n)$ preprocessing time, given the coefficients $\phi_0, \ldots, \phi_{2g}$ of a basic flow ϕ, we can either find a feasible (s,t)-flow homologous with ϕ, or determine correctly that no homologous feasible flow exists, in $O(\log^3 n)$ time and $O(g^{3/2} n^{3/2})$ work on a EREW PRAM.*

The preceding results imply that to compute a maximum (s,t)-flow in G, it suffices to find a basic flow ϕ of maximum value such that the dual residual network G^*_ϕ contains no negative cycles. We can formulate this optimization problem as a linear program as follows.

For any basic flow ϕ and any cocycle λ, we can decompose the total flow through λ as a linear combination of the flow parameters ϕ_i:

$$\phi(\lambda) = \phi_0 \cdot p_0(\lambda) + \sum_{i=1}^{2g} \phi_i \cdot \gamma_i(\lambda).$$

Thus, the optimal basic flow is the solution to the following linear programming problem.

$$\begin{array}{ll} \text{maximize} & \phi_0 \\ \text{subject to} & \phi(\lambda) \le c(\lambda) \quad \text{for each cocycle } \lambda \text{ in } G \end{array} \tag{LP}$$

Most of the constraints in this linear program are redundant; it suffices to consider only cocycles λ whose dual cycles λ^* are simple and have minimum cost in their homology class—that is, simple cocycles λ with minimum residual capacity in their cohomology class. However, the best upper bound we can prove on the number of non-redundant constraints is $n^{O(g)}$. The cohomology class of a cocycle λ can be identified by the vector of flow values $(p_0(\lambda), \gamma_1(\lambda), \ldots, \gamma_{2g}(\lambda))$. Since each curve in the basis is simple, each of these cohomology coefficients is an integer between $-n$ and n. Thus, there are at most $(2n+1)^{2g+1}$ different cohomology classes of simple cocycles in G.

Without a significant improvement in this upper bound, we cannot hope to solve (LP) directly; instead, we turn to implicit solution methods.

Specifically, two different methods: *central cut ellipsoid method* [101], and *multi-dimensional parametric search* [1, 2]. The ellipsoid method requires the capacities of the edges to be integers, and the running time of the algorithm is logarithmically dependent to the sum of the edge capacities. On the other hand, using parametric search we can obtain a combinatorial algorithm with a slower running time. We refer the interested reader to our paper [38] for details.

Theorem 4.2.11. *Given an undirected graph $G = (V, E)$ embedded on an orientable surface of genus g, a positive integer capacity function $c : E \to \mathbb{Z}^+$, and two vertices $s, t \in V$, a maximum (s,t)-flow in G can be computed in time $O(g^8 n \log^2 n \log^2 C)$, where C is the sum of the edge capacities.*

Theorem 4.2.12. *Given a graph $G = (V, E)$ embedded on an orientable surface of genus g, a positive capacity function $c : E \to \mathbb{R}^+$, and two vertices $s, t \in V$, a maximum (s,t)-flow in G can be computed in $g^{O(g)} n^{3/2}$ time.*

4.3 Cohomology cuts

Suppose we are given an undirected graph G (with no source or sink), a positive capacity function $c : E \to \mathbb{R}^+$, and a ***value*** function $\theta : \vec{E} \to \mathbb{R}$. The value of a circulation ϕ is the inner product $\langle \phi, \theta \rangle = \sum_{\vec{e} \in \vec{E}} \phi(\vec{e}) \cdot \theta(\vec{e})$. Like the capacity function c, the value function θ is not (in general) a 1-chain; the values of a dart and its reversal need not have any relationship. In particular, some darts may have negative value. The goal of the *maximum-value circulation* problem is to compute a feasible circulation ϕ whose value $\langle \phi, \theta \rangle$ is as large as possible. The standard maximum-flow problem can be reduced to this problem by adding an edge $t \to s$ with infinite capacity and value 1 to the flow network, and assigning every other edge value 0. The maximum-value circulation problem is equivalently—and much more

commonly—formulated as finding a feasible circulation of minimum *cost*, where the cost of an edge is just the negation of its value.

Our methods do not improve the fastest algorithms for the general maximum-value/minimum-cost circulation problem in surface-embedded graphs; even for planar graphs, the fastest algorithms known are those for arbitrary sparse graphs [64, 145]. However, a minor modification of our maximum-flow algorithm allows us to solve two interesting special cases in roughly the same running time. In the first special case, described in Section 4.3.1, we require that the value function is *homology invariant*; that is, any two homologous circulations must have the same value. In the second special case, described in Section 4.3.2, we find the minimum-cost circulation *in a given homology class*; this special case requires each edge to have a non-negative cost and infinite capacity. These two special cases are related by a combination of combinatorial (Poincaré) duality and linear programming duality.

4.3.1 Homology-invariant values

The maximum-flow algorithm described in the previous section can be easily modified to compute maximum-value circulations, provided all circulations in the same homology class have the same value. We call the value function $\theta : \vec{E} \to \mathbb{R}$ is ***homology-invariant*** if $\langle \phi, \theta \rangle = \langle \psi, \theta \rangle$ for any two homologous circulations ϕ and ψ, or equivalently, if $\langle \partial\alpha, \theta \rangle = 0$ for any 2-chain α. In particular, any homology-invariant value function must be a 1-chain.

Theorem 4.3.1. *Given a graph $G = (V, E)$ embedded on a surface of genus g, a capacity function $c : E \to \mathbb{R}^+$, and a homology-invariant value function $\theta : \vec{E} \to \mathbb{R}$, we can compute a maximum-value circulation in $g^{O(g)}n^{3/2}$ time, or in time $O(g^8 n \log^2 n \log^2 C)$ if capacities are integers that sum to C.*

Proof: The homology space $H(G) \cong \mathbb{R}^{2g}$ can be generated by (the homology classes of) $2g$ directed cycles $\gamma_1, \gamma_2, \ldots, \gamma_{2g}$ in independent homology classes. The proof of Lemma 4.2.8 implies that we can construct such a *homology basis* in $O(gn)$ time [76, 79].

Corollary 4.2.9 implies that it suffices to find the homology class of the maximum-value circulation. Specifically, we must find a feasible homology vector $(\phi_1, \ldots, \phi_{2g})$ such that the cost function

$$\left\langle \sum_{i=1}^{2g} \phi_i \cdot \gamma_i,\ \theta \right\rangle = \sum_{i=1}^{2g} \phi_i \cdot \langle \gamma_i, \theta \rangle$$

is maximized. Corollary 4.2.9 gives us both strong membership and strong separation oracles for this linear optimization problem, so we can apply either the central-cut ellipsoid method or multidimensional parametric search, exactly as we did for the standard maximum-flow problem. □

The following lemma exactly characterizes homology-invariant value functions. Recall that a 1-chain $\theta : E \to \mathbb{R}$ is a ***cocirculation*** if its dual 1-chain $\theta^* : E^* \to \mathbb{R}$,

defined by setting $\theta^*(\vec{e}^*) = \theta(\vec{e})$, is a circulation in G^*.

Lemma 4.3.2. *A value function $\theta : \vec{E} \to \mathbb{R}$ is homology invariant if and only if θ is a cocirculation.*

Proof: If the function $\theta : E \to \mathbb{R}$ is not a cocirculation, then for some face f, we have

$$\langle \partial f, \theta \rangle = \sum_{\vec{e}:\, left(\vec{e})=f} \theta(\vec{e}) \neq 0$$

Because θ gives non-zero value to the boundary circulation ∂f, it cannot be homology invariant.

On the other hand, observe that the value of a circulation ϕ is, in fact, the amount of flow that ϕ sends though θ. So, the other direction of the lemma follows from Corollary 4.2.3. □

4.3.2 Minimum-cost homologous circulation

The special case of maximum-value circulations considered in the previous section has the following natural dual interpretation. Consider the following classical linear programming formulation of the maximum-value circulation problem.

$$\begin{aligned}
\max \quad & \sum_{u\to v} \phi(u\to v)\cdot\theta(u\to v) & \\
\text{s.t.} \quad & \sum_{u:uv\in E} \big(\phi(u\to v) - \phi(v\to u)\big) = 0 & \text{for all } v \in V \\
& \phi(u\to v) \le c(u\to v) & \text{for all } u\to v \in \vec{E} \\
& \phi(u\to v) \ge 0 & \text{for all } u\to v \in \vec{E}
\end{aligned}$$

The dual of this linear program has a variable $\alpha(v)$ for each vertex v and a variable $x(u\to v)$ for each dart $u\to v$.

$$\begin{aligned}
\min \quad & \sum_{u\to v} x(u\to v)\cdot c(u\to v) & \\
\text{s.t.} \quad & \alpha(u) - \alpha(v) + x(u\to v) \ge \theta(u\to v) & \text{for all } u\to v \in \vec{E} \\
& x(u\to v) \ge 0 & \text{for all } u\to v \in \vec{E}
\end{aligned}$$

This dual linear program is more naturally cast in terms of the dual graph G^*, as follows:

$$\begin{aligned}
\min \quad & \sum_{f\uparrow g} x(f\uparrow g)\cdot c(f\uparrow g) & \\
\text{s.t.} \quad & \alpha(f) - \alpha(g) + x(f\uparrow g) \ge \theta(f\uparrow g) & \text{for all } f\uparrow g \in \vec{E}^* \\
& x(f\uparrow g) \ge 0 & \text{for all } f\uparrow g \in \vec{E}^*
\end{aligned} \tag{4.1}$$

Let $\alpha_{OPT}(f)$ and $x_{OPT}(f\uparrow g)$ denote the variables in the optimum solution to this dual-dual linear program. We view the vector α_{OPT} of face variables as a 2-chain. We define a 1-chain $\vartheta : E^* \to \mathbb{R}$ by setting $\vartheta_{OPT}(f\uparrow g) := x_{OPT}(f\uparrow g) - x_{OPT}(g\uparrow f)$

for every dart $f\uparrow g$. Because every primal capacity $c(u\to v)$ is non-negative, each dart variable $x_{OPT}(f\uparrow g)$ is individually as small as possible without violating any constraint; that is,

$$x_{OPT}(f\uparrow g) = \max\{0,\ \theta(f\uparrow g) - \alpha(f) + \alpha(g)\}.$$

It follows immediately that $\vartheta_{OPT} = \theta - \partial\alpha$; thus, ϑ_{OPT} is a circulation in G^*, homologous with the circulation θ. Equivalently, ϑ_{OPT} is a *cocirculation* in G, in the same *cohomology* class as θ. Moreover, the optimal objective value can be rewritten as follows:

$$\sum_{f\uparrow g} x_{OPT}(f\uparrow g)\cdot c(f\uparrow g) = \sum_{e^*\in E^*} c(e^*)\cdot|\vartheta_{OPT}(e^*)|$$

We conclude that ϑ_{OPT} is the minimum-capacity cocirculation in the same cohomology class as θ.

Theorem 4.3.3 (Homological Maxflow/Mincut). *Let $G=(V,E)$ be an undirected graph embedded on a surface of genus g, let $c: E\to\mathbb{R}^+$ be a capacity function, and let $\theta: E\to\mathbb{R}$ be a cocirculation in G. The maximum value $\langle\phi,\theta\rangle$ of any feasible circulation ϕ in G is equal to the minimum capacity of any cocirculation cohomologous with θ.*

The previous theorem is a special case of a more general result of Sullivan [181], relating optimal homologous $(d-1)$-chains ("surfaces") in any orientable d-manifold cell complex to minimum-cost flows in the 1-skeleton of the dual cell complex. Essentially the same result was rediscovered by Buehler *et al.* [22, 97, 124]; see also recent results of Grady [98, 99].

A simple modification of our maximum-value circulation algorithm computes the minimum-cost circulation in a given homology class, in any surface-embedded graph whose edges have non-negative costs but no capacities.

Theorem 4.3.4. *Given a graph $G=(V,E)$ embedded on a surface of genus g, a cost function $c: E\to\mathbb{R}^+$, and a circulation $\theta: E\to\mathbb{R}$, we can compute a minimum-cost circulation homologous with θ in $g^{O(g)}n^{3/2}$ time, or in time $O(g^8 n\log^2 n\log^2 C)$ if all capacities are integers that sum to C.*

Proof: Within the stated time bounds, we can compute a maximum-value feasible circulation ϕ^*_{OPT} in the dual graph G^*, using c as a capacity function and θ as a homology-invariant value function. Our algorithm optimizes the homology class of the circulation, using a linear-programming formulation similar to (LP):

$$\begin{aligned} \max\quad & \sum_{i=1}^{2g} \phi_i^*\cdot\langle\theta,\lambda_i^*\rangle \\ \text{s.t.}\quad & \sum_{i=1}^{2g} \phi_i^*\cdot\lambda_i^*(\gamma) \le c(\gamma) \quad \text{for every cycle } \gamma \text{ in } \vec{G} \end{aligned} \tag{4.2}$$

Here, $\lambda_1^*, \lambda_2^*, \ldots, \lambda_{2g}^*$ are cycles in G^* that generate the homology space of G^*. As described in the proof of Lemma 4.2.8, we can construct a suitable set of $2g$ cycles in $O(gn)$ time.

In any feasible basis for the homology linear program (4.2), exactly $2g$ of the linear constraints are satisfied with equality. These constraints are defined by $2g$ cycles $\gamma_1, \gamma_2, \ldots, \gamma_{2g}$ in $\vec{G}$, which necessarily lie in independent homology classes and thus comprise a basis for the homology space of G. (Moreover, each cycle γ_i is the minimum-cost cycle in its homology class.)

The dual of linear program (4.2) has a non-negative variable $a(\gamma)$ for each directed cycle γ in $\vec{G}$.

$$\begin{aligned} \max \quad & \sum_{\text{cycle } \gamma \text{ in } \vec{G}} c(\gamma) \cdot a(\gamma) \\ \text{s.t.} \quad & \sum_{\text{cycle } \gamma \text{ in } \vec{G}} \lambda_i^*(\gamma) \cdot a(\gamma) \;=\; \langle \theta, \lambda_i^* \rangle \quad \text{for all } 1 \le i \le 2g \\ & a(\gamma) \;\ge\; 0 \qquad\qquad \text{for every cycle } \gamma \text{ in } \vec{G} \end{aligned} \tag{4.3}$$

The variables $a(\gamma)$ are the coefficients of a cycle decomposition of a circulation $\vartheta = \sum_\gamma a(\gamma) \cdot \gamma$; conversely, any circulation ϑ can be expressed as a weighted sum of cycles with non-negative coefficients $a(\gamma)$. The objective function of (4.3) is just the cost of this circulation. Moreover, we can mechanically verify that

$$\sum_{\text{cycle } \gamma \text{ in } \vec{G}} \lambda_i^*(\gamma) \cdot a(\gamma) \;=\; \langle \vartheta, \lambda_i^* \rangle,$$

so the equality constraints specify that ϑ must be homologous with θ. In other words, the optimal solution to (4.3) describes the minimum-cost circulation $\vartheta_{OPT} = \sum_\gamma a_{OPT}(\gamma) \cdot \gamma$ homologous with the given circulation θ.

Complementary slackness implies that in the optimal solution to (4.3), any variable $a_{OPT}(\gamma)$ is non-zero only if the corresponding capacity constraint in (4.2) is satisfied with equality. Thus, the minimum-cost homologous circulation ϑ_{OPT} is a weighted sum of the $2g$ homology basis cycles γ_i.

To simplify notation, let $a_i = a_{OPT}(\gamma_i)$ for each index i, so that $\vartheta_{OPT} = \sum_{i=1}^{2g} a_i \gamma_i$. After computing the dual circulation ϕ_{OPT}^* and the saturated cycles γ_i, we can compute the coefficients a_i time as follows. For each index j, we have a linear equation

$$\langle \theta, \lambda_j^* \rangle = \langle \vartheta_{OPT}, \lambda_j^* \rangle = \left\langle \sum_{i=1}^{2g} a_i \gamma_i,\ \lambda_j^* \right\rangle = \sum_{i=1}^{2g} a_i \langle \gamma_i, \lambda_j^* \rangle.$$

We compute the $O(g^2)$ inner products $\langle \theta, \lambda_j^* \rangle$ and $\langle \gamma_i, \lambda_j^* \rangle$, each in $O(n)$ time, by a brute-force sum over the edges of $\vec{G}$. Finally, we solve the resulting system of $2g$ linear equations in $O(g^3)$ time via Gaussian elimination. □

Finally, consider the special case where the input circulation θ is a single directed cycle. The minimum-cost circulation ϑ_{OPT} homologous to θ is not necessarily a single cycle, but the proof of the previous theorem implies that it is a weighted sum

of at most $2g$ directed cycles. Moreover, ϑ_{OPT} is defined by a linear program (4.1) whose constraint matrix is totally unimodular, which implies that ϑ_{OPT} is an *integer* circulation. For more general applications of total unimodularity to optimization within an integer homology class, we refer the reader to Dey *et al.* [61] and Dunfield and Hirani [66].

Chapter 5

Homotopic Frechét distance

5.1 Introduction

Comparing the shapes of curves – or sequenced data in general – is a challenging task that arises in many different contexts. The ***Frechét distance*** and its variants have been used as a similarity measure in various applications such as matching of time series in databases [123], comparing melodies in music information retrieval [173], matching coastlines over time [137], as well as in map-matching of vehicle tracking data [18, 190], and moving objects analysis [20, 21]. See [6, 7, 86] for algorithms for computing the Frechét distance.

Frechét distance is not always an accurate measure between the curves, particularly where the ambient space is not Euclidian. However, homotopic Frechét distance deliberately considers the ambient space, which in many occasions, make its computing a more challenging problem (see Section 2.10).

Here, we are interested in the problems of computing the homotopic Frechét distance and the homotopy height between two simple polygonal curves that lie on the boundary of an arbitrary triangulated topological disk. Similar to previous chapters, we start with a brief discussion of related results. Section 5.3 provides motivation and an overview of the algorithm. Section 5.4 discusses preliminary results about the geodesics. The approximation algorithms for homotopy height and homotopic Frechét distance problems are described in Section 5.5 and Section 5.6, respectively. Basic definitions are provided in 2.10.

5.2 Related results

Chambers *et al.* [34] gave a polynomial time algorithm to compute the homotopic Frechét distance between two polygonal curves on the Euclidean plane with polygonal obstacles. Chambers and Letscher [19, 40, 41] introduced the notion of minimum homotopy height, and proved structural properties for the case of a pair of paths on the boundary of a topological disk. We remark that in general, it is not known whether the optimum homotopy has polynomially long description. In particular, it is not known whether the problem is in NP.

Alt and Godau [7] describe a polynomial time algorithm to compute the (regular) Frechét distance. Eiter and Mannila [71] study the easier discrete version of this

problem. Computing the Frechét distance between surfaces [87] appears to be a much more difficult task, and its complexity is poorly understood. The problem has been shown to be NP-hard by Godau [93], while the best algorithmic result is due to Alt and Buchin [6], who showed that it is upper semi-computable.

Efrat *et al.* [69] considered the Frechét distance inside a simple polygon as a way to facilitate sweeping it efficiently. They also used the Frechét distance with the underlining geodesic metric as a way to get a morphing between two curves. For recent work on the Frechét distance, see [51, 52, 104, 162] and references therein.

5.3 Motivation and overview

5.3.1 Why are these measures interesting?

For the sake of discussion, assume that we know the starting and ending leash of the homotopy between f and g. The region bounded by the two curves and these leashes is a topological disk, and any mapping realizing the homotopic Frechét distance with the given extreme leashes is a mapping of the unit square to this disk $\mathcal{D}$. This mapping specifies how to sweep over $\mathcal{D}$ in a geometrically "efficient" way (especially if the leash does not sweep over the same point more than once), so that the leash (i.e., the sweeping curve) is never too long [69]. As a concrete example, consider the two curves as enclosing several mountains between them on the surface – computing the homotopic Frechét distance corresponds to deciding which mountains to sweep first and in which order.

Furthermore, this mapping can be interpreted as a surface parameterization [84, 175] and can thus be used in applications such as texture mapping [8, 151]. In the texture mapping problem, we wish to find a continuous and invertible mapping from the texture, usually a two-dimensional rectangular image, to the surface. Obviously, this is not possible when the surface has a different topology from sphere. Nevertheless, finding a map that, roughly speaking, minimizes discontinuity is still interesting.

Another interesting interpretation is when f is a closed curve, and g is a point inside. Interpreting f as a rubber band in a 3d model, the homotopy height between f and g here is the minimum length the rubber band has to be so that it can be collapsed to a point (here, the rubber band stays on the surface as this is happening). In particular, a short closed curve with large homotopic height to any point in the surface is a "neck" in the 3d model.

To summarize, these measures seem to provide us with a fundamental understanding of the structure of the given surface/model.

5.3.2 Overview of the algorithms

In this chapter, we consider the problems of computing the homotopic Frechét distance and the homotopy height between two simple polygonal curves that lie on the

boundary of a triangulated topological disk $\mathcal{D}$ that is composed of n triangles.

We give a polynomial time $O(\log n)$-approximation algorithm for computing the homotopy height between f and g. Our algorithm to compute an approximate homotopy between f and g uses a simple yet delicate divide and conquer algorithm.

We use the homotopy height algorithm as an ingredient for an approximation algorithm for the homotopic Frechét distance problem. Here is an informal (and somewhat imprecise) description of our algorithm for approximating the homotopic Frechét distance: we first guess $d_{\mathcal{H}}(f,g)$; the actual algorithm implements guessing using a strongly polynomial search scheme. Using this guess, we interpret the potions of $\mathcal{D}$ over which a short leash cannot pass as "obstacles". Let D' be the punctured disk obtained from $\mathcal{D}$ after removing these obstacles. Observe that the leashes of the optimum solution belong to the same homotopy class (after contracting the input paths). We describe a greedy algorithm to compute a "small" number of homotopy classes out of infinite number of choices. The homotopic Frechét distance constrained to one of these classes is a polynomial approximation to the homotopic Frechét distance in $\mathcal{D}$. We can then do a binary search over this interval to get a better approximation. An extended version of the homotopy height algorithm is used in this algorithm in several places.

The $O(\log n)$ factor shows up in the homotopic Frechét distance algorithm only because it uses the homotopy height as a subroutine. Thus, any constant factor approximation algorithm for the homotopy height problem implies a constant factor approximation algorithm for the homotopic Frechét distance.

5.4 Geodesic paths, an overview

We say that a triangulated surface is ***non-degenerate*** if for every non-boundary vertex x, the sum of the angles of the triangles incident to x is not equal to 2π. We can turn any triangulated surface into a non-degenerate one by perturbing all edge lengths by a factor of at most $1+\epsilon$, for some $\epsilon = O(1/n^2)$. Since, each shortest path goes through $O(n)$ triangles, perturbation changes the metric of the surface by at most a factor of $O(1+1/n)$, and thus the minimum height of a homotopy. Since such a factor will be negligible for our approximation guarantee, we can assume that the input surface is always non-degenerate.

Recall that a path is a ***geodesic*** if and only if it is locally a shortest path; it cannot be shortened by an infinitesimal perturbation. In particular, global shortest paths are geodesics. Only during this section, we use the term ***curve*** as an alternative for path. A path or a curve is polygonal if it is composed of a finite number of segments.

Mitchell *et al.* [140] describe an algorithm to compute the shortest path distance from a single source to every other points of an embedded polygonal surface in $\mathbb{R}^3$ in $O(n^2 \log n)$ time. The same algorithm can be adapted to compute the shortest path distance from an edge to every other points of the surface in the same asymptotic running time. It follows that the shortest path from a set of $O(n)$ edges to the every

other points of the surface can be computed in $O(n^3 \log n)$.

The shortest path from a point in $\mathcal{D}$ to a set is a geodesic. So, it is a polygonal line that intersects every edge at most once at a point and passes through a face along a segment. The shortest path crossing an edge looks locally like a straight segment, if one rotates the adjacent faces so that they are coplanar [140].

Let the source S be a set of edges of $\mathcal{D}$ and let π be a shortest path from a point p to S. We define the ***crossing sequence*** of π to be the ordered set of edges (crossed or used) by π. Since π is locally a straight segment, we can rotate all faces that intersect π one by one so that π becomes a straight line. It follows that there cannot be two geodesics from p with the same crossing sequence.

A point p on the surface is a ***medial*** point (with respect to S) if there are more than one shortest paths (with different crossing sequences) from p to S.

The following theorem, essentially taken from Mitchell *et al.* [140], is very helpful during this section.

Theorem 5.4.1. *Let $\mathcal{D}$ be a triangulated polyhedral disc in $\mathbb{R}^3$, composed of n triangles, v be a vertex of $\mathcal{D}$, and S be a set of $O(n)$ edges of $\mathcal{D}$. Then,*

(A) The shortest path from v to every other points of $\mathcal{D}$ can be computed in $O(n^2 \log n)$ time,

(B) The shortest path from S to every other points of $\mathcal{D}$ can be computed in $O(n^3 \log n)$ time,

(C) Any shortest path intersects a face along a segment. Further, two adjacent segments of any shortest path become colinear after the adjacent faces are rotated to be coplanar.

5.5 Homotopy height

In this section, we give an approximation algorithm for finding a homotopy of minimum height in a topological disc $\mathcal{D}$, whose boundary is defined by two walks L and R that share their end-points s and t. We start with the discrete case, i.e. when the disk is a triangulated edge-weighted planar graph. We use the ideas developed here in the continuous case; see 5.5.3.

5.5.1 Settings

We are given an embedded planar graph $\mathcal{D}$ all of whose faces (except possibly the outer face) are triangles. Let s and t be vertices on the outer face of $\mathcal{D}$ and L and R be the two non-crossing (s,t)-walks with shared endpoints on the outer face in counter-clockwise and clockwise order, respectively. We also use $\mathcal{D}$ to refer to the topological disk enclosed by $L \cup R$. Our goal is to find a minimum height homotopy from L to R of non-crossing walks. Informally, the homotopy is defined by

a sequence of walks, where every two consecutive walks differ by either a triangle, or an edge (being traversed twice). For a formal definition, see Section 2.10.

Lemma 5.5.1. *Let x and y be vertices of $\mathcal{D}$. Any homotopy between L and R has height at least $d(x,y)$.*

Proof: Fix a homotopy of height δ. This homotopy contains an (s,t)-walk ω that passes through x, and an (s,t)-walk χ that passes through y. We have, by the triangle inequality, that $\rho = d_{\mathcal{D}}(x,y) \leq |\omega[s,x]| + |\chi[s,y]|$, and $\rho \leq |\omega[x,t]| + |\chi[y,t]|$. Therefore, $\rho \leq (|\omega| + |\chi|)/2 \leq \max(|\omega|, |\chi|) \leq \delta$, as required. □

Lemma 5.5.2. *Suppose d_1 is the maximum distance of a vertex of $\mathcal{D}$ from either of L or R, d_2 is the largest edge weight, and let $d = \max\{d_1, d_2\}$. Furthermore, let $\mathcal{D}$, L, and R be defined as above. Then any homotopy between L and R has height at least d.*

Proof: For every homotopy between L and R, and for every edge e, there exists a walk in the homotopy that passes through e. Therefore, the height of the homotopy is at least d_2. Moreover, the height is at least d_1 by 5.5.1. □

5.5.2 The discrete algorithm

Theorem 5.5.3. *Let $\mathcal{D}$ be an edge-weighted triangulated topological disk with n faces whose boundary is formed by two walks L and R that share endpoints s and t. One can compute, in $O(n \log n)$ time, a homotopy from L to R of height at most $|L| + |R| + O(d_L \log n)$, where d_L is the largest among (i) the maximum distance of a vertex of $\mathcal{D}$ from L, and (ii) the maximum edge weight.*

In particular, the generated homotopy has height $O(hh_{\text{opt}} \log n)$, where hh_{opt} is the minimum homotopy height between L and R.

Proof: Here, we describe a recursive algorithm to build a homotopy. Let $f(|L| + |R|, d_L, n)$ denote the maximum height of such a homotopy. For the purpose of analysis, we describe the height as a function of $|L| + |R|$, and not $|L|$ and $|R|$. We will show that $f(u, d_L, n) = u + O(d_L \log n)$.

The base case $n = 0$ is easy. Indeed, if we have two edges (u,v) and (v,u) consecutive in R (or in L) we can retract these two edges. By repeating this procedure we arrive at both L and R being identical, and we are done. The case $n = 1$ is handled in a similar fashion. After one face flip, the problem reduces to the case $n = 0$. As such, $f(|L| + |R|, d_L, 1) \leq |L| + |R| + d_L$.

So suppose $n > 1$. Compute for each vertex of $\mathcal{D}$ its shortest path to L, and consider the set of edges $\mathcal{E}$ used by all these shortest paths. These shortest paths can be chosen so that $L \cup \mathcal{E}$ form a tree; for example by contracting L and running any shortest path finder to all other vertices. Now we consider a coy R' of R and for each vertex $v \in R'$ we connect it to its corresponding copy of R with a ***corridor*** edge. Intuitively, we build a corridor along R to make further discussions simpler.

After duplicating R we obtain a graph whose faces are (original) triangles, the outer face, and the new 4-sided polygons within the corridor. Note that $\mathcal{E}$ does not use any edge of R'.

Now, if we cut $\mathcal{D}$ along the edges of $\mathcal{E}$, what remains is a simple polygon composed of triangles and 4-sided polygons gons. One can find a diagonal uv such that each side of the diagonal contains at least $\lceil n/3 \rceil$ triangles of $\mathcal{D}$ (and at most $(2/3)n$). We emphasize that we only count the number of triangles and not the 4-sided polygons. We consider two cases:

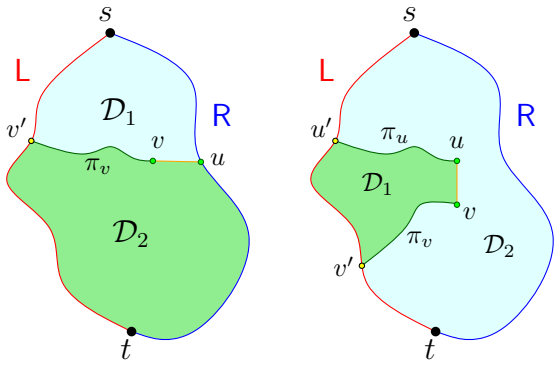


Figure 5.1. Left: uv us a corridor edge, right: uv is not a corridor edge.

Case 1: uv is a corridor edge: Assume, without loss of generality, that $v \in \mathcal{E}$ and $u \in R$. Let π_v be the shortest path in $\mathcal{D}$ from v to L, and let v' be its endpoint on L, see Figure 5.2, left.

Consider the disk $\mathcal{D}_1$ having left boundary $L_1 = L[s, v'] \cup \pi_v \cup vu$ and $R_1 = R[s, u]$ as its right boundary. This disk contains at most $2n/3$ triangles, and by the inductive hypothesis, it has a homotopy of height $f(|L_1| + |R_1|, d_L, (2/3)n)$. Observe that u and v are copies of the same vertex of R. That is, all shortest paths of vertices inside $\mathcal{D}_1$ to L are completely inside $\mathcal{D}_1$. Particularly, the distance of all vertices in $\mathcal{D}_1$ to L_1 are at most d_L.

Similarly, the topological disk $\mathcal{D}_2$ with left boundary $L_2 = uv \cup \pi_v \cup L[v', t]$ and right boundary $R_2 = R[u, t]$ has a homotopy of height $f\left(|L_2| + |R_2|, d_L, (2/3n\right)$.

Starting with L, extending a tendril from v' to v, from v to u, and then applying the homotopy to first half of this walk (i.e., L_1) to move to R_1, and then the homotopy of $\mathcal{D}_2$ to the second part, results in a homotopy of L to R of height

$$\max \begin{pmatrix} |L| + 2d_L, \\ f\left(|L_1| + |R_1|, d_L, (2/3)n\right) + |L_2|, \\ |R_1| + f\left(|L_2| + |R_2|, d_L, (2/3)n\right) \end{pmatrix}.$$

If the first number is the maximum, we are done. Otherwise, the above value is at most $f(|L| + |R| + 2d_L, d_L, 2/3n)$.

Case 2 (uv is not a corridor edge) Here we handle the case that u and v are both vertices of $L \cup \mathcal{E}$. Then, as before, let u' and v' be the closest points on L to u

and v, respectively. Now, let π_u (resp. π_v) be the shortest path from u (resp. v) to u' (resp. v').

Consider the disk $\mathcal{D}_1$ having $L_1 = L[u', v']$ as left boundary, and $R_1 = \pi_u \cup uv \cup \pi_v$ as right boundary. This disk contains between $n/3$ and $2n/3$ triangles of the original surface. The distance of any vertex of $\mathcal{D}_1$ to L_1 (when restricted to $\mathcal{D}_1$) is at most d_L, and as such by induction, there is a homotopy from L_1 to R_1 of height $\alpha = f\left(|L_1| + |R_1|, d_L, 2n/3\right) \leq f\left(|L[u', v']| + 3d_L, d_L, 2n/3\right)$. This yields a homotopy of height $\alpha_1 = |L[s, u']| + \alpha + |L[v', t]|$, from L to $L_2 = L[s, u'] \cup \pi_u \cup uv \cup \pi_v \cup L[v, t]$. It is straight forward to check that $\alpha_1 \leq f\left(|L| + 3d_L, d_L, 2n/3\right)$.

Next, let $\mathcal{D}_2$ be the disk with its left boundary being L_2 and its right boundary being $R_2 = R$. Observe, that as before, the maximum distance of any vertex of $\mathcal{D}_2$ to L_2 is at most d_L. As before, by induction, there is a homotopy form L_2 to R_2 of height $\alpha_2 = f\left(|L_2| + |R_2|, d_L, 2n/3\right)$. Since $|L_2| \leq |L| + 3d$, we have $\alpha_2 \leq f\left(|L| + |R| + 3d_L, d_L, 2n/3\right)$.

In all cases the length of the homotopy is at most

$$f\left(|L| + |R| + 3d_L, d_L, 2n/3\right).$$

Now, it is easy to verify that the solution to the recursion $f(u, d_L, n)$ that complies with all the above inequalities is $f(u, d_L, n) = u + O(d_L \log n)$, as desired.

The final guarantee of approximation follows as $d_L \leq h h_{\text{opt}}$, by 5.5.2.

We can compute the shortest path tree in linear time using the algorithm of Henzinger *et al.* [109]. The separating edge can also be found in linear time using DFS. So, the running time for a graph with n faces is $T(n) = T(n_1) + T(n_2) + O(n)$, where $n_1 + n_2 = n$ and $n_1, n_2 \leq 2n/3$. It follows that $T(n) = O(n \log n)$. □

In the algorithm of 5.5.3, it is not necessary that we have the shortest paths from L to all the vertices of $\mathcal{D}$. Instead, it suffices to have a tree of paths of length at most d_L from any vertex of $\mathcal{D}$ to L. We will use this property in the continuous case, where recomputing the shortest path tree is relatively expensive.

A more careful analysis shows that the height of the homotopy generated by Theorem 5.5.3 is at most $\max(|L|, |R|) + O\left(d_L \log n\right)$.

If $d_L = O\left(\max(|L|, |R|) / \log n\right)$ then Theorem 5.5.3 provides a constant factor approximation. In this situation L and R are close to each other compared to their relative length.

5.5.3 The continuous algorithm

In this section we extend the algorithm to the continuous case. Here we are given a piecewise linear triangulated topological disk $\mathcal{D}$ with n triangles. The boundary of $\mathcal{D}$ is composed of two paths L and R with shared endpoints s and t. Observe that the distance of any point x in $\mathcal{D}$ from L and R is not longer than the homotopy height as there is a (s, t)-path that contains x. Here, we build a homotopy of height

$|L| + |R| + O(d \log n)$, where d is the maximum distance of any point in $\mathcal{D}$ from either L or R.

5.5.3.1 Homotopy height if edges are short

Here, we assume that the longest edge in $\mathcal{D}$ has length at most $2d$, where d is the maximum distance for any point of $\mathcal{D}$ from either L or R.

As in the discrete case, let $\mathcal{E}$ be the union of all the shortest paths from the vertices of $\mathcal{D}$ to L. As before, we treat the edges and vertices of R as having infinitesimal thickness. For a vertex v of $\mathcal{D}$, its shortest path π_v is a polygonal path that crosses between faces (usually) in the middle of edges; it might also go to a vertex, merge with some other shortest paths and then follow a common shortest path back to L. Recall from Section 5.4 that each shortest path intersects a face of $\mathcal{D}$ along a single segment or not at all. As such, the cell-complex P resulted from cutting $\mathcal{D}$ along $\mathcal{E}$ has complexity $O(n^2)$. A face of P is a hexagon, a pentagon, a quadrilateral, or a triangle. However, each such face has at most three edges that are portions of the edges of $\mathcal{D}$. We say the degree of a face is i if it has i edges that are portions of the edges of $\mathcal{D}$. Observe that, each triangle of $\mathcal{D}$ is now decomposed into a set of faces. Obviously, each triangle of $\mathcal{D}$ contains at most one face of degree 3 in P. Overall, there are $O(n)$ faces of degree 3 in P.

Now consider C^*, the dual of the graph that is inside the polygon (ignore the edges on the boundary). More precisely, C^* has a vertex corresponding to each face inside the polygon P. Two vertices of C^* are adjacent if and only if their corresponding faces share a portion of an edge of $\mathcal{D}$ (this shared edge is a diagonal of the polygon resulting from the cutting). Since the maximum degree of the tree C^* is 3, there is an edge that is a good separator. We use this edge in a similar fashion to the proof of 5.5.3, except that in the recursion we avoid recomputing the shortest paths (i.e., we use the old shortest paths and distances computed in the original disk). So, we compute the shortest paths once in the beginning in $O(n^3 \log n)$ time (see Theorem 5.4.1). Then, in each step we can find the separator in $O\left(n^2\right)$ time. Namely, the total time spent on computing the separators is $T(n) = T(n_1) + T(n_2) + O(n^2)$, where $n_1 + n_2 = O\left(n^2\right)$ and $n_1, n_2 \leq (2/3)(n_1 + n_2)$; that is, $T(n) = O(n^2 \log n)$. As such, the total running time is dominated by the computation of the shortest paths.

The proof of 5.5.3 then goes through literally in this case. Since all the edges have length at most $2d$, by assumption, we get the following.

Lemma 5.5.4. *Let $\mathcal{D}$ be a topological disk with n faces where every face is a triangle (here, the distance between any two points on the triangle is their Euclidean distance). Furthermore, the boundary of $\mathcal{D}$ is formed by two walks L and R (that share two endpoints s, t). Let d_L be the maximum distance of any point of $\mathcal{D}$ from L. Furthermore, assume that all edges of $\mathcal{D}$ have length at most $2d_L$. Then, one can compute a continuous homotopy from L to R of height at most $|L| + |R| + O\left(d_L \log n\right)$ in $O(n^3 \log n)$ time.*

5.5.3.2 Breaking the disk into strips, pockets and chunks

For any two points in $\mathcal{D}$ consider a shortest path π connecting them. The crossing sequence of π is the ordered sequence of edges (crossed or used) and vertices used by π, see Section 5.4. For a point $p \in R$, let $s_L(p)$ denote the crossing sequence of the shortest path from p to L. The crossing sequence $s_L(p)$ is well defined in R except for a finite set of ***medial*** points, where there are two (or more) distinct shortest paths from L to p. In particular, let Π_R be the set of all shortest paths from any medial point on R to L. Observe that, the medial points are the only points that the crossing sequence of the shortest path from R to L changes in any non-degenerate triangulation.

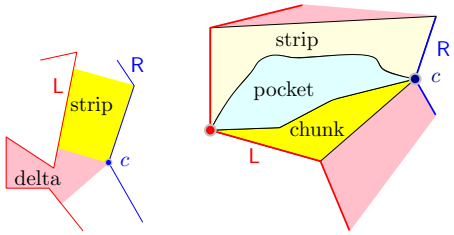


Figure 5.2. Strip, delta, pocket and chunk.

Cutting $\mathcal{D}$ along the paths of Π_R breaks $\mathcal{D}$ into corridors. If the intersection of a corridor with R is a point (resp. segment) then it is a delta (resp. ***strip***). In a strip C, all the shortest paths to L from the points in the interior of the segment $C \cap R$ have the same crossing sequence. Intuitively, strips have a natural way to morph from one side to the other. We further break each delta into chunks and pockets, as follows.

So, consider a delta C with an apex c (i.e., the point of R on the boundary of C). For a point $x \in L \cap C$, we define its crossing sequence (in relation to C), to be the crossing sequence of the shortest path from x to c (restricted to lie inside C). Again, we partition $L \cap C$ into maximum intervals that have the same crossing sequence. If a newly created region has a single intersection point with both L and R, then it is a ***pocket***, otherwise, it is a ***chunk***.

Applying the above partition scheme to all the deltas results in a decomposition of $\mathcal{D}$ into strips, chunks and pockets.

5.5.3.2.1 Analysis Let d the maximum distance of any point of $\mathcal{D}$ to either L or R, and consider a chunk C. Its intersection with L is a segment, and its intersection with R is a point (i.e., the apex c of the delta). Observe that the distance of any point of $x \in L \cap C$ to c is at most $2d$. To see that, consider the shortest path π_x from x to R in $\mathcal{D}$, and observe that if it intersects the boundary of C then it can be modified to connect to c, and its new length is $2d$. As such, for a chunk C there is a natural way to morph $L \cap C$ to c.

A pocket, on the other hand, is a topological disk that its intersections with L and R are both single points, and the two boundary paths between these intersections are

of length at most $2d$. Pockets are handled by using the recursive scheme developed for the discrete case.

5.5.3.3 Homotopy height if there are long edges

We use the algorithm described above to break the given disk $\mathcal{D}$ into strips, chunks and pockets (notice, that we assume nothing on the length of the edges). Next, order the resulting regions according to their order along L, and transform each one of them at time, such that starting with L we end up with R.

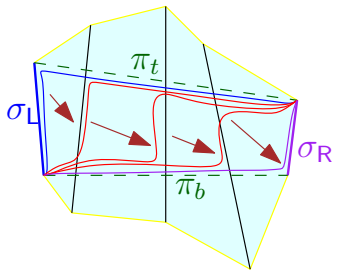


Figure 5.3. Morphing a chunk/strip.

- **Morphing a chunk/strip** S: Let $\sigma_L = L \cap S$ and $\sigma_R = R \cap S$. There is a natural homotopy from $\pi_t \cup \sigma_L$ to $\sigma_R \cup \pi_b$.

  The strip/chunk S has no vertex of $\mathcal{D}$ in its interior, and as such it is formed by taking planar quadrilaterals and gluing them together along common edges. Observe that by the triangle inequality, all such edges of any of these quadrilaterals are of length at most $\max(|\sigma_L|, |\sigma_R|) + 4d$. It is now easy to check that we can collapse each such quadrilateral in turn to get the required homotopy. Since each of π_t and π_b is composed of two shortest paths, there is a linear number of such quadrilaterals, and each collapse can be done in constant time.

- **Morphing a pocket**: A pocket has perimeter at most $4d$, and there is a point on its boundary, such that the distance of any point in it to this base point is at most $2d$. By the triangle inequality, we have that if in a topological disk $\mathcal{D}$ all the points of $\mathcal{D}$ are in distance at most $2d$ from some point c, then the longest edge in $\mathcal{D}$ has length at most $4d$. As such, all the edges inside a pocket can not be longer than $4d$. We can now apply 5.5.4 to such a pocket. This results in the desired homotopy.

The shortest paths from R to L can be computed in $O(n^3 \log n)$ time. The shortest paths inside a delta to its apex can be computed in $O(n^2 \log n)$. Since there is a linear number of deltas, the total running time for building the strips is $O(n^3 \log n)$.

Lemma 5.5.5. *The number of paths in* Π_R *is* $O(|V(\mathcal{D})|)$*, where* $V(\mathcal{D})$ *is the set of vertices of* $\mathcal{D}$.

Proof: Let $\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ be the paths in Π_R sorted by the order of their endpoints along R. Observe that these paths are geodesics and so one can assume that they are interior disjoint. Now, if $l_i \in L$ and $r_i \in R$ are the endpoints of σ_i, for $i = 1, \ldots, k$, then these endpoints are sorted along their respective curves. In particular, let $\mathcal{D}_i$ be the disk having $L[s, l_i] \cup \sigma_{i+1} \cup R[s, r_i]$ for boundary. We have that $\mathcal{D}_1 \subseteq \mathcal{D}_2 \subseteq \cdots \subseteq \mathcal{D}_k$. The crossing sequences of σ_i and σ_{i+2} must be different as otherwise they would be consecutive. Furthermore, because of the inclusion property, if an edge or a vertex of $\mathcal{D}$ intersects σ_i but does not intersect σ_{i+1} then, it can not intersect any later path. As such, every other path in Π_R can be charged to vertices or edges that are added or removed from the crossing sequence of the respective path. Since an edge or a vertex can be added at most once, and deleted at most once, this implies the desired bound on the number of paths. □

Arguing as in 5.5.5, we have that the total number of parts (i.e., strips, chunks, and pockets) generated by the above decomposition is $O(|V(\mathcal{D})|)$.

Lemma 5.5.6. *Consider a strip or a chunk S generated by the above partition of $\mathcal{D}$. Let $\sigma_L = L \cap S$ and $\sigma_R = R \cap S$. Let π_t and π_b be the top and bottom paths forming the two sides of S that do not lie on R or L.*

- *We have $|\pi_b| \leq 2d$ and $|\pi_t| \leq 2d$.*
- *If $|\sigma_L| > 0$ or $|\sigma_R| > 0$ then there is no vertex of $\mathcal{D}$ in the interior of S.*
- *If $|\sigma_L| > 0$ or $|\sigma_R| > 0$ then there is a homotopy from $\pi_t \cup \sigma_L$ to $\sigma_R \cup \pi_b$ of height $\max(|\sigma_L|, |\sigma_R|) + 4d$. This homotopy can be computed in linear time.*

Proof: (A) If the strip was generated by the first stage of partitioning then the claim is immediate.

Otherwise, consider a delta C with an apex c. For any point $x \in L \cap C$ we claim that there is a path of length at most $2d$ to c. Indeed, consider the shortest path π_x from x to R in $\mathcal{D}$. If this path goes to c the claim holds immediately. Otherwise, the shortest path (that has length at most d) must cross either the top or bottom shortest path forming the boundary of C that are emanating from c. We can now modify π_x, so that after its intersection point with this shortest path, it follows it back to c. Clearly, the resulting path has length at most $2d$ and lies inside the resulting chunk.

(B) Indeed, the boundary paths π_t and π_b have the same crossing sequence (formally, they are the limit of paths with the same crossing sequence). Since $\mathcal{D}$ is non-degenerate, if there was any vertex in the middle, then the path on one side of the vertex, and the path on the other side of the vertex can not possibly have the same crossing sequence.

(C) Immediate from the algorithm description. □

5.5.3.4 The result

Theorem 5.5.7. *Suppose that we are given a triangulated piecewise linear surface with the topology of a disk, such that its boundary is formed by two walks L*

and R. Then, there is a continuous homotopy from L to R of height at most $|L|+|R|+O\left(d\log n\right)$, where d is the maximum geodesic distance of any point of $\mathcal{D}$ from either L or R. This homotopy can be computed in $O\left(n^3\log n\right)$ time.

5.6 Homotopic Frechét distance

In this section, fix $\mathcal{D}$ to be a triangulated topological disk with n faces. Let the boundary of $\mathcal{D}$ be composed of T, R, B, and L, four internally disjoint walks appearing in clockwise order along the boundary. Also, let $t_l = L \cap T$, $b_l = L \cap B$, $t_r = R \cap T$, and $b_r = R \cap B$.[1] See Figure 5.4.

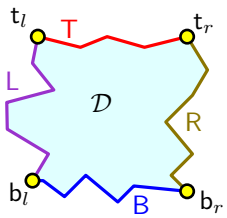


Figure 5.4. The input triangulated disk $\mathcal{D}$, whose boundary is composed of L, T, R and B.

5.6.1 Approximating the regular Frechét distance

5.6.1.1 The continuous case

Let $d_{\mathcal{F}}(T,B)$ (resp. $d_{\mathcal{H}}(T,B)$) be the regular (resp. homotopic) Frechét distance between T and B (when restricted to $\mathcal{D}$). Clearly, $d_{\mathcal{F}}(T,B) \leq d_{\mathcal{H}}(T,B)$. The following lemma implies that the Frechét distance can be approximated within a constant factor.

Lemma 5.6.1. *Let $\mathcal{D}$, n, T, and B be as above. Then, for the continuous case, one can compute, in $O(n^3\log n)$ time, reparametrizations of T and B of width at most $2d_{\mathcal{F}}(T,B)$.*

Proof: We compute the shortest path L (resp. R) between the left (resp. right) vertices of T and B. In the following, consider $\mathcal{D}$ to be region bounded by these four curves.

We decompose $\mathcal{D}$ into strips, chunks and pockets using the algorithm of Subsection 5.5.3.2 (using T and B here as the two boundary curves). Clearly, any region in this decomposition has boundary made out of portions of T and B, and two other curves that have length at most 2δ, as implied by the analysis in Subsection 5.5.3.2. Clearly, a region that is a strip or a chunk does not contain any vertex of $\mathcal{D}$ in its interior, and it provides a natural Frechét reparameterization, between corresponding portions of T and B, of width at most 2δ. As for pockets, observe that the leash does not have to move continuously and it can jump from one boundary curve to the

[1] We use the same notation to argue about the discrete and continuous problems.

other boundary curve (of this pocket). Namely, we computed a reparametrization of T and B of width 2δ, as desired. □

5.6.2 The discrete case

We can use a similar idea to the decomposition into atomic regions as done above.

Lemma 5.6.2. *Let $\mathcal{D}$ be a triangulated topological disk with n faces, and T and B be two internally disjoint walks on the boundary of $\mathcal{D}$. Then, for the discrete case one can compute, in $O(n)$ time, reparametrizations of T and B that approximate the discrete Frechét distance between T and B. The computed reparametrizations have width at most 3δ, where δ is the Frechét distance between T and B.*

Proof: First, compute the set of shortest paths, $\Pi_T = \{\pi_1, \pi_2, \cdots, \pi_k\}$, from vertices of T to the path B. To this end, we (conceptually) collapse all the vertices of B into a single vertex, and compute the shortest path from this meta vertex to all the vertices in $\mathcal{D}$.

Now, let π_i and π_{i+1} be two consecutive paths; that is, the endpoints of π_i and π_{i+1}, a_i and a_{i+1}, are adjacent vertices on T. For all $1 \leq i < k$, we add the paths $\pi_i^+ = (a_i, a_{i+1}) \cdot \pi_{i+1}$ to the set Π_T to obtain Π_T^+. Observe that each path in Π_T^+ has length at most 2δ; it is composed of zero or one edge of T and a shortest path from a vertex of T to B. Further, Π_T^+ partitions the graph into regions, similar to the continuous case. Now for each vertex of B that is not an endpoint of a path in Π_T^+, we compute the shortest path inside its region to T. Because the region is bounded by paths of length at most 2δ, the length of such a shortest path is at most 3δ. If Π_B is the set of all such shortest paths, then $\Pi_T^+ \cup \Pi_B$ is a leash sequence of height at most 3δ.

We use the algorithm of Henzinger *et al.* [109] to compute the shortest paths from B in linear time. Since all regions are disjoint, and every edge appears on the boundary of at most two regions, we can compute all the shortest paths inside all these regions to T in $O(n)$ time overall (this step requires careful implementation to achieve this running time). □

The paths realizing the Frechét distance computed by Lemma 5.6.2 are stored using implicit data-structure (essentially two shortest path trees that are intertwined). Therefore, the used space is linear, and it can be constructed in linear time. Of course, an explicit listing of the realizing paths may require quadratic space.

5.6.3 Without mountains

The following lemma implies that if all the vertices in $\mathcal{D}$ are not too far from the two curves, then we can approximate homotopic Frechét distance is doable.

Lemma 5.6.3. *Let $\mathcal{D}$ be a triangulated topological disk with n faces, and T and B be two internally disjoint walks on the boundary of $\mathcal{D}$. Further, assume for all $p \in \mathcal{D}$,*

p's distance to both T and B is at most x. Then, one can compute reparametrizations of T and B of width $O(x \log n)$. The running time is $O(n^4 \log n)$ (resp. $O(n^2 \log n)$) in the continuous (resp. discrete) case.

In particular, if $x = O(d_{\mathcal{H}}(T,B))$ then this is an $O(\log n)$-approximation to the optimal homotopic Frechét distance.

Proof: Consider the continuous case. Using the algorithm of Lemma 5.6.1 we compute (not necessarily continuous) reparametrization of T and B of width δ, realizing approximately (the regular) Frechét distance, where $\delta = O(x)$. Let $\ell(t)$ denote the leash at time t. The leash $\ell(\cdot)$ is not required to deform continuously in t. In particular, for a given time $t \in [0,1]$, let $\ell^-(t) = \lim_{t' \to t^-} \ell(t')$ and $\ell^+(t) = \lim_{t' \to t^+} \ell(t')$. By definition, the leash is discontinuous at t if and only if $\ell^-(t) \neq \ell^+(t)$.

Naturally, the above reparameterization can be used as long as it is continuous. Whenever the leash jumps over a gap (i.e., the leash is discontinuous at this point in time), say at time t, we are going to replace this jump by a $(\ell^-(t), \ell^+(t))$-homotopy between the two leashes. Clearly, this would result in the desired continuous homotopy.

To this end, observe that all the vertices inside the disc with boundary $\ell^-(t) \cup \ell^+(t)$ have distance $O(x)$ to T and B, and thus also so to $\ell^-(t)$ and $\ell^+(t)$. As such, using the algorithm of Theorem 5.5.7 compute an $(\ell^-(t), \ell^+(t))$-homotopy with height $O(x \log n)$. Since a gap must contain a vertex there are $O(n)$ gaps, so this filling in is done at most $O(n)$ times. Computing the initial reparameterization takes $O(n^3 \log n)$ time. Each gap can be filled in $O(n^3 \log n)$ time.

The discrete case is similar. The Frechét distance here can be computed in linear time using 5.6.2. Filling each gap takes $O(n \log n)$ time using 5.5.3. Overall, the resulting running time is $O(n^2 \log n)$. □

Lemma 5.6.3 demonstrates that if the starting and ending leashes are known (that is the region of the disk $\mathcal{D}$ swept over by the morphing) then we can approximate the homotopic Frechét distance within a $O(\log n)$ ratio. The challenge is that a priori, we do not know these two leashes, as the input is a topological disk $\mathcal{D}$ with the two curves T and B on its boundary, and the extreme leashes might lie in the interior of $\mathcal{D}$.

5.6.4 With mountains, a decision procedure

For a parameter $\tau \geq 0$, a vertex $v \in V(\mathcal{D})$ is ***τ-tall*** if and only if its distance to T or B is larger than τ (intuitively τ is a guess for the value of $d_{\mathcal{H}}(T,B)$). Here, we consider the case where there are τ-tall vertices. Intuitively, one can think about tall vertices as insurmountable mountains. Thus, to find a good homotopy between T and B, we have to choose which "valleys" to use (i.e., what homotopy class the solution we compute belongs to if we think about tall vertices as punctures in the disk).

In the discrete case, we subdivide each edge in the beginning so that if an edge has length $> 2\tau$, then the vertex inserted in the middle of it is τ-tall. Observe that, if $\tau \geq d_{\mathcal{H}}(T,B)$ then no leash of the optimum homotopic motion can afford to contain a τ-tall vertex. We use M^τ to denote the set of all τ-tall vertices in $V(\mathcal{D})$.

Now, let ω and ω' be two walks connecting points on T and B. We say that ω and ω' are ***homotopic*** in $\mathcal{D} \setminus M^\tau$ if and only if they are homotopic in $\mathcal{D} \setminus M^\tau$ after contracting T and B (each to a single point).

Given a subset $X \subseteq M^\tau$, consider a path σ from T to B, such that X is contained in one side of $\mathcal{D} \setminus \sigma$ (i.e., cutting $\mathcal{D}$ along σ breaks it into two connected components), and $M^\tau \setminus X$ is contained in the other side. The set of all such paths is the ***partitioning class*** of X. Note that each partitioning class contains infinitely many homotopy classes of paths. However, it is straightforward to check that two non-crossing paths in a same partitioning class are homotopic.

Let $\pi_{\mathrm{L,h}}$ be the ***left geodesic*** (resp. ***right geodesic***) of a partitioning class h; that is, $\pi_{\mathrm{L,h}}$ denotes the shortest path in h from t_l to b_l (resp. from t_r to b_r).

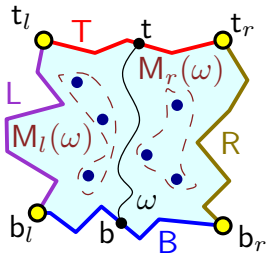


Figure 5.5. A partitioning class.

Let ω be any walk in h from $t \in B$ to $t \in T$. We define the ***left tall set*** of h, denote $M_l(h) = M_l(\omega)$ to be the set of all τ-tall vertices to the left of ω. Namely, $M_l(h)$ is the set of tall vertices inside the disc with boundary $L \cup T[t_l,t] \cup \omega \cup B[b_l,t]$, where L is the "left" portion of the boundary of $\mathcal{D}$, having endpoints t_l and b_l. We similarly define the ***right tall set*** of h, $M_r(h) = M_r(\omega)$, to be the set of all τ-tall vertices to the right of ω. See figure on the right.

We say that h is ***τ-extendable*** from the left if and only if $|\pi_{\mathrm{L,h}}| \leq \tau$ and there is a partitioning class h', such that $|\pi_{\mathrm{L,h'}}| \leq \tau$ and $M_l(h) \subset M_l(h')$. In particular, h is ***τ-saturated*** if it is not τ-extendable and $|\pi_{\mathrm{L,h}}| \leq \tau$.

5.6.4.1 On the left and right geodesics

Lemma 5.6.4. *Let h be a τ-saturated partitioning class and $\pi_{\mathrm{L,h}}$ be its left geodesic, where $\tau \geq d_{\mathcal{H}}(T,B)$. There is a (t_r,b_r)-path homotopic to $\pi_{\mathrm{L,h}}$, whose length is at most 4τ.*

Proof: Let h_{opt} be the partitioning class of the leashes in the optimum solution. Of course, no leash in the optimum solution contains a τ-tall vertex, and therefore, all leashes in the optimal solution are in fact homotopic. Let π_L^* and π_R^* be the extreme leashes in the optimum solution.

Since h is saturated the set $M_l(h)$ is not a proper subset of $M_l\left(h_{\text{opt}}\right)$. If $M_l(h) = M_l\left(h_{\text{opt}}\right)$ then either π_L^* and $\pi_{\text{L,h}}$ are homotopic, and in particular $|\pi_R| = |\pi_R^*| \leq \tau$, or $\pi_{\text{L,h}}$ crosses π_R^*.

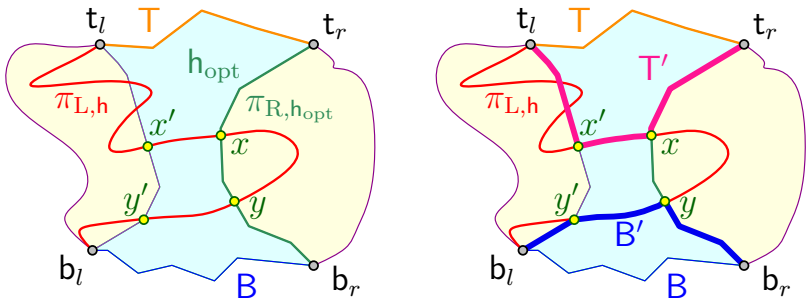


Figure 5.6. Computing a short homotopic path on the right.

Otherwise, the set $M_l(h) \cap M_r\left(h_{\text{opt}}\right)$ is not empty. Again, it follows that $\pi_{\text{L,h}}$ crosses π_R^*.

Now consider the overlayed graph of T, B, $M_l(h)$, π_L^* and π_R^*. This graph has a vertex for each intersection point of the mentioned paths, and two vertices are connected if there is a subpath between them that contains no other vertex. The overlayed graph has a natural embedding and so a natural set of faces. The outer face of the graph contains T and B. Let F_T and F_B be the other faces of the graph that contain T and B, respectively. Further, let $T' = \partial F_T \setminus T$ and $B' = \partial F_B \setminus B$; observe that T' and B' are homotopic to T and B, respectively.

It follows that $\pi_{\text{L,h}}$ is homotopic to $T' \cdot \pi_{\text{L,h}} \cdot B'$. In particular, the shortest path that connects t_r to b_r and is homotopic to $\pi_{\text{L,h}}$ has length at most:

$$|T' \cdot \pi_{\text{L,h}} \cdot B'| \leq |\pi_{\text{L,h}}| + (|\pi_L^*| + |\pi_{\text{L,h}}| + |\pi_R^*|) \leq 4\tau,$$

□

A region that contains no τ-tall vertices can still, potentially, contain τ-tall points (that are not vertices) on in its edges or faces. We next prove that this does not happen in our settings.

Lemma 5.6.5. *Let π_L and π_R be homotopic paths, such that $\max(\pi_L, \pi_R) \leq x$, where $x \geq \tau \geq d_{\mathcal{H}}(T, B)$. Let $\mathcal{D}'$ be the disk with boundary $T \cdot \pi_{\text{R,h}} \cdot B \cdot \pi_{\text{L,h}}$. Then, all the points inside $\mathcal{D}'$ are within distance $O(x)$ to both T and B in $\mathcal{D}'$.*

Proof: We first consider the continuous case. By the assumption of the lemma, the disk $\mathcal{D}'$ has no τ-tall vertices. Furthermore, by the definition of x, we have that the distance of any point on T to B, restricted to paths in $\mathcal{D}'$ is at most δ_1, where $\delta_1 = x + d_{\mathcal{F}}(T, B) \leq 2x$. Indeed, the shortest path from any point on T to B in $\mathcal{D}$, either stays inside $\mathcal{D}'$, or alternatively intersects either $\pi_{\text{L,h}}$ or $\pi_{\text{R,h}}$.

We can now deploy the decomposition of $\mathcal{D}'$ into strips, pockets and chunks as done in 5.5.3.2. Every strip (or a chunk) is being swept by a leash of length at most

$\delta_2 = 2\delta_1 \leq 4x$ (the factor two is because a strip might rise out of a delta), and as such the claim trivially holds for points inside such regions.

Every pocket P has perimeter of length at most $|\partial P| \leq \delta_3 = 2\delta_2 = 8x$ (the perimeter also contains two points of T and B and they are in distance at most δ_2 from each other in either direction along the perimeter).

So consider such a pocket P. Since $\mathcal{D}'$ contains no τ-tall vertices, P does not contain any tall vertex. Let e be an edge in P (or a subedge if it intersects the boundary of P). The two endpoints of e are in P, and such an endpoint is either a (not tall) vertex or it is contained in ∂P. In either case, these endpoints are in distance at most x from ∂P, and as such they are in distance at most $\delta_4 = 2x + |\partial P|/2 = 2x + \delta_2 \leq 6x$ from each other (inside P). We conclude that $|e| \leq \delta_4$, and as such, any point in e is in distance at most $\delta_5 = |e|/2 + x + \delta_2 \leq 3x + x + 8x \leq 12x$ from T and B.

Now, consider any point p in P, and consider the face F that contains it. Since the surface is triangulated, F is a triangle. Clipping F to P results in a planar region F' that has perimeter at most $\delta_6 = 3\delta_4 + |\partial P| \leq 3 \cdot 6x + \delta_3 \leq (18+8)x \leq 26x$ (note, that an edge might be fragmented into several subedges, but the furthest two points along a single edge is at most δ_4 using the same argument as above). As such, the furthest a point of P can be from an edge of P is at most $\delta_7 = \delta_6/2\pi \leq 5x$. As such, the maximum distance of a point of P from either T or B (inside $\mathcal{D}'$) is at most $\delta_5 + \delta_7 \leq 12x + 5x = 17x$.

The discrete case is easy. Any edge of length $\geq 2\tau$ was split, by introducing a middle vertex, which must be τ-tall. As such, the claim immediately holds. □

5.6.4.2 The decision algorithm

Lemma 5.6.6. *Let $\mathcal{D}, n, T, L, B, R, t_l, b_l, \tau$ be as above, and let $X \subseteq V(\mathcal{D})$ be a set of τ-tall vertices. Consider the shortest path σ_l (between t_l and b_l) that belongs to any homotopy class h such that $X \subseteq M_l(h)$. Then, the path σ_l can be computed in $O(n^4 \log n)$ (resp. $O(n \log n)$) time in the continuous (resp. discrete) case.*

Proof: For each vertex of $v \in X$, compute its shortest path ψ_v to L in $\mathcal{D}$. Cut the disk $\mathcal{D}$ along these paths. The result is a topological disk $\mathcal{D}'$. Compute the shortest path ζ in $\mathcal{D}'$ between t_l and b_l.

We claim that $\zeta = \sigma_l$. To this end, consider σ_l and any path ψ_v computed by the algorithm. We claim that σ_l and ψ_v do not cross in their interior. Indeed, if σ_l cross ψ_v an odd number of times, then v is inside the disk $\sigma_l \cdot T \cdot R \cdot B$, which contradict the condition that $v \in X \subseteq M_l(h)$. Clearly, σ_l and ψ_v can not cross in their interiors more than once, because otherwise, one can shorten one of them, which is a contradiction as they are both shortest paths. Thus, σ_l is a path in $\mathcal{D}'$ connecting t_l to b_l, thus implying that ζ is σ_l.

As for the running time, each shortest path computation takes time $O(n^2 \log n)$, in the continuous (resp. discrete) case. The resulting disk has complexity $O(n^2)$,

and computing a shortest path in it takes $O(n^4 \log n)$ time in the continuous case. In the discrete case, computing the paths can be done by collapsing L to a vertex, forbid the shortest path tree edges, and run shortest path algorithm in the remaining graph. Clearly, this takes $O(n \log n)$ time. □

Lemma 5.6.7. *Let $\mathcal{D}$ be a triangulated topological disk with n faces, and T and B be two internally disjoint walks on $\mathcal{D}$'s boundary. Given $\tau > 0$, one can compute a τ-saturated partitioning class, in $O(n^5 \log n)$ (resp. $O(n^2 \log n)$) time, in the continuous (resp. discrete) case.*

Proof: Start with an empty initial set $X = \emptyset$. At each iteration, try adding one of the τ-tall vertices $v \in M^{\tau}$ of $\mathcal{D}$ to X, by using Lemma 5.6.6. The algorithm of *Lemma* 5.6.6 outputs a path σ between t_l and b_l and a set $X' \supset X \cup \{v\}$.

If σ is of length at most τ update X to be the new set X', otherwise reject v. If v is rejected then the left geodesic of any superset of $X \cup \{v\}$ has length larger than τ. It follows that v cannot be accepted in any later iteration, so we do not need to reinspect it. Clearly, after trying all the vertices of M^{τ}, the set X defines the desired saturated class, which can be computed by using the algorithm of Lemma 5.6.6. □

Lemma 5.6.8. *Let $\mathcal{D}$ be a triangulated topological disk with n faces, and T and B be two internally disjoint walks on the boundary of $\mathcal{D}$. Given a real number $x > 0$, one can either:*

(A) Compute a homotopy from T to B of width $O(x \log n)$.

(B) Return that $x < d_{\mathcal{H}}(T, B)$.

The running time of this procedure is $O(n^5 \log n)$ (resp. $O(n^2 \log n)$) in the continuous (resp. discrete) case.

Proof: Assume $x \geq \delta_H = d_{\mathcal{H}}(T, B)$, and we use x as a guess for this value δ_H. Using Lemma 5.6.7, one can compute a x-saturated partitioning class, h. Lemma 5.6.4 implies that both $\pi_{\mathrm{L,h}}$ and $\pi_{\mathrm{R,h}}$ are at most $4x$. Let $\mathcal{D}' \subseteq \mathcal{D}$ be the disc with boundary $T \cup \cup \pi_{\mathrm{L,h}} \cup B \cup \pi_{\mathrm{R,h}}$. By Lemma 5.6.5, all the vertices in $\mathcal{D}'$ are in distance $O(x)$ from T and B (this holds for all points in $\mathcal{D}'$ in the continuous case). That is, there are no $O(x)$-tall vertices in $\mathcal{D}'$. Finally, Lemma 5.6.3 implies that a continuous leash sequence of height $\leq Z = O(x \log n)$ between T and B, inside $\mathcal{D}'$, can be computed.

Thus, if x is larger than $d_{\mathcal{H}}(T, B)$ then this algorithm returns the desired approximation; that is, is a homotopy of width $\leq Z$. If the width of the generated homotopy is however larger than Z (a value that can be computed directly from x), then the value of x was too small. That is, the algorithm fails in this case only if $x < d_{\mathcal{H}}(T, B)$. In the case of such failure, return that x is too small. □

5.6.5 A strongly polynomial approximation algorithm

For a vertex $v \in V(\mathcal{D})$, define cost(v) to be the length of the shortest path between t_l and b_l that has v on its left side. Similarly, for a set of vertices $X \subseteq V(\mathcal{D})$, let

Cost(X) be the length of the shortest path between t_l and b_l that has X on its left side. For a specific v or X, one can compute cost(v) and Cost(X) by invoking the algorithm of Lemma 5.6.6 once.

5.6.5.1 The algorithm

1. **Identifying the tall vertices.** Observe that using the algorithm of Lemma 5.6.8, we can decide given a candidate value δ_H for $d_{\mathcal{H}}(T,B)$ if it is too large, too small, or leads to the desired approximation. Indeed, if the algorithm returns an approximation of values $O(\delta_H \log n)$ but fails for $\delta_H/2$, we know it is the desired approximation.

   So, compute for each vertex $v \in V(\mathcal{D})$ its tallness; that is α_v would be the maximum distance of v to either T or B. Sort these values, and using binary search, compute the vertex w, with the minimum value α_w, such that Lemma 5.6.8 returns a parametrization with homotopic Frechét distance $O(\alpha_w \log n)$. If the algorithm of Lemma 5.6.8 returns that α_w/n is too small of a guess, then $[\alpha_w/n, \alpha_w \log n]$ contains δ_H. In this case, we can use binary search to find an interval $[\gamma/2, \gamma]$ that contains δ_H and use Lemma 5.6.8 to obtain the desired approximation. Similarly, if v is the tallest vertex shorter than w, then we can assume that $\alpha_v n$ is too small of a guess, otherwise we are again done as $[\alpha_v, \alpha_v n]$ contains δ_H.

   As such, in the following, we know that the desired distance δ_H lies in interval $[x,y]$ where $x = \alpha_v n$ and $y = \alpha_w/n$, and for every vertex u of $\mathcal{D}$ it holds that (i) $\alpha_u \leq x/n$, or (ii) $\alpha_u \geq yn$. Naturally, we consider all the vertices that satisfy (ii) as tall vertices, by setting $\tau = 2x/n$. In the following, let M denote the set of these τ-tall vertices.

2. **Computing candidate partitioning classes.** Start with $X_0 = \emptyset$. In the ith iteration, the algorithm computes the vertex $v_i \in M \setminus X_{i-1}$, such that $\text{Cost}(X_{i-1} \cup \{v_i\})$ is minimized, and set $X_i = X_{i-1} \cup \{v_i\}$. Let h_i be the partitioning class having X_i on its left side, and $M \setminus X_i$ on its right side.

3. **Binary search over candidates.** We approximate the homotopic Frechét width of each one of the classes $h_1, \ldots, h_n$. Let x be the minimum homotopic Frechét width computed among these n candidates.

   Next, do a binary search in the interval $[x/n^2, x]$ for the homotopic Frechét distance. We return the smallest width reparametrization computed as the desired approximation.

5.6.5.2 Analysis

Lemma 5.6.9. *(i) For any $X' \subseteq X \subseteq V(\mathcal{D})$, we have* $\text{Cost}(X') \leq \text{Cost}(X)$.

(ii) For any $x \in X \subseteq V(\mathcal{D})$, we have $\text{cost}(x) \leq \text{Cost}(X)$.

(iii) For $X, Y \subseteq V(\mathcal{D})$, we have that $\mathrm{Cost}(X \cup Y) \leq \mathrm{Cost}(X) + \mathrm{Cost}(Y)$.

Proof: (i) Observe that the path realizing $\mathrm{Cost}(X')$ is less constrained than the path realizing $\mathrm{Cost}(X)$, as such it might only be shorter.

(ii) Follows immediately from (i).

(iii) Consider the disk $\mathcal{D}$ and the two paths σ_X and σ_Y realizing $\mathrm{Cost}(X)$ and $\mathrm{Cost}(Y)$, respectively. The close curves $\sigma_x \cup L$ and $\sigma_Y \cup L$ encloses two topological disks. Consider the union of these two disks, and its connected outer boundary $\sigma_{X\cup Y} \cup L$. Clearly, $\sigma_{X\cup Y}$ connects t_l and b_l, and it has all the points of X and Y on one side of it, and finally $|\sigma_{X\cup Y}| \leq |\sigma_X| + |\sigma_Y|$ as $\sigma_{X\cup Y} \subseteq \sigma_X \cup \sigma_Y$. See figure on the right. □

Lemma 5.6.10. *The cheapest homotopic Frechét parametrization computed among $h_1, \ldots, h_n$ has width $O(d_{\mathcal{FH}}(T,B)\, n \log n)$.*

Proof: Consider the set Y that is the subset of tall vertices on the left side of the optimal solution. Let i be the first index such that $Y \subseteq X_i$ and $Y \not\subseteq X_{i-1}$. Let v be any vertex in $Y \setminus X_{i-1}$. By construction, we have that $\mathrm{Cost}\big(X_i\big) \leq \mathrm{Cost}\big(X_{i-1} \cup \{v\}\big)$, and furthermore, for all $j \leq i$, we have that $\mathrm{Cost}\Big(X_j\Big) \leq \mathrm{Cost}\Big(X_{j-1} \cup \{v\}\Big)$, by the greediness in the construction of $X_1, \ldots, X_i$. Now, we have

$$\begin{aligned}
\mathrm{Cost}\big(X_i\big) &\leq \mathrm{Cost}\big(X_{i-1} \cup \{v\}\big) && \text{(by construction of } X_i\text{)}\\
&\leq \mathrm{Cost}\big(X_{i-1}\big) + \mathrm{cost}(v) && \text{(by Lemma 5.6.9 5.6.9)}\\
&\leq \mathrm{Cost}\big(X_{i-1}\big) + \mathrm{Cost}(Y) && \text{(by Lemma 5.6.9 5.6.9)}\\
&\leq (\mathrm{Cost}\big(X_{i-2}\big) + \mathrm{Cost}(Y)) + \mathrm{Cost}(Y) && \text{(applying same argument to } X_{i-1}\text{)}\\
&= \mathrm{Cost}\big(X_{i-2}\big) + 2\mathrm{Cost}(Y)\\
&\leq \cdots \leq i\mathrm{Cost}(Y) \leq n\mathrm{Cost}(Y).
\end{aligned}$$

Now, setting $\tau = \mathrm{Cost}\big(X_i\big)$, it follows that X_i is τ-saturated. Applying Lemma 5.6.4, implies that $|\pi_{\mathrm{R,h_i}}| \leq 4\tau$. Observe, that the disk defined by T, $\pi_{\mathrm{L,h_i}}$, B, $\pi_{\mathrm{R,h_i}}$ can not contain any tall vertex (by construction).

Now, plugging this into Lemma 5.6.3 implies the homotopic Frechét width of h_i (starting with $\pi_{\mathrm{L,h_i}}$ and ending up with $\pi_{\mathrm{R,h_i}}$) is $O(\tau \log n)$, which implies the claim since $\mathrm{Cost}\big(X_i\big) \leq n\mathrm{Cost}(Y) \leq n d_{\mathcal{FH}}(T,B)$. □

5.6.5.3 The algorithm

Theorem 5.6.11. *Let $\mathcal{D}$ be a triangulated topological disk with n faces, and T and B be two internally disjoint walks on the boundary of $\mathcal{D}$. One can compute a homotopic Frechét parametrization of T and B of width $O(d_{\mathcal{FH}}(T,B)\log n)$, where $d_{\mathcal{FH}}(T,B)$ is the homotopic Frechét distance between T and B in $\mathcal{D}$.*

The running time of this procedure is $O(n^6 \log n)$ (resp. $O(n^3 \log n)$) in the continuous (resp. discrete) case.

Proof: For correctness, observe that the algorithmic either found the desired value, or identified correctly the tall vertices. Next, by Lemma 5.6.10, the range the algorithm searches over contains the desired value.

The algorithm requires $O(n^2)$ calls to Lemma 5.6.6, which takes $O(n^6 \log n)$ (resp. $O(n^3 \log n)$) time in the continuous (resp. discrete) case. Then, the algorithm requires Lemma 5.6.3 to compute the homotopic Frechét distance of the classes $h_1, \ldots, h_n$. The algorithm also performs $O(\log n)$ calls to the algorithm of Lemma 5.6.8. □

Chapter 6

Tracing compressed curves

6.1 Introduction

Given a surface combinatorially presented by a triangulation (or more generally an embedded graph), embedded curves can be presented by their interaction with the triangles (faces of the embedded graph). The sequence of edges that a curve crosses is an example of a method to present a sufficiently well-behaved curve on a triangulated surface up to continuous deformation that avoids vertices of the triangulation. Although crossing sequences are simple to deal with, they are usually very long, because of the redundancy in the encoding.

In fact, there are exponentially shorter *implicit* encodings for well-behaved curves on surfaces. Normal coordinates of curves is an example of such encodings, which is an specialization of normal coordinations to present surfaces (see Section 6.2 for an overview). Many algorithms in computational topology owe their efficiency to the compactness of the normal-coordinate representation [4, 23, 24, 107, 163, 164, 166, 169, 179].

On the flip side, it is usually more difficult to compute with a compact presentation of a curve in polynomial time (intuitively, without uncompressing the input). In this chapter, we consider several algorithmic problem about curves (presented by normal coordinates) such as computing the number of components of a curve, deciding whether two given curves are isotopic, and computing algebraic and geometric intersection numbers of pairs of curves. Classical algorithms for these problems require explicit traversal or crossing sequences as input.

The kernel of our algorithms is a method to trace a normal curve in linear time (with respect to the length of its compressed presentation). The output of our tracing method is an alterative cell complex of the underlying manifold, on which the input curve has a compact *explicit* representation. Using that, we design simple natural algorithms for several problems about normal curves, our algorithm are faster than the previously known results in most of the cases; see Section 6.3 for an overview.

We can extend our result to trace a geodesic path on a piece-wise linear surface to find its first self-crossing point. Here the input is a piecewise linear surface described by a set of Euclidian polygons and a collection of rules to identify some pairs of same length edges of the polygons (so that the resulting space is a 2-manifold). The geodesic path is specified by a point inside a polygon and a local direction.

The goal is finding the closest self-crossing point on the geodesic path in the given direction. We refer the interested reader to our paper [78] for a full description of that algorithm.

6.2 Related results

There are several compact methods to present simple (sufficiently well-behaved) curves on triangulated surfaces. For example, given a triangulation of the surface, such a curve curve can be described by listing the number of elementary segments connecting each pair of edges in each triangle; an *elementary segment* is a connected subpath between two consecutive edges of the triangulation. These numbers are called the *normal coordinates* of the curve [102, 127]. Any vector of normal coordinates identifies a unique simple curve (again up to continuous deformation), because there is only one way to fill each triangle with the correct number of elementary segments without intersection. The normal coordinate representation is remarkably compact; only $O(n \log(X/n))$ bits are needed to list the normal coordinates of a curve with X crossings on a triangulated surface with complexity n. Several algorithms in two- and three-dimensional topology owe their efficiency to the compactness of the normal-coordinate representation [4, 23, 24, 107, 163, 164, 166, 169, 179].

Schaefer *et al.* [163, 166, 179] consider several algorithmic questions about normal curves, such as computing the number of components of a curve, deciding whether two given curves are isotopic, and computing algebraic and geometric intersection numbers of pairs of curves. Classical algorithms for these problems require explicit traversal or crossing sequences as input.

By connecting normal coordinates with grammar-based text compression [132, 133, 141, 161] and word equations [65, 152, 159, 160], Schaefer *et al.* developed algorithms whose running times are polynomial in the bit complexity of the normal coordinate vector, which they call the *normal complexity* of the curve. These algorithms rely on a complex algorithm of Plandowski and Rytter [152] to compute compressed solutions of word equations. We are unaware of any precise time analysis, but as Plandowski and Rytter's algorithm uses a nested sequence of quadratic- and cubic-time reductions, its running time is quite high. Štefankovic [179] described simpler algorithms for some of these problems in time *linear* in the normal complexity, or $O(n \log(X/n))$ time in our notation, by reducing them to an elegant algorithm of Robson and Deikert [159, 160] to solve word equations with a certain special structure.

Some of the problems considered by Schaefer *et al.* can also be solved in polynomial time using the polynomial-time *orbit-counting* algorithm of Agol, Hass, and Thurston [4], which was originally designed to compute the number of components of normal *surfaces* in triangulated 3-manifolds in polynomial time, but in fact (like the word-equation algorithms of Schaefer *et al.* [163, 166, 179]) works for similar problems in any dimension. Agol *et al.* do not claim a precise time bound, but a

direct reading of their analysis implies a running time of $O(n^4 \log^3(X/n))$. Dynnikov and Wiest [67] later developed a special case of the orbit-counting algorithm to reconstruct braids from their planar curve diagrams; Dehornoy *et al.* [56] refer to this variant as the *transmission-relaxation method*.

Other compact representations of curves include weighted train tracks [11,12,82, 83,150], Dehn-Thurston coordinates (with respect to a fixed pants decomposition of the surface) [55,82,83,149,184], and compressed intersection sequences [166,179].

6.3 Overview

6.3.1 Tracing

We propose an alternate strategy to efficiently compute with curves on surfaces. Instead of using complex compression techniques to avoid unpacking the crossing sequence of the input curve, our algorithms *modify the underlying cellular decomposition* of the surface so that the curve has a small *explicit* description with respect to the new decomposition. Specifically, given the normal coordinates of a curve γ on a triangulated surface with n edges, we compute a new cellular decomposition of the surface with complexity $O(n)$, called a *street complex*, such that γ is a simple path or cycle in the 1-skeleton. After reviewing some background terminology, we formally define the street complex in Section 6.4; see Figure 6.2 for an example.

At a high level, our algorithm simply *traces* the curve, continuously updating the street complex to reflect the portion of the curve traced so far. A naïve implementation of our tracing strategy runs in $O(X)$ time, where X is the total number of edge crossings; each time the curve enters a triangle by crossing an edge, we can easily determine in $O(1)$ time which of the other two edges of the triangle to cross next. The main result of this paper is a tracing algorithm that runs in $O(n^2 \log X)$ time, an exponential improvement over the naïve algorithm for any fixed surface triangulation.

Our new algorithm relies on two simple ideas. First, we observe that for typical curves, most of the decisions made by the brute-force tracing algorithm are redundant. If a curve enters a triangle Δ between two older elementary segments that leave Δ through the same edge, the new elementary segment must also leave Δ through that edge; see Figure 1.1. The street complex allows us to skip these redundant decisions automatically.

Second, even with redundant decisions filtered out, the naïve algorithm may repeat the same series of crossings many times when the input curve contains a *spiral* [67, 146, 165, 167]. Our algorithm detects spirals as they occur, quickly determines the depth of the spiral (the number of repetitions), and then skips ahead to the first crossing after the spiral. See Figure 6.6 below.

We describe our generic tracing algorithm in Section 6.5 and analyze its running time in Section 6.6. We also describe and analyze a symmetric *untracing* algorithm in Section 6.7, which works backward from the street complex of a curve to its

normal coordinates.

6.3.2 Applications

The street complex allows us to answer several fundamental topological questions about simple curves using elementary algorithms. For example, to determine whether a curve represented by normal coordinates is connected, we can trace one component of the curve, and then check whether the number of edge crossings we encountered is equal to the sum of the normal coordinates. To determine whether a connected normal curve is contractible, we can trace the curve and then apply a $O(n)$-time depth-first search in the dual of the resulting street complex [76]. To find the normal coordinates of a single component of a curve, we can trace just that component, discard the untraced components, and then untrace the street complex.

In Section 6.8, we describe algorithms to solve these and several other related problems for normal curves in $O(n^2 \log X)$ time. All of the problems we consider were previously solved in (large) polynomial time by Schaefer *et al.* [163]; however, our algorithms are significantly faster and simpler. Our algorithms are also faster than the orbit-counting algorithm of Agol *et al.* [4] and more general than Dynnikov and Wiest's transmission-relaxation method [56, 67]. For some of the problems we consider, our algorithms appear to be slower by a factor of n than algorithms described by Štefankovic [179]; however, we optimistically conjecture that this gap can be closed with more careful time analysis.

6.3.3 Computational assumptions

Most of our time bounds are stated functions of two variables: the number n of triangles in the input triangulation and the total crossing number X of the traced curve. We assume that $X = \Omega(n^2)$, since otherwise our analysis yields a time bound slower than the trivial bound $O(n + X)$; this assumption implies that $\log(X/n) = \Theta(\log X)$.

We formulate and analyze our algorithms for normal curves in the standard unit-cost integer RAM with w-bit words, where $w = \Omega(\log X)$; that is, we assume that the *sum* of the normal coordinates can be stored in a single memory word. This assumption implies that all necessary integer arithmetic operations (comparison, addition, subtraction, multiplication, and division) required by our tracing algorithm can be executed in constant time. The $O(n \log X)$ time bound for Štefankovic's word-equation algorithms [159, 160, 179] and the $O(n^4 \log^3 X)$ time bound for the Agol-Hass-Thurston orbit-counting algorithm [4] require the same model of computation.[1] For integer RAMs with smaller word sizes (for example, if the word size is only large enough to the largest *individual* normal coordinate), all these running times increase by at most a polylogarithmic factor in X.

[1] For several of his algorithms, Štefankovic [4] only claims running times on integer RAMs with significantly larger word sizes, but his estimates are unnecessarily conservative.

6.4 Normal coordinates vs. street complex

In this section, we describe two different methods to compactly present normal curves on surfaces, namely normal coordinates and street complex. Normal coordinates is a well-known method that is a special case of the presentation of the surfaces in 3-manifolds. Street complex is another method, that we [78] introduced, to compactly present normal curves; Jaco *et al.* [115, 116] apply a similar idea to present surfaces in 3-manifolds.

6.4.1 Normal curves, normal isotopy, and normal coordinates

Let T be a triangulation of a surface Σ. A properly embedded curve γ in Σ is ***normal*** with respect to T if (1) γ avoids the vertices of T; (2) every intersection between γ and an edge of T is transverse; and (3) the intersection of γ with any triangular face of T is a finite set of disjoint ***elementary segments***: simple paths whose endpoints lie on distinct sides of the triangle. A ***normal isotopy*** between two normal curves is a proper isotopy h such that $h(t,\cdot)$ is a normal curve for all t; or, equivalently, h avoids vertices of T. Two curves are ***normal isotopic***, or in the same ***normal isotopy class***, if there is a normal isotopy between them.

Any normal curve can be identified, up to normal isotopy, by two different vectors of $O(n)$ non-negative integers, where n is the number of faces of T. There are three types of elementary segments within any face Δ, each separating one corner of Δ from the other two; the ***corner coordinates*** of γ list the number of elementary segments of each type in each face of T. The ***edge coordinates*** of γ list the number of times γ intersects each edge of T. See Figure 6.1. We collectively refer to the corner and edge coordinates of a curve as its ***normal coordinates***.[2] Given either normal coordinate representation, it is easy to compute the other representation in $O(n)$ time. The ***total crossing number*** of a normal curve is the sum of its edge coordinates; this number is also equal to the sum of the curve's corner coordinates plus the number of edges in T.

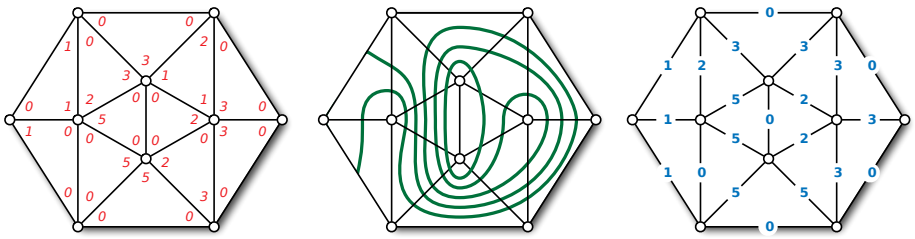


Figure 6.1. Corner and edge coordinates of a normal curve with two components in a triangulated disk.

[2] Schaefer *et al.* [163, 166, 179] refer to the edge coordinates as "normal coordinates", but the standard coordinate system for normal surfaces [102] is a generalization of corner coordinates.

6.4.2 Ports, blocks, junctions, and streets

We now introduce the street complex and its components.

The intersections between any normal curve γ and the edges of any triangulation T partition γ into ***elementary segments*** and partition the edges of T into segments called ***ports***. The ***overlay graph*** $T\|\gamma$ is the cellularly embedded graph whose edges are these elementary segments and ports. Every vertex of $T\|\gamma$ is either a vertex of T or an intersection point of γ and some edge of T. Every face of $T\|\gamma$ is a subset of some face Δ of T. We call each face a ***junction*** if it is incident to all three sides of its containing face Δ, and a ***block*** if it is incident to only two sides of Δ; these are the only two possibilities. Each face of T contains exactly one junction. Each block is bounded either by two elementary segments and two ports, or by one elementary segment and two ports that share a vertex of T.

We call a port ***redundant*** if it separates two blocks; because each face of T contains exactly one junction, each edge of T contains at most two non-redundant ports. Removing all the redundant ports from the overlay graph $T\|\gamma$ merges contiguous sets of blocks into ***streets***. Each street is either a single open disk with exactly two non-redundant ports on its boundary (called the ***ends*** of the street), an open annulus bounded by a trivial component of γ and a vertex of T, or an annulus bounded by two parallel components of γ (in particular, if γ is *reduced*, all streets are of the first type.) For any normal curve γ, the ***street complex*** $S(T,\gamma)$ is the complex of streets and junctions in the overlay $T\|\gamma$. See Figure 6.2. Streets and junctions are two-dimensional analogues of the *product regions* and *guts* of normal surfaces, defined by Jaco, Letscher, and Rubinstein [115, 116].

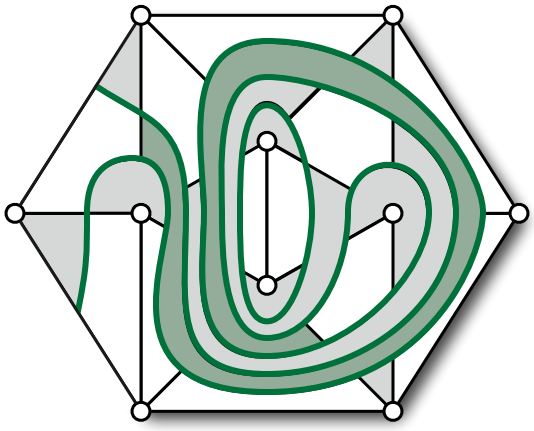


Figure 6.2. The street complex of the normal curve in Figure 6.1. Unshaded faces are junctions; shaded faces are streets; one street is shaded darker (green) for emphasis.

The ***crossing sequence*** of a street is the sequence of edges in the original triangulation T crossed by any path that traverses the street from one end to the other. The ***crossing length*** of a street is the length of its crossing sequence, or equivalently, the number of constituent blocks plus one. To simplify our analysis, we regard any port between two junctions, as well as any boundary port incident to a junction, as

a street with crossing length 1. The sum of the crossing lengths of the streets in any street complex $S(T,\gamma)$ is the total crossing number of γ plus the number of edges in T.

Any normal curve γ' that is disjoint from γ subdivides each port in $S(T,\gamma)$ into smaller ports, each street in $S(T,\gamma)$ into narrower "blocks", and each junction in $S(T,\gamma)$ into blocks and exactly one smaller junction. Removing all redundant ports from this refinement gives us the refined street complex $S(T,\gamma\cup\gamma')$. Conversely, the intersection of γ' with any street or junction in $S(T,\gamma)$ is a set of elementary arcs. There are three types of elementary arcs within any junction, each connecting two of the junction's three ports. The ***junction coordinates*** of γ' list the number of elementary arcs of each type in each junction of $S(T,\gamma)$. Similarly, the ***street coordinates*** of γ' list the number of such arcs within each street of $S(T,\gamma)$. Junction and street coordinates have the same simple linear relationship as corner and edge coordinates; in fact, the normal coordinates of a curve γ are just the junction and street coordinates of γ in the trivial street complex $S(T,\varnothing)$.

Our tracing strategy must handle normal curves that are partially drawn on the surface; we slightly extend our definitions to include such curves. A ***normal path*** is any simple path whose endpoints lie in the interior of edges of T and that can be extended to a normal curve on Σ. Let γ be composed of a normal curve γ' and possibly a normal path π disjoint from γ'. A ***fork*** is the union of two ports that share one of the endpoints of π; for most purposes, we can think of a fork as a degenerate junction. Formally, we call a port *redundant* if it separates two blocks and it is not part of a fork; modified definitions of streets and the street complex follow immediately. The modified street complex $S(T,\gamma)$ clearly still has complexity $O(n)$. See Figure 6.3.

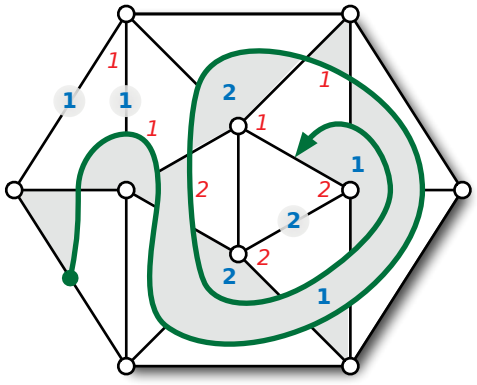


Figure 6.3. A partially traced street complex with **street** and *junction* coordinates. (Zero coordinates are omitted.)

6.4.3 Reduced curves

A normal cycle is ***trivial*** if it bounds a disk in Σ containing a single vertex of T; note that any normal bounds at most two discs, each of them contains at least one vertex

of T. We call a normal curve ***reduced*** if no component of γ is a trivial cycle and no two components of γ are normal isotopic. The following lemma and its corollary, observed by Kneser [127], are helpful during the rest of this chapter.

Lemma 6.4.1. *The components of a properly embedded curve on a surface with genus g and b boundary cycles fall into at most $6g + 6b - 8$ isotopy classes.*

Proof: Fix a properly embedded curve γ on a surface Σ. We separately bound the contractible components, noncontractible cycles, and noncontractible arcs in γ; thus, our analysis is not tight.

Two contractible arcs are isotopic if and only if their endpoints lie on the same boundary cycle of Σ; thus, contractible arcs fall into at most b isotopy classes. All contractible cycles in Σ are isotopic. We conclude that γ has at most $b + 1$ contractible components.

Let $\mathcal{C}$ be a maximal set of pairwise-disjoint noncontractible *cycles* in distinct isotopy classes. Cutting the surface along any cycle leaves its Euler characteristic unchanged. Each component of $\Sigma \setminus \mathcal{C}$ is either a pair of pants bounded by three cycles in $\mathcal{C}$ or an annulus bounded by a cycle in $\mathcal{C}$ and a boundary cycle of Σ. A pair of pants has Euler characteristic -1; an annulus has Euler characteristic 0; and each annular component of $\Sigma \setminus \mathcal{C}$ contains exactly one boundary cycle of Σ. Thus, $\Sigma \setminus \mathcal{C}$ consists of exactly $-\chi(\Sigma) = 2g + b - 2$ pairs of pants and b annuli, which implies that $|\mathcal{C}| = (3(2g + b - 2) + b)/2 = 3g + 2b - 3$.

Similarly, let $\mathcal{A}$ be a maximal set of pairwise-disjoint noncontractible *arcs* in distinct isotopy classes. Each component of $\Sigma \setminus \mathcal{A}$ is a disk bounded by exactly three arcs in $\mathcal{A}$ and three boundary arcs. Contracting each boundary cycle of Σ to a point transforms $\mathcal{A}$ into a b-vertex triangulation of a surface of genus g with no boundary. Thus, Euler's formula implies that $|\mathcal{A}| = 3g + 3b - 6$. □

Corollary 6.4.2. *A reduced normal curve in a surface triangulation with n triangles has at most $\lfloor (3n - 1)/2 \rfloor = O(n)$ components.*

In this chapter we restrict our attention to the reduced curves, because of the following reasons. First, the street complex of any non-reduced curve γ contains annular faces, which would complicate our algorithms (but probably not seriously); observe that the street complex of a reduced curve is cellular. More importantly, the street complex of a non-reduced curve can have arbitrarily high complexity, since the curve can have arbitrarily many components. Fortunately, as we argue in Section 6.8, it is easy to avoid tracing trivial components or more than one component in the same normal isotopy class.

By construction, the components of any reduced normal curve γ appear as disjoint paths and cycles in the 1-skeleton of the street complex. Although the complexity of the overlay graph $T \| \gamma$ can be arbitrarily large, even when the curve γ is connected, the street complex $S(T, \gamma)$ of a reduced normal curve is never more than a constant factor more complex than the original triangulation.

Lemma 6.4.3. *Let T be a surface triangulation with n triangles. For any **reduced** normal curve γ in T, the street complex $S(T,\gamma)$ has complexity $O(n)$.*

Proof: Each triangle of T contains exactly one junction of $S(T,\gamma)$, it follows that the total number of junctions is $O(n)$. Each junction is adjacent to at most 3 streets, so the number of streets is $O(n)$. Each junction is adjacent to at most 6 vertices and each vertex is either adjacent to a junction or it is a vertex of T, that is the number of vertices of $S(T,\gamma)$ is $O(n)$. Finally, each edge of $S(T,\gamma)$ is on the boundary of a street or a junction, and so the number of edges is $O(n)$ as well. □

6.5 Tracing connected normal curves

In this section, we describe our algorithm to trace *connected* normal curves. Given a triangulation T of an orientable surface Σ and the corner and edge coordinates of a connected normal curve γ, our tracing algorithm computes the street complex $S(T,\gamma)$. We extend our algorithm to arbitrary *reduced* curves in Section 6.6, and we consider arbitrary normal curves in Section 6.8.

Our algorithm maintains a normal subpath π of γ that is growing at one end, along with the street complex $S(T,\pi)$ and the junction and street coordinates of the complementary path $\gamma \setminus \pi$. If γ is an arc, we trace it from one endpoint to the other. If γ is a cycle, we trace it starting at some intersection point with an edge of T. Initially, π is an arbitrary crossing point, which splits some edge into a fork.

6.5.1 Steps

In each ***step*** of our algorithm, we extend the path π through one junction or fork, and then through one street, updating both the street complex and the junction and street coordinates. We call the streets that contain the moving endpoint of π the left and right ***active*** streets.

Suppose π is about to enter a junction. We call the streets adjacent to the junction but not the endpoint of π the *left exit* and the *right exit*. Suppose the local junction coordinates are a, b, and c, and the active street coordinates are l and r, as shown in Figure 6.4. These coordinates satisfy the equation $l + r + 1 = a + c$, so either $l < a$ or $r < c$. If $l < a$, we extend π through the junction and through its left exit into the next junction; the left active street grows to the end of the left exit, and the left exit becomes the new right active street. We call this case a ***left turn***; the symmetric case $r < c$ is called a ***right turn***. In either case, we update the street and junction coordinates as shown in Figure 6.4. A similar case analysis applies when π crosses a fork; see Figure 6.5.

The tracing algorithm ends when π hits either the boundary of Σ or the starting point of the trace. In all other cases, each step makes one active street longer, replaces the other active street, and makes the new active street narrower. All necessary operations for a single step—comparing and updating the junction and street coordinates and updating the street complex—can be performed in $O(1)$ time.

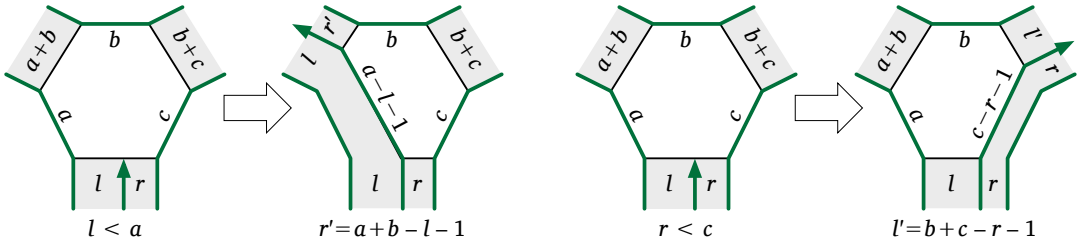


Figure 6.4. Tracing a curve through a junction.

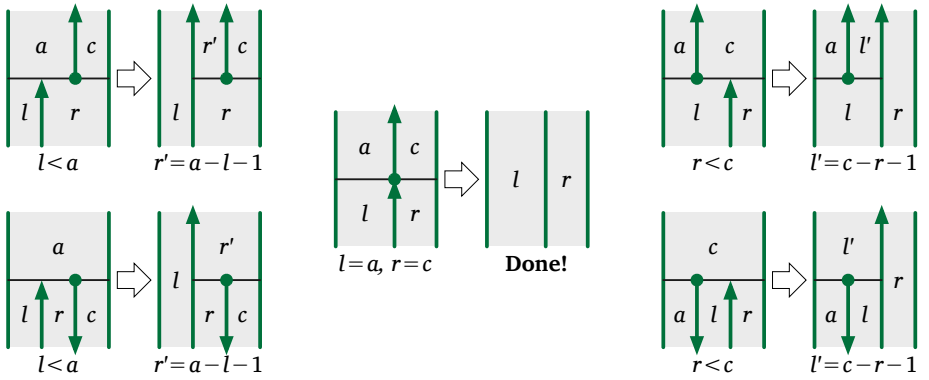


Figure 6.5. Tracing a curve through a fork.

6.5.2 Phases and spirals

Unfortunately, executing each step by brute force is not necessarily efficient. To improve the brute-force algorithm, we more coarsely partition the tracing process into ***phases***. Each phase is a maximal sequence of either left turns or right turns. Every step in a phase consisting of left turns extends the same left active street; similarly, every step in a right phase extends the same right active street. In either case, each phase extends a single ***active street***.

During each phase, we maintain a sequence of *directed* streets and junctions traversed during that phase. If the growing path π ever enters a street for the second time, in the same direction, during the same phase, then π has entered a ***spiral***. The reentered street must be the first street traversed during the current phase; for the remainder of the phase, π repeatedly traverses the same sequence of directed streets and junctions. The ***length*** of a spiral is the total number of streets it traverses, counted with multiplicity, and the ***depth*** of the spiral is the number of times it repeats the *entire* sequence of directed streets and junctions. If the spiral has length ℓ and traverses m distinct directed streets, its depth is $\lceil \ell/m \rceil - 1$. See Figure 6.6.

Instead of tracing the spiral step by step, we compute the depth of the spiral directly in $O(m)$ time as follows. Let $J_0, J_1, \ldots, J_{m-1}$ be the junction coordinates modified during the first iteration of the spiral. Let w denote the ***width*** of the active street, defined as the corresponding street coordinate plus 1. The depth of the spiral is $d = \min_i \lfloor J_i/w \rfloor$, and the spiral ends at the first junction whose coordinate J_i is

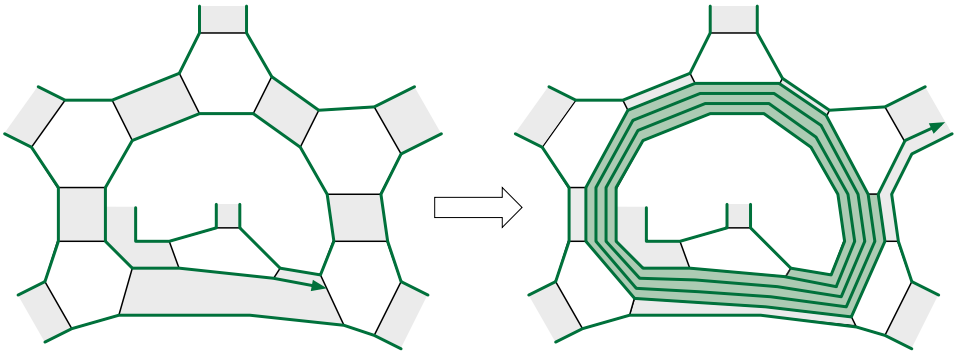


Figure 6.6. A spiral with length $\ell = 16$ and depth $d = 3$ through $m = 5$ distinct streets, plus one step of the next phase.

smaller than dw. Once we compute d, we can update the street complex $S(T, \pi)$ and the appropriate street and junction coordinates in $O(m)$ time. In particular, as long as the depth of the spiral is at least 2, the combinatorial structure of $S(T, \pi)$ (the 1-skeleton and the rotation system encoding its embedding in Σ) depends only on the last ℓ mod m steps of the spiral.

The lengths and widths of the streets, as well as junction and street coordinates of the remainder of the curve, can all be updated in $O(m)$ time. The length of the active street increases by d times the total length of the m distinct directed streets in the spiral, plus the total length of the last ℓ mod m streets; no other street changes length. Each *undirected* street in the spiral is traversed d, $d+1$, $2d$, $2d+1$ or $2d+2$ times, depending on whether the street is traversed in one or both directions, and which of those traversals occur in the last ℓ mod m steps of the phase. We can compute all such numbers in $O(m)$ time, after which updating the widths of the streets traversed by the spiral is straightforward.

The crude upper bound $m = O(n)$ immediately implies that each phase of our tracing algorithm can be executed in $O(n)$ time. We analyze the number of phases, as a function of the total crossing number of the traced curve, in Section 6.6.

6.5.3 History

For some applications of our tracing algorithm, it is useful to maintain the ***history*** of the street complex, which records the evolution of each street during the algorithm's execution. For each phase of the tracing algorithm, the history records the tuple $(a;\ \ell;\ m;\ i_0, i_1, \ldots, i_{m-1})$, where

- a is the index of the street that is active for the entire phase;
- ℓ is the number of steps in the phase;
- m is the number of distinct directed streets traversed during the phase; and
- $i_0, i_1, \ldots, i_{m-1}$ are the indices of these m directed streets in the order they are traversed.

The resulting history is equivalent to a context-free grammar whose terminals correspond to the edges of T and *most* of whose productions have the following form, where $d = \lceil l/m \rceil - 1$:

$$X_a \to X_b \, (X_{i_0} X_{i_1} \cdots X_{i_{m-1}})^d \, X_{i_0} X_{i_1} \cdots X_{i_{(\ell-1) \bmod m}}$$
$$\overline{X}_a \to \overline{X}_{i_{(\ell-1) \bmod m}} \cdots \overline{X}_{i_1} \overline{X}_{i_0} (\overline{X}_{i_{m-1}} \cdots \overline{X}_{i_1} \overline{X}_{i_0})^d \, \overline{X}_b.$$

The language of each non-terminal X_i is a single string, recording the crossing sequence of the street at the end of some phase. In the example above, X_a is the crossing sequence of the *active* street just *after* the phase ends; X_b is the crossing sequence of the active street just *before* the phase begins; and $\overline{X}_i$ denotes the reversal of X_i. If the same street is traversed in both directions during a phase, we will have $X_{i_j} = \overline{X}_{i_k}$ for some indices $j \neq k$, so both the forward and reverse productions are necessary; otherwise, the indices i_j are distinct. The grammar also contains terminal productions of the form $X_i \to e_i$ and $\overline{X}_i \to e_i$ for each edge e_i in the input triangulation.

This context-free grammar can be transformed into Chomsky normal form by replacing each production in the form above with $O(m + \log d)$ productions of the form $A \to B\,C$. (Chomsky normal form grammars for one-string languages are sometimes called *straight-line programs* [119] or *grammar-based codes* [122].) Thus, our history data structure is equivalent to the *compressed intersection sequence* constructed by Schaefer *et al.* [166, 179]. We analyze the complexity of our history data structure and the resulting compressed intersection sequence in the next section.

6.6 Analysis

We now bound the running time of our tracing algorithm. In Section 6.6.1, we bound the time required to trace a *connected* normal curve; we extend our analysis to curves with multiple components in Section 6.6.2. Throughout our analysis, we let N denote the current number of streets in the evolving street complex; although N changes during the algorithm's execution, Lemma 6.4.3 implies that $N = \Theta(n)$ at all times.

Our analysis can be viewed as a generalization of Lamé's classical analysis of Euclid's GCD algorithm in terms of Fibonacci numbers [130, 174]. This connection is not a coincidence; for tracing geodesics on a minimal triangulation of the torus, our algorithm actually reduces to Euclid's algorithm [78]. In particular, handling each phase in $O(n)$ time, instead of constant time per step, generalizes the use of division in Euclid's algorithm instead of repeated subtraction. Euclid's algorithm is invoked explicitly by the orbit-counting algorithm of Agol *et al.* [4] and by the compressed pattern-matching algorithms [119, 141, 161] underlying the results of Schaefer *et al.* [166, 179]. See also related results of Moeckel [142] and Series [171, 172] on encoding (infinite) geodesics in surfaces of constant curvature by continued

fractions.

In retrospect, our analysis (at least for connected curves) is nearly identical to Dynnikov and Weist's analysis of their transmission-relaxation method [56, 67], although the algorithms themselves appear to be quite different. In particular, the potential function Φ in the proof of Lemma 6.6.1 closely resembles their definition of the *AHT-complexity* of a braid (named after Agol, Hass, and Thurston). Dehornoy *et al.* [56, page 196] draw a similar analogy between their approach and the fast Euclidean algorithm.

6.6.1 Abstract tracing

In each phase of our tracing algorithm, the crossing length of the active street increases by the sum of the crossing lengths of the other traversed streets, counted with appropriate multiplicity. The algorithm AbstractTrace, shown in Figure 6.7, abstractly models this growth.

```
AbstractTrace(N):
  for j ← 1 to N
      x[j] ← 1
  a ← 1
  while not done
      choose an integer m ∈ [N]
      choose an integer ℓ ≥ m
      choose a vector (i_0, i_1, ..., i_{m-1}) ∈ [N]^m
      d ← ⌈ℓ/m⌉ − 1
      for j ← 0 to m − 1
          x[a] ← x[a] + d · x[i_j]
      for j ← 0 to (ℓ − 1) mod m
          x[a] ← x[a] + x[i_j]
      a ← i_{(ℓ−1) mod m}
```

Figure 6.7. Our abstract tracing algorithm.

AbstractTrace maintains an array $x[1..N]$ of positive integers, corresponding to the crossing lengths of the streets maintained in our tracing algorithm, along with the index a of the current active street. Each iteration of the outer loop of AbstractTrace models a phase of our tracing algorithm. The inner loops update the crossing length $x[a]$ of the active street as the curve traverses a spiral of length ℓ and depth d, containing m distinct streets whose indices are in the vector $(i_0, i_1, \ldots, i_{m-1})$. The last street traversed in the current phase becomes the active street for the next phase. For purposes of analysis, we assume that the termination condition for the outer loop and the parameters ℓ, m, and $(i_0, i_1, \ldots, i_{m-1})$ of each iteration are determined ***by a malicious adversary*** instead of the topology of the input curve.

To prove an upper bound on the number of phases of AbstractTrace, we can assume conservatively that $m = 1$ in every phase; equivalently, we can ignore the contribution to the active street's crossing length from all but the last street in every spiral. Thus, we consider the simpler algorithm SimpleTrace shown in Figure 6.8. The new variable δ is the number of times the last street in the spiral is traversed;

specifically, $\delta = d$ if ℓ/m is an integer and $\delta = d+1$ otherwise. The other new variable Δ is used only in the analysis.

```
SIMPLETRACE(N):
  for j ← 1 to N
      x[j] ← 1
  Δ ← 0
  a ← 1
  while not done
      choose an index i ∈ [N]
      choose an integer δ ≥ 1
      x[a] ← x[a] + δ · x[i]
      Δ ← Δ + lg(δ + 1)
      a ← i
```

Figure 6.8. A simplified tracing algorithm for analysis.

Lemma 6.6.1. *At the end of each iteration of* SIMPLETRACE*, we have* $\Delta \le 2\sum_{i=1}^{N} \lg x[i]$.

Proof: Consider the potential function $\Phi := 2\sum_{i=1}^{N} \lg x[i] - \lg x[a]$. Initially we have $\Phi = 0$. There are two cases to consider, depending on whether $x[a]$ is smaller or larger than $x[i]$ at the start of each iteration of the loop.

- If $x[a] \le x[i]$, then the assignment $x[a] \leftarrow x[a] + \delta \cdot x[i]$ increases Φ by at least $\lg(\delta+1)$, and the assignment $a \leftarrow i$ does not decrease Φ.
- If $x[a] \ge x[i]$, then the assignment $x[a] \leftarrow x[a] + \delta \cdot x[i]$ does not decrease Φ, and the assignment $a \leftarrow i$ increases Φ by at least $\lg(\delta+1)$.

In both cases, Φ increases by at least $\lg(\delta+1)$ in each iteration. It immediately follows by induction that $\Delta \le \Phi \le 2\sum_{i=1}^{N} \lg x[i]$ at the end of every iteration. □

Lemma 6.6.2. *ABSTRACTTRACE*(N) *runs for at most* $2L = O(N \log X)$ *phases, where* L *is the final value of* $\sum_{i=1}^{N} \lg x[i]$ *and* X *is the final value of* $\sum_{i=1}^{N} x[i]$.

Proof: To maximize the number of phases, we assume that $\ell = 1$ in every phase. This assumption allows us to simplify the execution to an instance of SIMPLETRACE where $\delta = 1$ in every phase, and therefore Δ is simply the number of phases. Lemma 6.6.1 implies that the algorithm terminates after at most $2L$ phases. The parameter L is maximized as a function of N and X when $x[i] = X/N$ for all i. (Our assumption that $X = \Omega(n^2)$ implies that $\log(X/N) = \Theta(\log X)$.) □

The trivial inequality $m \le N$ now implies the following time bound:

Corollary 6.6.3. *ABSTRACTTRACE*(N) *runs in* $O(NL) = O(N^2 \log X)$ *time, where* L *is the final value of* $\sum_{i=1}^{N} \lg x[i]$ *and* X *is the final value of* $\sum_{i=1}^{N} x[i]$.

Theorem 6.6.4. *Let* Σ *be a surface composed of* n *triangles, and let* γ *be a* ***connected*** *normal curve in* Σ *with total crossing number* X*. Given the normal coordinates of* γ*, we can trace* γ *in* $O(n^2 \log X)$ *time.*

There is an interesting tension between the two steps of our analysis. To bound the number of phases in Lemma 6.6.2, we conservatively assume that each phase traverses only a constant *number* of streets; however, to bound the total number of steps in Corollary 6.6.3, we conservatively assume that each phase traverses a constant *fraction* of the streets. Despite this tension, both bounds are asymptotically tight in the worst case, at least when X is sufficiently large.

Lemma 6.6.5. *AbstractTrace(N) executes $\Omega(N \log X)$ phases in the worst case.*

Proof: Suppose the adversary chooses $i = (a \bmod n) + 1$ and $\delta = 1$ in every phase of SimpleTrace. An easy inductive argument implies that for any integer $r \geq 1$, at the end of $r \cdot (n-1)$ phases we have $x[i] \leq 2^r$ for all i. Thus, SimpleTrace must perform at least $(N-1)\lg(X/N) = \Omega(N \log X)$ iterations before $\sum_i x[i] = X$. □

Lemma 6.6.6. *AbstractTrace(N) runs in $\Omega(N^2 \log X)$ time in the worst case, assuming $X = \Omega(N^{2+\varepsilon})$ for some $\varepsilon > 0$.*

Proof: Suppose $N = 2k$ for some integer $k \geq 2$, and in every phase of AbstractTrace, the adversary chooses $\ell = m = k+1$ (and therefore $d = 0$) and $(i_0, i_1, \ldots, i_k) = (k+1, k+2, \ldots, 2k, (a \bmod k) - 1)$. In other words, the adversary mimics the strategy described in the previous proof in the lower half $x[1..k]$ of the array, but uses the upper half $x[k+1..2k]$ to add k additional steps to each phase. The values in $x[k+1..2k]$ never change; at all times, $x[i] = 1$ for all $i > k$. Thus, the additional steps have very little impact on the growth of the sum $\sum_i x[i]$.

A straightforward inductive argument implies that for any integer $r \geq 1$, at the end of $r \cdot (k-1)$ phases, we have $\sum_{i=1}^{k} x[i] < (2^r - 1)k^2 + k < 2^r k^2 - k$ and therefore $\sum_{i=1}^{N} x[i] < 2^r n^2/4$. Thus, AbstractTrace must execute at least $(N-1)\lg(4X/N^2) = \Omega(N \log X)$ phases before $\sum_{i=1}^{N} x[i] = X$. Each phase requires $\Omega(N)$ time. □

We leave open the possibility that our analysis is *not* tight for instances that actually arise from tracing normal curves on triangulated surfaces. We conjecture that Lemma 6.6.2 is still tight in this context, but that Corollary 6.6.3 is not.

6.6.2 Tracing reduced curves

Now consider the more general case where γ is a *reduced* curve, possibly with more than one component. (For the applications we describe in Section 6.8, this is the most general case we need to consider.) Our tracing algorithm requires little modification to handle these curves; we simply trace the components one at a time, in arbitrary order. Each component refines the street complex defined by the previous components. Lemma 6.4.3 immediately implies that the resulting algorithm runs in $O(n^3 \log X)$ time, but this time bound can be improved with more careful analysis.

Theorem 6.6.7. *Let Σ be a surface composed of n triangles, and let γ be a **reduced** normal curve in Σ with total crossing number X. Given the normal coordinates of γ, we can trace every component of γ in $O(n^2 \log X)$ time.*

Proof: Consider the effect of ending one component and starting another on the vector of crossing lengths modeled by the array $x[1..N]$ in SIMPLETRACE. When we begin tracing a new cycle component, we split some street into three smaller streets by introducing a fork; one of the three new streets becomes the active street for the first phase of the new component. This update can be modeled in SIMPLETRACE by adding the following lines:

```
if starting a cycle:
    choose an index i ∈ [N]
    choose an integer y ∈ [x[i]]
    x[i] ← x[i] − y + 1
    x[N + 1] ← y
    x[N + 2] ← y
    N ← N + 2
    a ← N + 2
```

When we finish tracing a cycle component, we merge the four streets adjacent to the initial fork into two longer streets; see the center of Figure 6.4. This update can be modeled in SIMPLETRACE by adding the following lines:

```
if ending a cycle:
    choose an index j ∈ [N]
    choose an index k ∈ [N]
    x[j] ← x[j] + x[N − 1] − 1
    x[k] ← x[k] + x[N] − 1
    N ← N − 2
```

Similarly, when we begin tracing a new arc component, we split some street (ending on the boundary of Σ) into two narrower streets. This update can be modeled in SIMPLETRACE by adding the following lines:

```
if starting an arc:
    choose an index i ∈ [N]
    x[N + 1] ← x[i]
    N ← N + 1
    a ← N + 1
```

No additional changes are necessary when we end an arc component. Again, for purposes of analysis, we assume that the decision of when to end one component and begin another, whether each new component is an arc or a cycle, and the array elements involved in starting or ending a component are all chosen ***adversarially*** instead of being determined by the topology of a curve.

Altogether, ending one component and starting a new one decreases the potential function Φ by at most $O(\log X)$. An easy modification of the proof of Lemma 6.6.1 now implies that after each iteration of SIMPLETRACE, we have $\Delta \le 2\sum_{i=1}^{N} \lg x[i] +$

$O(t \lg X)$, where t is the number of components we have completely traced so far. Lemma 6.4.1 implies that any reduced normal curve has $O(n)$ components. We conclude that SIMPLETRACE(N) executes at most $O(N \log X) = O(n \log X)$ phases; each phase trivially requires $O(n)$ time. □

When we trace curves with multiple components, we also record the start and end of each component in the tracing history. We omit the straightforward but tedious details.

6.7 Untracing

Several of the problems we consider ask for the normal coordinates of one or more components of the input curve, with respect to the input triangulation. These coordinates can be recovered from the street complex and some additional information, essentially by running the tracing algorithm backward. We emphasize that recovering the normal coordinates of a curve from the street complex alone is impossible; two curves may have combinatorially isomorphic street complexes even if they are not normal isotopic.

6.7.1 Untracing from history

The simplest method to untrace a curve uses the full history of the street complex, as defined in Section 6.5.3. The normal coordinates of any normal curve γ can be recovered from a straight-line program of length T encoding the crossing sequences of γ's components, by straightforward dynamic programming, in $O(nT)$ time [90, 166, 179].

Our untracing algorithm maintains the street coordinates of the already-untraced components in the devolving street complex. Initially, all street coordinates are equal to zero; when the curve is completely untraced, the streets degenerate to edges, and the street coordinates are the required edge coordinates. We can then easily recover the corner coordinates in $O(n)$ time.

Lemma 6.7.1. *Let Σ be a surface composed of n triangles, let γ be a reduced normal curve in Σ with total crossing number X, and let λ be the union of any subset of components of γ. Given the street complex $S(T, \gamma)$* ***and its history****, we can compute the normal coordinates of λ with respect to T in $O(n^2 \log X)$ time.*

Proof: Our untracing algorithm maintains an array $st[1..N]$ of street coordinates, initially all equal to zero, and a bit ϕ that indicates whether we are currently untracing a component of λ. We consider the phases stored in the history in reverse order. To undo a phase with parameters $(a; \ell; m; i_0, i_1, \ldots, i_{m-1})$, we update the street coordinates as follows:

```
d ← ⌈ℓ/m⌉ − 1
for j ← 0 to m − 1
    st[i_j] ← st[i_j] + d · (st[a] + φ)
for j ← 0 to (ℓ − 1) mod m
    st[i_j] ← st[i_j] + (st[a] + φ)
```

(Compare with the AbstractTracing algorithm in Figure 6.7.) Some additional bookkeeping is required at the beginning and end of each component of γ; we omit the straightforward but tedious details. Note that the street coordinates $st[\cdots]$ do not actually change until we start untracing a component of λ. When the algorithm ends, the array $st[\cdots]$ contains the edge coordinates of λ; we can then easily recover the corner coordinates of λ in $O(n)$ time.

Since we spend $O(m)$ time untracing each phase, the total time to untrace the entire curve is the same as the time spent tracing the curve, up to small constant factors. The $O(n^2 \log X)$ time bound now follows directly from Theorem 6.6.7. □

6.7.2 Untracing without history

Even without the complete tracing history, we can untrace a curve γ given only the crossing lengths of every street in street complex $S(T, \gamma)$. In fact, it is not necessary to follow the tracing algorithm backward; we can untrace the components of γ in any order, starting each cycle component at any crossing.

Lemma 6.7.2. *Let Σ be a surface composed of n triangles, let γ be a reduced normal curve in Σ with total crossing number X, and let λ be the union of any subset of components of γ. Given the street complex $S(T, \gamma)$* ***and the crossing length of every street****, we can compute the normal coordinates of λ with respect to T in $O(n^2 \log X)$ time.*

Proof: Our untracing algorithm maintains the devolving street complex, its associated street and junction coordinates (all initially zero), and an array $x[1..N]$ storing the crossing lengths of each street. Our algorithm untraces every component of $\gamma \setminus \lambda$ (in arbitrary order), resets all street and junction coordinates to 0, and then untraces the components of λ (again in arbitrary order). When the algorithm terminates, all crossing lengths are equal to 1, and the street and junction coordinates are just the normal coordinates of λ.

First consider the untracing process for a single component of γ. Following the intuition of the tracing algorithm, we maintain a normal subpath π that is *shrinking* from one end toward the other. The last segment of π either separates two streets or separates a street and a junction. We can easily remove the last segment of and update the appropriate street coordinates and crossing lengths in $O(1)$ time, by time-reversing the case analysis in Figures 6.4 and 6.5.

To complete the proof, it remains only to prove that we can untrace any spiral of any depth through m distinct streets in $O(m)$ time. The last segment of π is a spiral

if and only if the junction incident to the end of the active street is also incident to the active street along another edge; see Figures 6.6 and 6.9. This condition can be tested easily in constant time at each step.

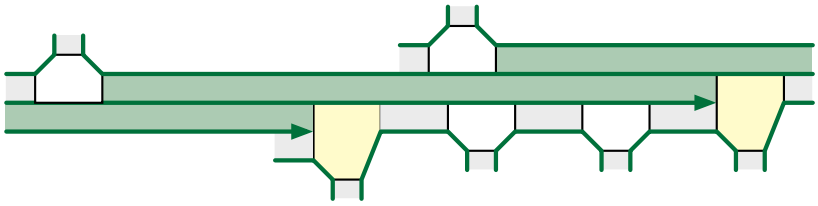


Figure 6.9. After tracing a spiral, the active street is incident to itself at the terminal junction.

Without loss of generality, suppose the street to the left of the last segment of π has greater crossing length than the street to the right, as in Figure 6.9. The active street a is left of the last segment of π, and the m directed streets and junctions traversed by the spiral are incident to the right side of the active street. Thus, we can recover the number m and indices $i_0, i_1, \ldots, i_{m-1}$ of the relevant streets in $O(m)$ time by traversing π backward until some edge is incident on the left. The depth of the spiral is

$$d := \left\lfloor \frac{x[a]}{\sum_{j=0}^{m-1} x[i_j]} \right\rfloor.$$

To untrace d complete turns of the spiral, we add $d \cdot (st[a] + 1)$ to the m relevant street and junction coordinates (where $st[a]$ is the street coordinate of the active street) and subtract $d \cdot \sum_{j=0}^{m-1} x[i_j]$ from the active crossing length $x[a]$. We then untrace the last $\ell \bmod m$ steps of the spiral by brute force in constant time each. Although computing the length ℓ of the spiral is straightforward, it is not actually necessary. The total time to untrace the entire spiral is $O(m)$, as required. □

6.7.3 Abstract untracing

We can also analyze our tracing algorithm directly by considering the growth of the street coordinates, just as we analyzed the forward tracing algorithm by the evolution of crossing lengths. Moreover, our tracing and untracing algorithms have the same running time (up to constant factors), we obtain a new analysis of our *tracing* algorithm. Although our backward analysis leads to the same asymptotic time bound $O(n^2 \log X)$, we obtain more refined bounds for *connected* normal curves in terms of the bit-complexity of the normal coordinates. As in Section 6.6, $N = \Theta(n)$ denotes the number of streets in the current street complex.

First, suppose we are untracing a *connected* normal curve. Again, we ignore the actual topology of the curve and consider instead the abstract algorithm shown in Figure 6.10.

The values in the array $st[1..N]$ correspond to the street coordinates of the N streets. At the end of each backward phase, the current *a*ctive street becomes one of the streets traversed (and therefore widened) in the next phase; we re-index

```
AbstractUntrace(N):
  for j ← 1 to N
      st[j] ← 0
  i_0 ← 1
  while not done
      choose an integer a ∈ [N]
      choose an integer m ∈ [N]
      choose an integer ℓ ≥ m
      choose a vector (i_1, ..., i_{m-1}) ∈ [N]^{m-1}
      d ← ⌈ℓ/m⌉ − 1
      for j ← 0 to m − 1
          w[i_j] ← w[i_j] + d · (st[a] + 1)
      for j ← 0 to (ℓ − 1) mod m
          w[i_j] ← w[i_j] + (st[a] + 1)
      i_0 ← a
```

Figure 6.10. Our abstract untracing algorithm.

the streets in each spiral so that i_0 is always the index of the previous active street. As in the forward analysis, we conservatively assume that the parameters of each phase and the termination condition for the main loop are determined ***adversarially*** instead of by the topology or tracing history of the curve.

As in the forward analysis, to maximize the number of phases, we can assume conservatively that $m = 1$ in every phase, which simplifies the abstract algorithm to the form shown in Figure 6.11. To simplify the algorithm further, we work with an array $w[1..N]$ of street *widths*, where $w[i] = st[i] + 1$ for all i. Again, we introduce a new variable Δ strictly for purposes of analysis. Except for variable names, SimpleUntrace is *identical* to our earlier algorithm SimpleTrace, so our earlier analysis applies immediately.

```
SimpleUntrace(n):
  for j ← 1 to n
      w[j] ← 1
  Δ ← 0
  i ← 1
  while not done
      choose an index a ∈ [n]
      choose an integer δ ≥ 1
      w[i] ← w[i] + δ · w[a]
      Δ ← Δ + lg(δ + 1)
      i ← a
```

Figure 6.11. Our simplified abstract untracing algorithm; compare with Figure 6.8.

Lemma 6.7.3. *AbstractUntrace(N) runs for at most $2W = O(N \log X)$ phases and $O(nW) = O(n^2 \log X)$ total time, where W is the final value of $\sum_{i=1}^{N} \lg w[i]$ and X is the final value of $\sum_{i=1}^{N} w[i]$.*

Again, both bounds in Lemma 6.7.3 are tight in the worst case.

Ignoring lower-order terms, the parameter W is the number of bits needed to store the edge coordinates of the traced curve γ; Schaefer *et al.* [163, 166, 179]

call W the ***normal complexity*** of γ. Recall from Section 6.6.1 that L is the total number of bits needed to store the crossing lengths in the street complex $S(T,\gamma)$. Both W and L are between $\Omega(n+\log X)$ and $O(n\log X)$, which implies the crude bounds $W = O(nL)$ and $L = O(nW)$. In fact, these crude bounds are tight in the worst case, even for actual curves; we leave the proof as an amusing exercise for the reader.

Corollary 6.7.4. *Let Σ be a surface composed of n triangles, and let γ be a **connected** normal curve in Σ. Given the normal coordinates of γ, we can trace γ in $O(n \cdot \min\{L, W\})$ time, where W is the total bit-length of the normal coordinates of γ, and L is the total bit-length of all crossing lengths in the resulting street complex $S(T,\gamma)$.*

The backward analysis can be extended to disconnected reduced curves, exactly as in Section 6.6. However, since the resulting time bound does not improve our earlier analysis, we omit further details.

6.8 Normal coordinate algorithms

In this section, we describe efficient algorithms for several problems involving normal curves represented by their normal coordinates. For each of our algorithms, the input consists of a surface Σ composed of n triangles and the edge and corner coordinates of either one or two normal curves with total crossing length at most X.

6.8.1 One component

Theorem 6.8.1. *Let Σ be a surface composed of n triangles, and let γ be a normal curve in Σ with total crossing length X, represented by its normal coordinates. We can determine whether γ is connected in $O(n^2 \log X)$ time.*

Proof: The input curve γ is connected if and only if, after tracing an arbitrary component of γ, every street coordinate in the resulting street complex is equal to zero. Because we need only trace one component of γ, the result now follows immediately from Theorem 6.6.4. □

Štefankovic described an algorithm to test whether a normal curve γ is connected in $O(W) = O(n \log X)$ time, where W is the bit-complexity of γ's normal coordinates [179, Observation 3.3.1]. Our backward analysis in Section 6.7.3 actually implies that our algorithm actually runs in $O(nW')$ time, where W' is the bit-complexity of the normal coordinates of *just the traced component* of γ.

Theorem 6.8.2. *Let Σ be a surface composed of n triangles; let γ be a normal curve in Σ with total crossing length X, represented by its normal coordinates; and let x be any intersection point of γ with an edge of Σ, represented by its index along that edge. We can compute the normal coordinates of the component of γ containing x in $O(n^2 \log X)$ time.*

Proof: Suppose x is the ith crossing point along some edge e; let $\gamma(e)$ denote the number of crossings between γ and e; and let γ_x denote the component of γ containing x. We trace γ_x starting at x, by splitting e into two smaller edges with street coordinates $i-1$ and $\gamma(e)-i$; these two new edges and e define a fork. If γ_x is a cycle, the tracing algorithm eventually reaches x again. Otherwise, when the tracing algorithm reaches an endpoint y of γ_x, we continue the trace from x to the other endpoint, as if starting a new component of γ. (Alternatively, we can simply start over and trace γ_x from y to the other endpoint.) In all cases, tracing γ_x requires $O(n^2 \log X)$ time. Finally, to recover the normal coordinates of γ_x, we reset all the street and junction coordinates in $S(\Sigma, \gamma_x)$ to zero and then untrace γ_x, using either Lemma 6.7.1 or Lemma 6.7.2. □

Štefankovic described an algorithm for this problem that runs in $O(nW) = O(n^2 \log X)$ time; see the proof of Lemma 3.3.3 in his thesis [179]. Like the previous theorem, more careful analysis implies that our algorithm runs in $O(nW')$ time, where W' is the bit-complexity of the normal coordinates of γ_x.

6.8.2 Forward and reverse indexing

Let x be a point of intersection between γ with an edge e of the surface triangulation. The *edge-index* of x is the position of x in the sequence of intersection points along e (directed arbitrarily). Similarly, if x lies on an arc component of γ, the *arc-index* of x is the position of x in the sequence of intersection points along that arc (again, directed arbitrarily). Schaefer *et al.* [163] describe an algorithm to compute the arc-index of an intersection point from its edge-index in time polynomial in $n \log X$. We can more efficiently transform either index into the other using our tracing and untracing algorithms.

Theorem 6.8.3. *Let Σ be a surface composed of n triangles; let γ be a normal* ***arc*** *in Σ with total crossing length X, represented by its normal coordinates; and let x be any intersection point of γ with an edge e of Σ, represented by its edge-index. We can compute the arc-index of x in $O(n^2 \log X)$ time.*

Proof: We trace γ *against* its chosen indexing direction, starting at x. As we trace γ, we maintain the crossing lengths of all streets in the evolving street complex. Also, whenever we traverse a street, we add its crossing length to a running counter. Then the trace reaches the boundary of Σ, the counter contains the arc-index of x. □

Theorem 6.8.4. *Let Σ be a surface composed of n triangles; let γ be a normal* ***arc*** *in Σ with total crossing length X, represented by its normal coordinates; and let x be any intersection point of γ with an edge of Σ, represented by its arc-index. We can compute the edge containing x and the index of x along that edge in $O(n^2 \log X)$ time.*

Proof: We trace γ along its chosen indexing direction, starting at one boundary point, maintaining the crossing lengths of all streets. Whenever the tracing algorithm

traverses a street, we add its crossing length to a running counter. When the counter reaches the curve-index of x, we stop the tracing algorithm and add a fork to the street complex at the point x. Note that x may lie in the interior of the last street traversed by the trace. We then untrace the traced subpath of γ, again starting at the boundary endpoint and untracing toward x. When the untracing algorithm reaches x, the desired edge-index is one of the street coordinates of the fork. □

6.8.3 Normal isotopy classes

Theorem 6.8.5. *Let Σ be a surface composed of n triangles, and let γ be a normal curve in Σ with total crossing length X, represented by its normal coordinates. We can compute the number of normal isotopy classes of components of γ and the number of components in each normal isotopy class in $O(n^2 \log X)$ time.*

Proof: We begin by counting and deleting the trivial components of γ. The number of trivial components that separate any vertex v from the other vertices is just the minimum of the corner coordinates incident to v. Thus, we can easily count all trivial components and delete them from γ, by reducing the appropriate normal coordinates, in $O(n)$ time. We separately record the number of trivial cycles and the number of trivial arcs with endpoints on each boundary cycle.

Next, we repeatedly trace one component of γ and then count and remove all other components in the same normal isotopy class, as follows. Suppose we have already traced components $\gamma_1, \ldots, \gamma_{i-1}$. Let $\hat{\gamma}_{<i}$ denote the reduced normal curve $\gamma_1 \cup \cdots \cup \gamma_{i-1}$, and let $\gamma_{\geq i}$ denote the union of all components of γ that are *not* normal-isotopic to any component of $\hat{\gamma}_{<i}$. In particular, we have $\hat{\gamma}_{<1} = \emptyset$ and $\gamma_{\geq 1} = \gamma$. By assumption, we have computed the street complex $S(T, \hat{\gamma}_{<i})$ as well as the street and junction coordinates of $\gamma_{\geq i}$. Let x be the leftmost intersection point between $\gamma_{\geq i}$ and some non-redundant port p in $S(T, \hat{\gamma}_{<i})$, and let γ_i denote the component of $\gamma_{\geq i}$ that contains x. We trace γ_i through $S(T, \hat{\gamma}_{<i})$ to produce the street complex $S(T, \hat{\gamma}_{<(i+1)})$, along with the street and junction coordinates of $\gamma_{\geq i} \setminus \gamma_i$. The number of other components of γ that are normal isotopic to γ_i is the minimum of the junction coordinates just to the right of γ_i in the new street complex $S(T, \hat{\gamma}_{<(i+1)})$. Thus, we can easily count these components and reduce the appropriate street and junction coordinates in $O(n)$ time, thereby computing the street and junction coordinates of $\gamma_{\geq(i+1)}$.

Theorem 6.6.7 implies that the total time spent tracing all components γ_i is $O(n^2 \log X)$. Lemma 6.4.1 implies that there are at most $O(n)$ normal-isotopy classes of components in γ, so the total time spent counting and removing parallel components of γ is only $O(n^2)$. □

The output of the above algorithm is the street complex $S(\Sigma, \hat{\gamma})$. Štefankovic described an algorithm to count normal isotopy classes in $O(n^3 \log^2 X)$ time [179, Lemma 3.3.3]; his algorithm actually computes the normal coordinates of one component in each class. We can compute the same output representation by

independently untracing each component of $\hat{\gamma}$, using either Lemma 6.7.1 or Lemma 6.7.2. Corollary 6.4.2 implies that the total time to untrace all components is $O(n^3 \log X)$, which is still slightly faster than Štefankovic's algorithm.

Corollary 6.8.6. *Let Σ be a surface composed of n triangles, and let γ be a normal curve in Σ with total crossing length X, represented by its normal coordinates. We can compute the normal coordinates of each normal-isotopy class of components of γ in $O(n^3 \log X)$ time.*

Theorem 6.8.5 also implies immediately that we can compute the number of components of a given normal curve in $O(n^2 \log X)$. Štefankovic described an algorithm that solves this problem in $O(n \log X)$ time [179, Observation 3.3.1].

Corollary 6.8.7. *Let Σ be a surface composed of n triangles, and let γ be a normal curve in Σ with total crossing length X, represented by its normal coordinates. We can compute the number of components of γ in $O(n^2 \log X)$ time.*

6.8.4 Isotopy classes

Theorem 6.8.8. *Let Σ be a surface composed of n triangles, and let γ be a normal curve in Σ with total crossing length X, represented by its normal coordinates. We can compute the number of isotopy classes of components of γ and the number of components in each isotopy class in $O(n^2 \log X)$ time.*

Proof: We begin by computing the number and multiplicities of the *normal* isotopy classes of components of γ in $O(n^2 \log X)$ time, as described in the proof of Theorem 6.8.5. Let $\hat{\gamma}$ be the reduced curve containing one component of γ in each non-trivial normal isotopy class, and let $\gamma_1, \gamma_2, \ldots$ denote the components of $\hat{\gamma}$. The rest of the algorithm requires only $O(n)$ time.

Next we compute the Euler characteristic of the components of $\Sigma \setminus \hat{\gamma}$; to avoid confusion, we will refer to the components of $\Sigma \setminus \hat{\gamma}$ as *pieces*. Because each curve γ_i is a simple arc or cycle in the 1-skeleton of the street complex $S(T, \hat{\gamma})$, we can compute the Euler characteristic of every piece in $O(n)$ time using a depth-first search in the dual graph of $S(T, \hat{\gamma})$ [76]. In particular, we can identify which pieces are disks ($\chi = 1$) and annuli ($\chi = 0$).

We can now cluster the components of $\hat{\gamma}$ into isotopy classes as follows. Call a cycle or arc γ_i *obviously contractible* if it is the only component of $\hat{\gamma}$ on the boundary of a disk piece. Call any two arcs γ_i and γ_j *obviously isotopic* if they are the only components of $\hat{\gamma}$ on the boundary of a disk piece. Finally, call any two cycles γ_i and γ_j *obviously isotopic* if they comprise the boundary of an annulus piece. Let G be the graph whose nodes are the components of $\hat{\gamma}$ and whose edges connect obviously isotopic components. This graph has $O(n)$ nodes and $O(n)$ edges, and we can easily construct in $O(n)$ time.

An arc or cycle in $\hat{\gamma}$ is contractible if and only if it lies in the same component of G as an obviously contractible arc or cycle, and two arcs or cycles in $\hat{\gamma}$ are isotopic

if and only if they lie in the same component of G. Thus, we can easily cluster the components of $\hat{\gamma}$ into isotopy classes in $O(n)$ time. We can also compute the number of components of γ in each isotopy class in $O(n)$ time by adding the sizes of the appropriate normal-isotopy classes. □

Schaefer *et al.* [163] describe an algorithm to compute isotopy classes of normal curves in time polynomial in $n \log X$.[3] Their algorithm actually computes the normal coordinates of one component in each isotopy class. We can compute these normal coordinates by untracing one component in each isotopy class; Lemma 6.4.1 implies that there are at most $O(g + b)$ classes to consider.

Corollary 6.8.9. *Let Σ be a surface composed of n triangles, and let γ be a normal curve in Σ with total crossing length X, represented by its normal coordinates. We can compute the* ***normal coordinates*** *of each isotopy class of components of γ in $O((g + b)n^2 \log X)$ time.*

6.8.5 Algebraic intersection numbers

Finally, suppose γ^+ and δ^+ are *directed* normal curves that intersect only transversely and only at a finite number of points. We call an intersection point in $\gamma \cap \delta$ a *positive* (resp. *negative*) crossing if γ^+ crosses δ^+ from left to right (resp. from right to left) at that point; see Figure 6.12. The ***algebraic intersection number*** $\hat{\imath}(\gamma^+, \delta^+)$ is the number of positive crossings minus the number of negative crossings. We easily observe that $\hat{\imath}(\gamma^+, \delta^+) = -\hat{\imath}(\delta^+, \gamma^+) = -\hat{\imath}(\gamma^-, \delta^+)$, where γ^- is the reversal of γ^+. Algebraic intersection numbers are invariant under isotopy. In fact, the algebraic intersection number is an invariant of the *integer homology* classes of the two curves; think about a curve as a flow (or circulation), see Chapter 4.

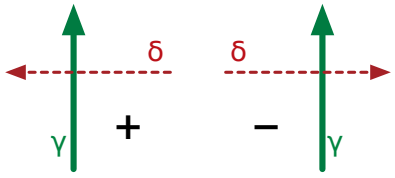


Figure 6.12. Positive and negative crossings.

The *signed edge coordinates* of a directed normal curve γ^+ are a list of the algebraic intersection numbers of γ^+ with each (arbitrarily oriented) edge in the triangulation. Similarly, the *signed corner coordinates* of γ^+ record, for each corner of the triangulation, the number of counterclockwise elementary segments in that corner, minus the number of clockwise segments. Reversing the direction of a normal curve negates all of its signed normal coordinates.

[3] In their second paper [166], Schaefer *et al.* claim to have an algorithm to list the isotopy classes of components of a given normal curve in $O(gn^2 \log^2 X)$ time (in our notation); however, no such result appears in any of their papers [163, 166, 179]. In particular, it is unclear how to determine whether two components of $\hat{\gamma}$ are isotopic using Štefankovic's techniques [179].

Signed normal coordinates do *not* determine a unique curve up to normal isotopy; nevertheless, given the signed normal coordinates of γ^+ and δ^+, we can compute $\hat{\imath}(\gamma^+, \delta^+)$ in $O(n)$ time by choosing an appropriate drawing of the two curves [163]. For each edge of the triangulation, we move all intersections with γ^+ close to one of the endpoints, chosen arbitrarily, and all intersections with δ^+ close to the other endpoint, and we then draw every elementary segment as a straight line segment, as shown in Figure 6.13. Then it is easy to compute the number of positive and negative crossings within each triangle in constant time, by multiplying at most six pairs of signed corner coordinates.

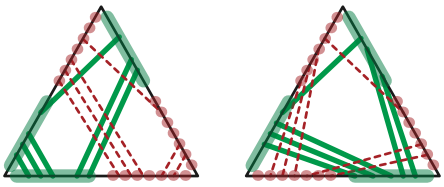


Figure 6.13. Intersection patterns of two normal curves within a single triangle.

Given the (unsigned) normal coordinates of an undirected normal arc or cycle γ, we can compute the signed normal coordinates of some orientation γ^+ of γ as follows. We begin by tracing γ in the chosen direction. We give each street in the resulting street complex $S(\Sigma, \gamma)$ an arbitrary reference direction. Then we untrace γ, maintaining *signed* street coordinates. Thus, in each untracing step, we either add or subtract the active street coordinates, depending on whether the directions of the active streets on either side of γ agree or disagree. The additional bookkeeping increases the running time of the untracing algorithm by only a small constant factor. When the untracing algorithm ends, we have the signed edge coordinates of γ^+; computing the signed corner coordinates in $O(n)$ additional time is straightforward.

Theorem 6.8.10. *Let Σ be a surface composed of n triangles, and let γ be a **connected** normal curve in Σ with total crossing length X, represented by its unsigned normal coordinates. We can compute the signed normal coordinates of some orientation of γ in $O(n^2 \log X)$ time.*

The algebraic intersection number of two *undirected* normal curves γ and δ is well-defined only if both curves are connected, and then only up to a sign change. Formally, we define $\hat{\imath}(\gamma, \delta) = |\hat{\imath}(\gamma^+, \delta^+)|$, where the directions of γ^+ and δ^+ are chosen arbitrarily.

Corollary 6.8.11. *Let Σ be a surface composed of n triangles, and let γ and δ be **connected** normal curves in Σ with total crossing length at most X, represented by their normal coordinates. We can compute the algebraic intersection number $\hat{\imath}(\gamma, \delta)$ in $O(n^2 \log X)$ time.*

Štefankovic described algorithms to compute signed normal coordinates and algebraic intersection numbers in $O(n \log X)$ time [179, Observation 3.6.1], which is a factor of $O(n)$ faster than our algorithms.

References

[1] R. Agarwala and D. Fernández-Baca. Weighted multidimensional search and its application to convex optimization. *SIAM J. Comput.* 25:83–99, 1996.

[2] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.* 30:412–458, 1998.

[3] P. K. Agarwal, M. Sharir, and S. Toledo. An efficient multi-dimensional searching technique and its applications. Tech. Rep. CS-1993-20, Dept. Comp. Sci., Duke Univ., August 1993. ⟨ftp://ftp.cs.duke.edu/pub/dist/techreport/1993/1993-20.ps.gz⟩.

[4] I. Agol, J. Hass, and W. P. Thurston. The computational complexity of knot genus and spanning area. *Trans. Amer. Math. Soc.* 358(9):3821–3850, 2006.

[5] R. K. Ahuja, T. L. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[6] H. Alt and M. Buchin. Semi-computability of the Fréchet distance between surfaces. *Proc. 21st Euro. Workshop on Comput. Geom.*, 45–48, 2005.

[7] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.* 5:75–91, 1995.

[8] C. Bennis, J.-M. Vézien, G. Iglésias, and A. Gagalowicz. Piecewise surface flattening for non-distorted texture mapping. *Proc. SIGGRAPH '91*, 237–246, 1991. vol. 25. ⟨citeseer.nj.nec.com/bennis91piecewise.html⟩.

[9] M. D. Berg, M. V. Kreveld, and S. Schirra. A new approach to subdivision simplification. *Proc. 12th International Symp Symp. Comp. Assist. Cartog.*, 79–88, 1995.

[10] S. Bespamyatnikh. Computing homotopic shortest paths in the plane. *J. Algorithms* 49(2):284–303, 2003.

[11] J. S. Birman and C. Series. Geodesics with bounded intersection number on surfaces are sparsely distributed. *Topology* 24(2):217–225, 1985.

[12] J. S. Birman and C. Series. Algebraic linearity for an automorphism of a surface group. *J. Pure Appl. Algebra* 52:227–275, 1988.

[13] R. V. Book. Dehn's algorithm and the complexity of word problems. *Amer. Math. Monthly* 95(10):919–925, 1988.

[14] G. Borradaile. *Exploiting Planarity for Network Flow and Connectivity Problems*. Ph.D. thesis, Brown University, May 2008. ⟨http://www.cs.brown.edu/research/pubs/theses/phd/2008/glencora.pdf⟩.

[15] G. Borradaile and P. Klein. An $O(n \log n)$-time algorithm for maximum st-flow in a directed planar graph. *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithms*, 524–533, 2006.

[16] G. Borradaile and P. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM* 56(2): 9:1–30, 2009.

[17] G. Borradaile, P. N. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, 170–179, 2011. FOCS '11, IEEE Computer Society. ⟨http://dx.doi.org/10.1109/FOCS.2011.73⟩.

[18] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. *Proc. 31st VLDB Conference*, 853–864, 2005. VLDB Endowment.

[19] G. R. Brightwell and P. Winkler. Submodular percolation. *SIAM J. Discret. Math.* 23(3):1149–1178. Society for Industrial and Applied Mathematics, 2009. ⟨http://dx.doi.org/10.1137/07069078X⟩.

[20] K. Buchin, M. Buchin, and J. Gudmundsson. Detecting single file movement. *Proc. 16th ACM SIGSPATIAL Int. Conf. Adv. GIS*, 288–297, 2008.

[21] K. Buchin, M. Buchin, J. Gudmundsson, M. L., and J. Luo. Detecting commuting patterns by clustering subtrajectories. *Proc. 19th Annu. Internat. Sympos. Algorithms Comput.*, 644–655, 2008.

[22] C. Buehler, S. J. Gortler, M. F. Cohen, and L. McMillan. Minimal surfaces for stereo. *Proc. 7th European Conf. Comput. Vision*, 885–899, 2002. vol. 3.

[23] B. A. Burton. The complexity of the normal surface solution space. *Proc. 26th Ann. Symp. Comput. Geom.*, 201–209, 2010.

[24] B. A. Burton and M. Ozlen. Computing the crosscap number of a knot using integer programming and normal surfaces. *ACM Trans. Math. Software* (to appear), 2012.

[25] B. Buttenfield. Treatment of the cartographic line. *Cartographica* 22:1–26, 1985.

[26] S. Cabello. Many distances in planar graphs. *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithms*, 1213–1220, 2006.

[27] S. Cabello and E. W. Chambers. Multiple source shortest paths in a genus g graph. *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, 89–97, 2007.

[28] S. Cabello, É. Colin de Verdière, and F. Lazarus. Finding shortest non-trivial cycles in directed graphs on surfaces. *Proc. 26th Ann. Symp. Comput. Geom.*, 156–165, 2010.

[29] S. Cabello, É. Colin de Verdière, and F. Lazarus. Finding cycles with topological properties in embedded graphs. *SIAM J. Discrete Math.* 25:1600–1614, 2011.

[30] S. Cabello, Y. Liu, A. Mantler, and J. Snoeyink. Testing homotopy for paths in the plane. *Discrete Comput. Geom.* 31:61–81, 2004.

[31] S. Cabello and B. Mohar. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. *Discrete Comput. Geom.* 37:213–235, 2007.

[32] G. Carlsson, T. Ishkhanov, V. de Silva, and A. Zomorodian. On the local bahavior of spaces of natural images. *Int. J. Comput. Vision* 76(1):1–12, 2008.

[33] E. Chambers, S. Cabello, and J. Erickson. Multiple-source shortest paths in embedded graphs. , 2012.

[34] E. W. Chambers, E. Colin de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Homotopic fréchet distance between curves or, walking your dog in the woods in polynomial time. *Comput. Geom. Theory Appl.* 43(3):295–311, 2010.

[35] E. W. Chambers, É. Colin de Verdière, J. Erickson, F. Lazarus, and K. Whittlesey. Splitting (complicated) surfaces is hard. *Proc. 22nd Ann. Symp. Comput. Geom.*, 421–429, 2006.

[36] E. W. Chambers, É. Colin de Verdière, J. Erickson, F. Lazarus, and K. Whittlesey. Splitting (complicated) surfaces is hard. *Comput. Geom. Theory Appl.* 41(1–2):94–110, 2008.

[37] E. W. Chambers, J. Erickson, and A. Nayyeri. *SIAM Journal of Computing*.

[38] E. W. Chambers, J. Erickson, and A. Nayyeri. Homology flows, cohomology cuts. *Proc. 42nd Ann. ACM Symp. Theory Comput.*, 273–282, 2009.

[39] E. W. Chambers, J. Erickson, and A. Nayyeri. Minimum cuts and shortest homologous cycles. *Proc. 25th Ann. Symp. Comput. Geom.*, 377–385, 2009.

[40] E. W. Chambers and D. Letscher. On the height of a homotopy. *Proc. 21st Canad. Conf. Comput. Geom.*, 2009.

[41] E. W. Chambers and D. Letscher. Erratum for on the height of a homotopy. ⟨http://mathcs.slu.edu/~chambers/papers/hherratum.pdf⟩. http://mathcs.slu.edu/~chambers/papers/hherratum.pdf, 2010.

[42] C. Chen and D. Freedman. Quantifying homology classes. *Proc. 25th Ann. Symp. Theoretical Aspects Comput. Sci.*, 169–180, 2008. Dagstuhl Seminar Proceedings 08001. ⟨http://drops.dagstuhl.de/opus/volltexte/2008/1343/⟩.

[43] C. Chen and D. Freedman. Hardness results for homology localization. *Proc. 21st Ann. ACM-SIAM Symp. Discrete Algorithms*, 1594–1604, 2010.

[44] E. Cohen. Efficient parallel shortest-paths in digraphs with a separator decomposition. *J. Algorithms* 21:331–357, 1996.

[45] É. Colin de Verdière. Topological algorithms for graphs on surfaces. Preprint, May 2012.

[46] É. Colin de Verdière and J. Erickson. Tightening non-simple paths and cycles on surfaces. *Proc. 17th Ann. ACM-SIAM Symp. Discrete Algorithms*, 192–201, 2006.

[47] É. Colin de Verdière and J. Erickson. Tightening non-simple paths and cycles on surfaces. *SIAM J. Compu.*, 3784–3813, 2010.

[48] É. Colin de Verdière and F. Lazarus. Optimal system of loops on an orientable surface. *Discrete Comput. Geom.* 33(3):507–534, 2005.

[49] É. Colin de Verdière and F. Lazarus. Optimal pants decompositions and shortest homotopic cycles on an orientable surface. *J. ACM* 54(4), 2007.

[50] A. Collins, A. Zomorodian, G. Carlsson, and L. J. Guibas. A barcode shape descriptor for curve point cloud data. *Comput. & Graphics* 28(6):881–894, 2004.

[51] A. F. Cook, A. Driemel, S. Har-Peled, J. Sherette, and C. Wenk. Computing the Fréchet distance between folded polygons. *Proc. 12th Workshop Algorithms Data Struct.*, 267–278, 2011.

[52] A. F. Cook and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Trans. Algo.* 7:9:1–9:19. ACM, 2010.

[53] M. Dehn. Über unendliche diskontinuierliche Gruppen. *Math. Ann.* 71(1):16–144, 1911.

[54] M. Dehn. Transformation der Kurven auf zweiseitigen Flächen. *Math. Ann.* 72(3):413–421, 1912.

[55] M. Dehn. Die Gruppe der Abbildungsklassen (Das arithmetische Feld auf Flächen). *Acta Mathematica* 69:135–206, 1938.

[56] P. Dehornoy, I. Dynnikov, D. Rolfsen, and B. Wiest. *Ordering Braids*. Mathematical Surveys and Monographs 148. American Mathematical Society, 2008.

[57] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge Univ. Press, 2007.

[58] T. K. Dey. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge Univ. Press, 2007.

[59] T. K. Dey and S. Guha. Transforming curves on surfaces. *J. Comput. System Sci.* 58:297–325, 1999.

[60] T. K. Dey, A. N. Hirani, and B. Krishnamoorthy. Optimal homologous cycles, total unimodularity, and linear programming. *Proc. 42nd Ann. ACM Symp. Theory Comput.*, 221–230, 2010.

[61] T. K. Dey, A. N. Hirani, and B. Krishnamoorthy. Optimal homologous cycles, total unimodularity, and linear programming. *SIAM J. Comput.* 40(4):1026–1044, 2011.

[62] T. K. Dey and H. Schipper. A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *Discrete Comput. Geom.* 14(1):93–110, 1995.

[63] T. K. Dey, J. Sun, and Y. Wang. Approximating loops in a shortest homology basis from point data. *Proc. 26th Ann. Symp. Comput. Geom.*, 166–175, 2010.

[64] S. I. Diatch and D. A. Spielman. Faster lossy generalized flow via interior point algorithms. *Proc. 40th Ann. ACM Symp. Theory Comput.*, 451–460, 2008.

[65] V. Diekert and M. Kufleitner. A remark about quadratic trace equations. *Proc. 6th Int. Conf. Devel. Language Theory*, 59–66, 2003. Lecture Notes Comput. Sci. 2450, Springer-Verlag.

[66] N. M. Dunfield and A. N. Hirani. The least spanning area of a knot and the optimal bounding chain problem. *Proc. 27th Ann. Symp. Comput. Geom.*, 135–144, 2011.

[67] I. Dynnikov and B. Wiest. On the complexity of braids. *J. Europ. Math. Soc.* 9(4):801–840, 2007.

[68] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[69] A. Efrat, L. J. Guibas, S. Har-Peled, J. S. Mitchell, and T. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete Comput. Geom.* 28:535–569, 2002.

[70] A. Efrat, S. G. Kobourov, and A. Lubiw. Computing homotopic shortest paths efficiently. *Comput. Geom. Theory Appl.* 35(3):162–172, 2006.

[71] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Tech. Report CD-TR 94/64, Christian Doppler Lab. Expert Sys., TU Vienna, 1994.

[72] D. Eppstein. Dynamic generators of topologically embedded graphs. *Proc. 14th Ann. ACM-SIAM Symp. Discrete Algorithms*, 599–608, 2003.

[73] J. Erickson. Maximum flows and parametric shortest paths in planar graphs. *Proc. 21st Ann. ACM-SIAM Symp. Discrete Algorithms*, 794–804, 2010.

[74] J. Erickson. Shortest non-trivial cycles in directed surface graphs. *Proc. 27th Ann. Symp. Comput. Geom.*, 236–243, 2011.

[75] J. Erickson. Combinatorial optimization of cycles and bases. *Proceedings of Simposia in Applied Mathematics*, 2012. 70.

[76] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. *Discrete Comput. Geom.* 31(1):37–59, 2004.

[77] J. Erickson and A. Nayyeri. Shortest homologous cycles and minimum cuts via homology covers. *Proc. 22nd Ann. ACM-SIAM Symp. Discrete Algorithms*, 1166–1176, 2011.

[78] J. Erickson and A. Nayyeri. Tracing compressed curves in triangulated surfaces. , 2012.

[79] J. Erickson and K. Whittlesey. Greedy optimal homotopy and homology generators. *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms*, 1038–1046, 2005.

[80] J. Erickson and K. Whittlesey. Transforming curves on surfaces redux. , 2012.

[81] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.* 72(5):868–889, 2006.

[82] A. Fathi, F. Laudenbach, and V. Poénaru. *Travaux de Thurston sur les surfaces*. Astérisque 66-67. Soc. Math. de France, 1979. Séminaire Orsay. English translation in [83].

[83] A. Fathi, F. Laudenbach, and V. Poénaru. *Thurston's Work on Surfaces*. Mathematical Notes. Princeton Univ. Press, 2011. Translated by Djun Kim and Dan Margalit. English translation of [82].

[84] M. S. Floater. Parameterization and smooth approximation of surface triangulations. *Comput. Aided Geom. Design* 14(4):231–250, 1997.

[85] K. Fox. Shortest non-trivial cycles in directed and undirected surface graphs. , 2012.

[86] M. Frechét. Sur quelques points du calcul fonctionnel. *Rendic. Circ. Mat. Palermo* 22:1–74, 1906.

[87] M. Frechét. Sur la distance de deux surfaces. *Ann. Soc. Polonaise Math.* 3:4–19, 1924.

[88] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs with applications. *SIAM J. Comput.* 16(6):1004–1004, 1987.

[89] M. L. Furst, J. L. Gross, and L. A. McGeoch. Finding a maximum-genus graph imbedding. *J. Assoc. Comput. Mach.* 35(3):523–534, 1988.

[90] L. Gąsieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. *Proc. 8th Scand. Workshop Algorithm Theory*, 392–403, 1996. Lecture Notes Comput. Sci. 1097, Springer.

[91] S. M. Gersten and H. B. Short. Small cancellation theory and automatic groups. *Invent. Math.* 102:305–334, 1990.

[92] R. Ghrist. Configuration spaces, braids, and robotics. Unpublished manuscript, July 2007. ⟨http://www.math.uiuc.edu/∼ghrist/preprints/singaporetutorial.pdf⟩.

[93] M. Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions*. Ph.D. thesis, Free University of Berlin, 1999.

[94] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM* 45(5):783–797, 1998.

[95] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. Assoc. Comput. Mach.* 35(4):921–940, 1988.

[96] H. Gong, J. Sim, M. Likhachev, and J. Shi. Multi-hypothesis motion planning for visual object tracking. *Computer Vision, IEEE International Conference on* 0:619–626. IEEE Computer Society, 2011.

[97] S. J. Gortler and D. Kirsanov. A discrete global minimization algorithm for continuous variational problems. Comput. Sci. Tech. Rep. TR-14-04, Harvard Univ., 2004. ⟨http://gvi.seas.harvard.edu/sites/all/files/Gortler_DiscreteGlobal.pdf⟩.

[98] L. Grady. Computing exact discrete minimal surfaces: Extending and solving the shortest path problem in 3D with applicaton to segmentation. *Proc. IEEE CS Conf. Comput. Vis. Pattern Recog.*, 67–78, 2006. vol. 1.

[99] L. Grady. Minimal surfaces extend shortest path segmentation methods to 3D. *IEEE Trans. Pattern Anal. Mach. Intell.* 32(2):321–334, 2010.

[100] J. L. Gross and T. W. Tucker. *Topological graph theory*. Wiley-Interscience, 1987. Reprinted by Dover Publ., 2001.

[101] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1(2):169–197, 1981.

[102] W. Haken. Theorie der Normalflächen: Ein Isotopiekriterium für den Kreisknoten. *Acta Mathematica* 105:245–375, 1961.

[103] S. Har-Peled, A. Nayyeri, M. Salavatipour, and A. Sidiropoulos. How to walk your dog in the mountains with no magic leash. *Proceedings of the 28th symposuim on Computational Geometry*, 121–130, 2012. SoCG '12, ACM. ⟨http://doi.acm.org/10.1145/2261250.2261269⟩.

[104] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, 448–457, 2011. http://www.cs.uiuc.edu/~sariel/papers/10/frechet3d/.

[105] R. Hassin. Maximum flow in (s,t) planar networks. *Inform. Proc. Lett.* 13:107, 1981.

[106] R. Hassin and D. B. Johnson. An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM J. Comput.* 14(3):612–624, 1985.

[107] J. Hass, J. C. Lagarias, and N. Pippenger. The computational complexity of knot and link problems. *J. ACM* 46(2):185–211, 1999.

[108] A. Hatcher. *Algebraic Topology*. Cambridge Univ. Press, 2002. ⟨http://www.math.cornell.edu/~hatcher/AT/ATpage.html⟩.

[109] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.* 55(1):3–23, 1997.

[110] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Comput. Geom. Theory Appl.* 4:63–98, 1994.

[111] J. M. Hochstein and K. Weihe. Maximum s-t-flow with k crossings in $O(k^3 n \log n)$ time. *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, 843–847, 2007.

[112] H. Imai and K. Iwano. Efficient sequential and parallel algorithms for planar minimum cost flow. *Proc. SIGAL Int. Symp. Algorithms*, 21–30, 1990. Lecture Notes Comput. Sci. 450, Springer-Verlag.

[113] A. Itai and Y. Shiloach. Maximum flow in planar networks. *SIAM J. Comput.* 8:135–150, 1979.

[114] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. *Proc. 43rd Ann. ACM Symp. Theory Comput.*, 313–322, 2011.

[115] W. Jaco, D. Letscher, and J. H. Rubinstein. Algorithms for essential surfaces in 3-manifolds. *Topology and Geometry: Commemorating SISTAG*, 107–124, 2002. Contemporary Mathematics 314, American Mathematical Society.

[116] W. Jaco and J. H. Rubinstein. 0-efficient triangulations of 3-manifolds. *J. Diff. Geom.* 65:61–168, 2003.

[117] L. Janiga and V. Koubek. Minimum cut in directed planar networks. *Kybernetika* 28(1):37–49, 1992.

[118] D. B. Johnson and S. M. Venkatesan. Partition of planar flow networks (preliminary version). *Proc. 24th IEEE Symp. Found. Comput. Sci.*, 259–264, 1983.

[119] M. Karpinski, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nordic J. Comput.* 4:172–186, 1997.

[120] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping to massive dataset. *Proc. of the Third Euro. Conf. Princip. Data Mining and Know. Disc.*, 1–11, 1999.

[121] S. Khuller, J. Naor, and P. Klein. The lattice structure of flow in planar graphs. *SIAM J. Discrete Math.* 477–490, 1993.

[122] J. C. Kieffer and E. Yang. Grammar based codes: A new class of universal lossless source codes. *IEEE Trans. Inform. Theory* 46(3):737–754, 2000.

[123] M. Kim, S. Kim, and M. Shin. Optimization of subsequence matching under time warping in time-series databases. *Proc. ACM symp. Applied comput.*, 581–586, 2005.

[124] D. Kirsanov. *Minimal discrete curves and surfaces*. Ph.D. thesis, Div. Engin. Appl. Sci., Harvard Univ., September 2004. ⟨http://www.eecs.harvard.edu/~sjg/papers/danilthesis.pdf⟩.

[125] P. Klein. Multiple-source shortest paths in planar graphs. *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms*, 146–155, 2005.

[126] P. Klein, S. Mozes, and O. Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *ACM Trans. Algorithms* 6(2):article 30, 2010.

[127] H. Kneser. Geschlossene Flächen in dreidimensionalen Mannigfaltigkeiten. *Jahresbericht Deutschen Math.-Verein.* 38:248–260, 1930.

[128] M. Kutz. Computing shortest non-trivial cycles on orientable surfaces of bounded genus in almost linear time. *Proc. 22nd Ann. Symp. Comput. Geom.*, 430–438, 2006.

[129] J. Lacki, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Single source - all sinks max flows in planar digraphs.

[130] G. Lamé. Note sur la limite du monbre des divisions dans la recherche du plus grand commun diviseur entre deux nombres entiers. *Compt. Rend. Acad. Sci., Paris* 19:857–870, 1844.

[131] F. Lazarus and J. Rivaud. On the homotopy test on surfaces. Preprint, October 21, 2011.

[132] Y. Lifshits. *Algorithms and Complexity Analysis for Processing Compressed Texts*. Ph.D. thesis, Steklov Inst. Math., May 2007. In Russian.

[133] Y. Lifshits. Processing compressed texts: A tractabiliity border. *Proc. 18th Ann. Symp. Combin. Pattern Matching*, 228–240, 2007. Lecture Notes Comput. Sci. 4850, Springer-Verlag.

[134] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.* 16:346–358, 1979.

[135] L. A. Lyusternik. *Shortest Paths: Variational Problems*. Popular Lectures Math. 13. Pergamon Press, 1964. Translated and adapted from the Russian by P. Collins and Robert B. Brown.

[136] F. M. Maley. *Single-Layer Wire Routing and Compaction*. MIT Press, Cambridge, MA, 1990.

[137] A. Mascret, T. Devogele, I. L. Berre, and A. Hénaff. Coastline matching process based on the discrete Fréchet distance. *Proc. 12th Int. Sym. Spatial Data Handling*, 383–400, 2006.

[138] S. T. McCormick, M. R. Rao, and G. Rinaldi. Easy and difficult objective functions for max cut. *Math. Program., Ser. B* 94:459–466, 2003.

[139] G. L. Miller and J. Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.* 24(5):1002–1017, 1995.

[140] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.* 16:647–668, 1987.

[141] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. *J. Discrete Algorithms [Hermes]* 1(1):187–204, 2000.

[142] R. Moeckel. Geodesics on modular surfaces and continued fractions. *Ergodic Theory Dynam. Sys.* 2:69–83, 1982.

[143] B. Mohar and C. Thomassen. *Graphs on Surfaces*. Johns Hopkins Univ. Press, 2001.

[144] S. Mozes and C. Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log\log n)$ time. *Proc. 18th Ann. Europ. Symp. Algorithms*, 206–217, 2010. Lecture Notes Comput. Sci. 6347, Springer-Verlag.

[145] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.* 41(2):338–350, 1993.

[146] J. Pach and G. Tóth. Recognizing string graphs is decidable. *Discrete Comput. Geom.* 28(4):593–606, 2001.

[147] V. Y. Pan and J. H. Reif. Fast and efficient parallel solution of sparse linear systems. *SIAM J. Comput.* 22(6):1227–1250, 1993.

[148] M. Parizeau and R. Plamondon. A comparative analysis of regional correlation, dynamic time warping, and skeletal tree matching for signature verification. *IEEE Trans. Pattern Anal. Mach. Intell.* 12(7):710–717. IEEE Computer Society, 1990. ⟨http://dx.doi.org/10.1109/34.56215⟩.

[149] R. C. Penner. The action of the mapping class group on curves in surfaces. *L'Enseignment Mathématique* 30:39–55, 1984.

[150] R. C. Penner and J. L. Harer. *Combinatorics of Train Tracks*. Annals of Math. Studies 125. Princeton Univ. Press, 1992.

[151] D. Piponi and G. Borshukov. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. *Proc. SIGGRAPH 2000*, 471–478, 2000.

[152] W. Plandowski and W. Rytter. Application of Lempel-Ziv encodings to the solution of word equations. *Proc. 25th Int. Conf. Automata Lang. Prog.*, 731–742, 1998. Lecture Notes Comput. Sci. 1443, Springer-Verlag.

[153] H. Poincaré. Analysis Situs. *J. École Polytechnique* 1:1–121, 1895. Reprinted in *Ouvres* VI:193–288. English translation in [155].

[154] H. Poincaré. Cinquième complement à l'analysis situs. *Rendiconti del Circulo Matematico di Palermo* 18:45–110, 1904. English translation in [155].

[155] H. Poincaré. *Papers on Topology:* Analysis Situs *and Its Five Supplements*. History of Mathematics 37. American Mathematical Society, 2010. Translated from the French and with an introduction by John Stillwell.

[156] J. Reif. Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM J. Comput.* 12:71–81, 1983.

[157] G. Ringel. *Map Color Theorem*. Springer-Verlag, 1974.

[158] G. Ringel and J. W. T. Youngs. Solution of the Heawood map-coloring problem. *Proc. Nat. Acad. Sci. USA* 60:438–445, 1968.

[159] J. M. Robson and V. Diekert. On quadratic word equations. *Proc. 16th Ann. Conf. Theoretical Aspects Comput. Sci.*, 217–226, 1999. Lecture Notes Comput. Sci. 1563, Springer-Verlag.

[160] J. M. Robson and V. Diekert. Quadratic word equations. *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, 314–326, 1999. Springer-Verlag.

[161] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoret. Comput. Sci.* 302:211–222, 2003.

[162] A. D. abd S. Har-Peled and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete Comput. Geom.* 48:94–127, 2012.

[163] M. Schaefer, E. Sedgwick, and D. Štefankovič. Algorithms for normal curves and surfaces. *Proc. 8th Int. Conf. Comput. Combin.*, 370–380, 2002. Lecture Notes Comput. Sci. 2387, Springer-Verlag.

[164] M. Schaefer, E. Sedgwick, and D. Štefankovič. Recognizing string graphs in NP. *J. Comput. System Sci.* 67(2):365–380, 2003.

[165] M. Schaefer, E. Sedgwick, and D. Štefankovič. Spiraling and folding: The topological view. *Proc. 19th Ann. Canadian Conf. Comput. Geom.*, 73–76, 2007.

[166] M. Schaefer, E. Sedgwick, and D. Štefankovič. Computing Dehn twists and geometric intersection numbers in polynomial time. *Proc. 20th Canadian Conf. Comput. Geom.*, 111–114, 2008. Full version: Tech. Rep. 05–009, Comput. Sci. Dept., DePaul Univ., April 2005, ⟨http://facweb.cs.depaul.edu/research/techreports/abstract05009.htm⟩.

[167] M. Schaefer, E. Sedgwick, and D. Štefankovič. Spiraling and folding: The word view. *Algorithmica* 60(3):609–626, 2011.

[168] H. Schipper. Determining contractibility of curves. *Proceedings of the eighth annual symposium on Computational geometry*, 358–367, 1992. SCG '92, ACM. ⟨http://doi.acm.org/10.1145/142675.142749⟩.

[169] S. Schleimer. Sphere recognition lies in NP. Preprint, July 2004.

[170] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics 24. Springer-Verlag, 2003.

[171] C. Series. The modular surface and continued fractions. *J. London Math. Soc.* 31:69–80, 1985.

[172] C. Series. Geometrical Markov coding of geodesics on surfaces of constant negative curvature. *Ergodic Theory Dynam. Sys.* 6(4):601–625, 1986.

[173] J. Serrà, E. Gómez, P. Herrera, and X. Serra. Chroma binary similarity and local alignment applied to cover song identification. *IEEE Transactions on Audio, Speech & Language Processing* 16(6):1138–1151, 2008.

[174] J. Shallit. Origins of the analysis of the Euclidean algorithm. *Hist. Math.* 21:401–419, 1994.

[175] A. Sheffer and E. de Sturler. Surface parameterization for meshing by triangulation flattening. *Proc. 9th International Meshing Roundtable*, 161–172, 2000.

[176] V. de Silva, R. Ghrist, and A. Muhammad. Blind swarms for coverage in 2-d. *Robotics: Science and Systems'05*, 335–342, 2005.

[177] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.* 26(3):362–391, 1983.

[178] E. Sriraghavendra, K. K., and C. Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 01*, 461–465, 2007. ICDAR '07, IEEE Computer Society. ⟨http://dl.acm.org/citation.cfm?id=1304595.1304769⟩.

[179] D. Štefankovič. *Algorithms for simple curves on surfaces, string graphs, and crossing numbers*. Ph.D. thesis, Dept. Comput. Sci., Univ. Chicago, June 2005.

[180] J. Stillwell. *Classical Topology and Combinatorial Group Theory*, 2nd edition. Graduate Texts in Mathematics 72. Springer-Verlag, 1993.

[181] J. M. Sullivan. *A Crystalline Approximation Theorem for Hypersurfaces*. Ph.D. thesis, Princeton Univ., October 1990. ⟨http://torus.math.uiuc.edu/jms/Papers/thesis/thesis.pdf⟩.

[182] S. Tazari and M. Müller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to Steiner tree approximation. *Discrete Appl. Math.* 157:673–684, 2009.

[183] C. Thomassen. Embeddings of graphs with no short noncontractible cycles. *J. Comb. Theory Ser. B* 48(2):155–177, 1990.

[184] W. P. Thurston. On the geometry and dynamics of diffeomorphisms of surfaces. *Bull. Amer. Math. Soc.* 19(2):417–431, 1988. Circulated as a preprint in 1976.

[185] P. M. Vaidya. Speeding-up linear programming using fast matrix multiplication. *Proc. 30th IEEE Symp. Found. Comput. Sci.*, 332–337, 1989.

[186] S. M. Venkatesan. *Algorithms for network flows*. Ph.D. thesis, The Pennsylvania State University, 1983. Cited in [118].

[187] Y. Wang. Measuring similarity between curves on 2-manifolds via minimum deformation area. , 2008.

[188] K. Weihe. Edge-disjoint (s,t)-paths in undirected planar graphs in linear time. *J. Algorithms* 23(1):121–138, 1997.

[189] K. Weihe. Maximum (s,t)-flows in planar networks in $O(|V|\log|V|)$-time. *J. Comput. Syst. Sci.* 55(3):454–476, 1997.

[190] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. *Proc. 18th Int. Conf. Sci. Statis. Database Manag.*, 879–888, 2006.

[191] A. Zomorodian. *Topology for Computing*. Cambridge Univ. Press, 2005.