FAST ALGORITHMS FOR SURFACE EMBEDDED GRAPHS VIA HOMOLOGY

BY

KYLE J. FOX

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, 2013

Urbana, Illinois

Doctoral Committee:

Professor Jeff Erickson, Chair
Professor Chandra Chekuri
Assistant Professor Derek Hoiem
Professor David Eppstein, University of California, Irvine
Abstract

We describe several results on combinatorial optimization problems for graphs where the input comes with an embedding on an orientable surface of small genus. While the specific techniques used differ between problems, all the algorithms we describe share one common feature in that they rely on the algebraic topology construct of homology. We describe algorithms to compute global minimum cuts and count minimum $s,t$-cuts. We describe new algorithms to compute short cycles that are topologically non-trivial. Finally, we describe ongoing work in designing a new algorithm for computing maximum $s,t$-flows in surface embedded graphs.

We begin by describing an algorithm to compute global minimum cuts in edge weighted genus $g$ graphs in $g^{O(g)} n \log \log n$ time. When the genus is a constant, our algorithm’s running time matches the best time bound known for planar graphs due to Łącki and Sankowski. In our algorithm, we reduce to the problem of finding a minimum weight separating subgraph in the dual graph and provide two subroutines tailored to different kinds of separating subgraphs.

We describe algorithms to compute short non-trivial cycles in edge weighted graphs. Some of the algorithms are tailored to take advantage of undirected edge weights, but others are designed to work in graphs where the edges are directed. For undirected graphs embedded on surfaces of genus $g$, our algorithms compute non-separating, non-contractible, and non-null-homologous cycles in $2^{O(g)} n \log \log n$ time, improving the previous best algorithms of Italiano et al. which run in $g^{O(g)} n \log \log n$ time. For directed graphs, we give an algorithm to compute shortest non-null-homologous cycles in $O(g^2 n \log n)$ time, matching the running time of Erickson’s algorithm for computing shortest non-separating cycles. We also give an $O(g^3 n \log n)$ time algorithm for computing shortest non-contractible cycles in directed graphs. This last result improves upon the previous best algorithm by Erickson which runs
in $g^{O(g)} n \log n$ time.

We describe an algorithm to count minimum $s,t$-cuts. Despite the problem being #P-complete in general graphs, our algorithm runs in $2^{O(g)} n^2$ time. After running our algorithm once it becomes possible to sample minimum cuts uniformly at random in $O(n \log n)$ time per sample. This result directly generalizes an $O(n^2)$ time algorithm for counting minimum $s,t$-cuts in planar graphs by Bezáková and Friedlander. Like Bezáková and Friedlander, we reduce the problem of counting minimum $s,t$-cuts to one of counting *forward* $t,s$-cuts in an embedded directed acyclic graph.

Finally, we describe ongoing work toward computing maximum $s,t$-flows. Borradaile and Klein describe an $O(n \log n)$ time algorithm to compute maximum $s,t$-flows in planar graphs. We give a new algorithm that appears to generalize their techniques naturally to surfaces with positive genus, and similar to their algorithm we are able to send flow down augmenting paths in $O(g \log n + g^2)$ amortized time per path. We prove that our algorithm performs a quadratic number of augmentations, giving it an overall time bound of $O(g n^2 (\log n + g))$. While we have been unable to find a proof so far, we believe our algorithm may actually run in near-linear time when the surface has constant genus.
For my parents, who taught me where to aim, and for my wife, who pushed me to hit my target.
Acknowledgments

First off, I want to thank my advisor Jeff Erickson. His instruction was the reason I became interested in algorithms in the first place, and I strive to follow his example in my own teaching and research. His encouragement and patience got me to my first results, and he has given me the confidence needed to find the rest.

The work in this thesis is due to collaborations with Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. During my graduate research I have also collaborated with Antonios Antoniadis, Neal Barcelo, Daniel Cole, Sungjin Im, William B. Kinnersley, Madhukar Korupolu, Janardhan Kulkarni, Daniel McDonald, Benjamin Moseley, Michael Nugent, Nathan Orlow, Kirk Pruhs, and Gregory J. Puleo. It has been an honor to work with so many outstanding individuals, and I can only hope that I find so many fruitful collaborations in the future.

My research was supported in part by the Department of Energy Office of Science Graduate Fellowship Program (DOE SCGF), made possible in part by the American Recovery and Reinvestment Act of 2009, administered by ORISE-ORAU under contract no. DE-AC05-06OR23100. Thank you too all the staff involved in the DOE SCGF, especially Ping Ge and Cayla Stephenson for always being available to answer questions about the fellowship resources.

Thank you to the current and former student members of UIUC’s theory group, in particular Hsien-Chih Chang, Alina Ene, Nitish Korula, Nirman Kumar, Hemanta Maji, David Morrison, Benjamin Raichel, Mike Rosulek, Md. Abul Hassan Samee, Jason Sauppe, and Daniel Schreiber. My discussions with all of you were invaluable for helping me figure out what I am doing, or simply for distracting me when I needed a break. Thank you to the core theory faculty here who were always willing to offer advice, Chandra Chekuri, Sariel Har-Peled, Alexandra Kolla, and Manoj Prabhakaran.
Thank you to my other dissertation committee members David Eppstein and Derek Hoiem. Finally, I would like to thank some others who offered advice, gave me assistance, or joined me in interesting discussions on algorithms research: Glencora Borradaile, Philip N. Klein, Justin Kopinsky, Shay Mozes, Anastasios Sidiropoulos, and Douglas B. West.

Thank you to all the staff of UIUC’s Computer Science Department, especially Rhonda McElroy and Elaine Wilson. You keep everything from breaking down, and I will never forget your friendship. In addition, I wish to thank all my other friends who gave me encouragement and advice throughout my graduate experience, or who just kept me distracted enough not to become overwhelmed. In particular, I want to thank Austin Mohr, Chris Rishel, Paul Sorensen, Anthony Schuster, Terence Yen, and the many wonderful dancers of the Illini Swing Society.

Finally, I wish to thank my family who made me the person I am today and support me as I move on to the next stages of life. Thanks to my wonderful wife Carol for improving my life in so many ways and for her patience whenever I come home in a math induced daze. I hope your parents weren’t too disappointed that I didn’t already have a Ph.D. when you married me. Thank you to my brother Ryan who is largely responsible for my original interest in mathematics and computer science and who has always been a great friend. Finally, thank you to my parents for showing me love and support throughout my whole life. Without your encouragement, I would never have gotten this far.
Table of Contents

Chapter 1  Introduction ...................................................... 1
  1.1  Homology, Homotopy, and Related Results .................. 3
  1.2  Organization and Results ......................................... 5

Chapter 2  Preliminary Material .......................................... 7
  2.1  Surfaces and Curves ................................................. 7
  2.2  Graph Embeddings .................................................. 8
  2.3  Homotopy and Homology ........................................... 10
  2.4  Flows and Cuts .................................................... 13
  2.5  Covering Spaces and Cutting .................................... 14

Chapter 3  Global Minimum Cuts ........................................ 16
  3.1  Homology Cuts ..................................................... 18
  3.2  Contractible Cycle ................................................ 19
  3.3  Non-contractible Components .................................. 22
  3.4  Conclusions and Open Problems ............................... 25

Chapter 4  Shortest Non-trivial Cycles ................................ 27
  4.1  New Results ....................................................... 29
  4.2  Shortest Non-trivial Cycles in Undirected Graphs .......... 31
  4.3  Shortest Non-null-homologous Cycles in Directed Graphs .. 35
  4.4  The Infinite Cyclic Cover ....................................... 38
  4.5  Lifting Shortest Non-contractible Cycles .................... 41
  4.6  Shortest Non-contractible Cycles in Directed Graphs ...... 44
  4.7  Conclusions and Open Problems ................................ 46

Chapter 5  Counting Minimum s,t-cuts ................................. 48
  5.1  New Results ....................................................... 49
  5.2  Directed Graphs and Definitions .............................. 51
  5.3  Minimum Cuts and Forward Cuts ............................... 52
  5.4  Forward Cuts and Cocirculations .............................. 53
  5.5  Counting Cuts in Triangulations .............................. 56
  5.6  Handling Non-triangulations ................................... 62
  5.7  Sampling Minimum Cuts ......................................... 69
<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>Maximum Flows</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>New Results</td>
<td>75</td>
</tr>
<tr>
<td>6.2</td>
<td>Assumptions and Data Structures</td>
<td>76</td>
</tr>
<tr>
<td>6.3</td>
<td>Homological Maxflow/Mincut</td>
<td>80</td>
</tr>
<tr>
<td>6.4</td>
<td>Parametric Minimum Capacity Coflows</td>
<td>83</td>
</tr>
<tr>
<td>6.5</td>
<td>Shortest Path Coflows</td>
<td>85</td>
</tr>
<tr>
<td>6.6</td>
<td>Augmentation Steps and Pivots</td>
<td>87</td>
</tr>
<tr>
<td>6.7</td>
<td>Time Bounds</td>
<td>90</td>
</tr>
<tr>
<td>6.8</td>
<td>Conclusions and Future Directions</td>
<td>91</td>
</tr>
</tbody>
</table>

References | 93 |
Chapter 1

Introduction

The search for efficient graph algorithms has played a key role in the study of theoretical computer science for most of the last century. Graphs provide a natural abstraction for many real world constructs such as road or computer networks, and describing these concepts in the language of graphs often invites simple or rigorous solutions to real world problems. Looking at the various algorithms for minimum spanning trees, shortest paths, maximum flows, and more, time has proven the algorithms research community’s ability to continue presenting higher utility, more general, and more costly algorithms for graphs.

Of course, it is unlikely that there exist efficient algorithms for the various NP-hard graph problems such as Maximum Cut [86]. However, even some of the best known successes in efficient graph algorithm research such as the Bellman-Ford algorithm for shortest paths or the Ford-Fulkerson paradigm for computing maximum flows [57] still come with a high computational cost. These results present algorithms with at least a quadratic running time, which is still prohibitively high for very large graphs.

Fortunately, these running times can be improved by making certain assumptions on the input graph being used. In particular, there is a large body of research on finding efficient algorithms for graphs embeddable in the plane. The planarity of a graph is a natural assumption to make. Many graphs representing real world objects such as road networks or VLSI circuits are inherently planar or exhibit very few crossings when drawn in the plane. Further, planarity aids in the computation of many constructs also studied in more general graphs. These constructs include minimum spanning trees [100, 109]; single-source shortest paths [73, 94, 98, 103, 123]; multiple-source shortest paths [13, 93]; replacement paths [50, 129]; and maximum flows and minimum cuts [9, 19, 59, 71, 76, 77, 97, 112, 126]. In fact, planarity even helps in finding approximations [6–8, 36, 44] and exact solu-
tions [4, 43, 44, 65, 74, 101, 107, 111] for many problems with no known polynomial time solution for general graphs. We will discuss some of these results later in the thesis.

The successes in finding algorithms that require assumptions on the input graph do not stop with planarity. Many of the above algorithms can be extended for generalizations of planar graphs such as graphs embeddable on a surface of small genus or graphs forbidding a fixed minor. Indeed, these more general graphs share many convenient properties with planar graphs such as sparsity and the existence of small, well-balanced separators [2, 61, 99].

Not all of these extensions come easily, however. In particular, progress on computing maximum flows and minimum cuts came only very recently for graphs of bounded genus [22, 23, 48, 51, 77] and graphs forbidding certain fixed minors [21]. Similarly, Galluccio, Loebl, and Vondrák [60] recently described an algorithm for computing maximum weight cuts in surfaces of fixed genus where edges are weighted with small integers, and Patel [108] recently generalized a planar graph algorithm to compute minimum quotient cuts to surfaces of fixed genus.

Obtaining fast algorithms for graphs forbidding fixed minors is arguably the most natural long term goal for research of this type. Generalizing planar algorithms such as those for flows and cuts to work on graphs of small genus is a natural incremental step toward that big goal. In fact, it may be a necessary first step due to the Robertson-Seymour decomposition theory for graphs that forbid a fixed minor [35, 113]. At a high level, this theory states any graph avoiding a fixed minor $H$ is decomposable into clique-sums of bounded genus graphs augmented with a constant number of apices and vortices of fixed complexity. An apex of an embedded graph is a vertex adjacent to an arbitrary subset of vertices. A vortex of an embedded graph $G$ is a graph of bounded path width embedded into a face of $G$ in a particular way. A natural strategy for finding fast algorithms on graphs forbidding a fixed minor is to first find fast algorithms for all the pieces of the decomposition and then find a way to link the algorithms’ results together. This strategy of extending results for certain graph families to clique-sums of those families has been used explicitely before [21, 25].
1.1 Homology, Homotopy, and Related Results

In this dissertation, we describe several results on combinatorial optimization problems for graphs where the input comes with an embedding on an orientable surface of small genus. While the specific techniques used differ between problems, all the algorithms we describe share one common feature in that they rely on the algebraic topology construct of homology originally developed by Poincaré [110]. Homology provides a way to classify curves on surfaces such that two curves are considered equivalent if their difference bounds a weighted sum of faces. We give much more rigorous definitions of homology and the other concepts we need for our results in Chapter 2.

Homology and its closely related cousin homotopy are a popular source for problems and techniques within the realm of algorithms for surface embedded graphs. Informally, two paths with common endpoints or two cycles on a surface are homotopic if one can be continuously deformed on the surface to match the other. Just over 100 years ago, Max Dehn [34] described the first efficient algorithms to decide if two cycles are homotopic and to decide if a given cycle is contractible (homotopic to a point). Efficient implementations of Dehn’s algorithm run in linear time when the genus of the surface is fixed [41]. When the complexity of the surface itself is expressed as part of the input, an algorithm of Schipper [114] determines if a cycle is contractible in $O(gn + g^2\ell)$ time where $g$ is the genus of the surface and $\ell$ is the complexity of the cycle. The running time was improved by Dey and Schipper [40] to $O(n + \ell \log g)$. Dey and Guha [37] removed the dependency on $g$, improving the running time to an optimal $O(n + \ell)$. They also claimed a linear time algorithm to test if two cycles are homotopic, but Lazarus and Rivaud [96] found a subtle flaw in their algorithm. Lazarus and Rivaud described a different linear time algorithm to decide if two cycles are homotopic. Erickson and Whittlesey [54] describe simpler linear time algorithms for both the contractibility and homotopy test problems. In contrast to the above results, deciding whether a cycle or a collection of cycles is null-homologous or determining if two collections of cycles are homologous in $O(gn)$ time is considerably easier. See for example Erickson and Whittlesey [53, Section 4] or Section 5.5 in Chapter 5.

When a graph has weights on its edges, it becomes natural to ask for shortest representatives within certain homotopy or homology classes. Colin
de Verdière and Erickson [32] described an algorithm to compute the shortest cycle homotopic to a given cycle in $O(gn\ell \log n\ell)$ time, generalizing and improving an earlier result of Colin de Verdière and Lazarus for simple cycles [33]. The minimum $s,t$-cut algorithms of Chambers et al. [22], Erickson and Nayyeri [51], and Italiano et al. [77] work by computing a minimum-cost even subgraph in a given $\mathbb{Z}_2$-homology class, a problem which is NP-hard but fixed-parameter tractable with respect to genus [22]. We use similar strategies to compute global minimum cuts in Chapter 3. Chen and Friedman [27, 28] proved that this problem becomes NP-hard to approximate within any constant factor when generalized to simplicial complexes. Following a strategy first suggested by Sullivan [120], Chambers et al. [23] describe an algorithm to quickly find minimum-cost circulations in any given real or integer homology class in a directed surface graph. Their observations are key to an algorithm we give in Chapter 6 for computing maximum $s,t$-flows. Their result was subsequently generalized by Dey et al. [38] to work with arbitrary chains of arbitrary dimension in arbitrary simplicial complexes. For more related results, see [29, 39, 42, 49, 53].

Finally, we note that there is extensive work in finding shortest cycles that are homotopically or homologically non-trivial [14, 15, 18, 47, 49, 51, 77, 95, 124]. See Chapter 4 for more details. Cabello et al. [17] give an $O(n \log n)$ time algorithm to find a single cycle that is shortest for its own non-trivial homotopy class, although that cycle may not be the shortest non-contractible cycle. Cabello et al. [20] give an algorithm to compute a shortest splitting cycle, a non-crossing closed walk which is null-homologous but also non-contractible. Erickson and Worah [55] give an algorithm to compute a shortest essential cycle. Cabello [12] in a sense considers a problem of the opposite type: find a shortest simple cycle that is trivial with regard to homotopy or homology. Cabello, Colin de Verdière, and Lazarus [16] show a linear time algorithm to decide the existence of simple cycles that are either contractible, non-contractible, or non-separating while also showing the hardness of deciding existence for separating or splitting cycles.
1.2 Organization and Results

We begin by giving a formal overview of the definitions and tools used throughout the dissertation in Chapter 2. We generally describe techniques used for specific problems within each problem’s chapter.

In Chapter 3, we describe an algorithm to compute global minimum cuts in edge weighted genus $g$ graphs in $O(g) n \log \log n$ time. This chapter is based on work done with Jeff Erickson and Amir Nayyeri [48]. When the genus is a constant, our algorithm’s running time matches the best time bound known for planar graphs due to Łacić and Sankowski [97]. In our algorithm, we use an observation of Chambers et al. [22] to reduce to the problem of finding a minimum weight separating subgraph in the dual graph. We consider the cases where either (1) some minimum weight separating subgraph consists of a simple contractible cycle or (2) all minimum weight separating subgraphs can be decomposed into non-contractible simple cycles. We then use known and novel observations on how the minimum weight separating subgraph interacts with different types of shortest non-trivial curves to form our algorithm.

In Chapter 4, we describe algorithms to compute short non-trivial cycles in edge weighted graphs. Some of the algorithms are tailored to take advantage of undirected edge weights, but others are designed to work in graphs where the edges are directed. The results of this chapter previously appeared in a paper at the ACM/SIAM Symposium on Discrete Algorithms [58]. For undirected graphs embedded on surfaces of genus $g$, our algorithms compute non-separating, non-contractible, and non-null-homologous cycles in $2O(g)n \log \log n$ time, improving the previous best algorithms of Italiano et al. [77] which run in $O(g)n \log \log n$ time. For directed graphs, we give an algorithm to compute shortest non-null-homologous cycles in $O(g^2n \log n)$ time, matching the running time of Erickson’s [47] algorithm for computing shortest non-separating cycles. We also give an $O(g^3n \log n)$ time algorithm for computing shortest non-contractible cycles in directed graphs. This last result improves upon the previous best algorithm by Erickson [47] which runs in $O(g)n \log n$ time. All of these results require lifting the input graphs to subsets of appropriate covering spaces. In particular, we use and prove several useful properties about a covering space known as the infinite cyclic cover for our directed non-contractible cycle result.
In Chapter 5, we describe an algorithm to count minimum $s,t$-cuts based on work done with Erin W. Chambers and Amir Nayyeri [24]. Despite the problem being \#P-complete in general graphs, our algorithm runs in $2^{O(g)n^2}$ time. After running our algorithm once it becomes possible to sample minimum cuts uniformly at random in $O(n \log n)$ time per sample. This result directly generalizes an $O(n^2)$ time algorithm for counting minimum $s,t$-cuts in planar graphs by Bezáková and Friedlander [4]. Like Bezáková and Friedlander, we reduce the problem of counting minimum $s,t$-cuts to one of counting forward $t,s$-cuts in an embedded directed acyclic graph. Forward $t,s$-cuts separate $t$ from $s$ and have no edges spanning the cut backwards from the $s$ side to the $t$ side. We require many new observations related to homology with integer coefficients in order to efficiently count the forward cuts. In addition, we have to modify the surface itself in a non-trivial way to guarantee that the counting algorithm returns the correct result.

Finally, we describe ongoing work toward computing maximum $s,t$-flows in Chapter 6. This work is done in collaboration with Jeff Erickson. Borradaile and Klein [9] describe an $O(n \log n)$ time algorithm to compute maximum $s,t$-flows in planar graphs. In their algorithm, they send flows down left-most augmenting paths. Through use of the correct data structures, their algorithm performs each augmentation and selects the new augmentation to perform in $O(\log n)$ amortized time each. We give a new algorithm that appears to generalize their techniques naturally to surfaces with positive genus, and similar to their algorithm we are able to perform augmentations and find the next set of augmentations to perform in $O(g \log n + g^2)$ amortized time each. We prove that our algorithm performs a quadratic number of augmentations, giving it an overall time bound of $O(gn^2(\log n + g))$. While we have been unable to find a proof so far, we believe our algorithm may actually run in near-linear time when the surface has constant genus.
Chapter 2

Preliminary Material

We begin by recalling several useful definitions related to surface-embedded graphs. For further background, we refer the reader to Gross and Tucker [66] or Mohar and Thomassen [102] for topological graph theory, and to Hatcher [72] or Stillwell [119] for surface topology and homology. Parts of the presentation of our terminology and the notation used are taken directly from previous works [22, 47, 51, 52].

2.1 Surfaces and Curves

A surface (more formally, a 2-manifold with boundary) is a compact Hausdorff space in which every point has an open neighborhood homeomorphic to either the plane $\mathbb{R}^2$ or a closed halfplane $\{(x, y) \in \mathbb{R}^2 \mid x \geq 0\}$. The points with halfplane neighborhoods make up the boundary of the surface; every component of the boundary is homeomorphic to a circle. A surface is non-orientable if it contains a subset homeomorphic to the Möbius band, and orientable otherwise. In this dissertation, we consider only compact, connected, and orientable surfaces.

A path in a surface $\Sigma$ is a continuous function $p: [0,1] \rightarrow \Sigma$. 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 internally disjoint from the boundary of $\Sigma$ whose endpoints lie on the boundary of $\Sigma$. A cycle is a continuous function $\gamma: S^1 \rightarrow \Sigma$; the only difference between a cycle and a loop is that a loop has a distinguished basepoint. We say a loop $\ell$ and a cycle $\gamma$ are equivalent if, for some real number $\delta$, we have $\ell(t) = \gamma(t + \delta)$ for all $t \in [0,1]$. We collectively refer to paths, loops, arcs, and cycles as curves. A curve is simple if it is injective; we usually do not distinguish between simple curves and their images in $\Sigma$. A simple curve $p$ is separating if $\Sigma \setminus p$
The reversal $\text{rev}(p)$ of a path $p$ is defined by setting $\text{rev}(p)(t) = p(1-t)$. 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 $t \geq 1/2$. Finally, let $p[x,y]$ denote the subpath of a path $p$ from point $x$ to point $y$.

The genus of a surface $\Sigma$ is the maximum number of disjoint simple cycles in $\Sigma$ whose complement is connected. Up to homeomorphism, there is exactly one orientable surface and one non-orientable surface with any genus $g \geq 0$ and any number of boundary cycles $b \geq 0$. Orientable surfaces with $b$ boundary components are differentiated by their Euler characteristic $\chi = 2 - 2g - b$ (for non-orientable surfaces, $\chi = 2 - g - b$). See Figure 2.1.

2.2 Graph Embeddings

An embedding of an undirected graph $G = (V,E)$ on a surface $\Sigma$ maps vertices to distinct points and edges to simple, interior-disjoint paths. The faces of the embedding are maximal connected subsets of $\Sigma$ that are disjoint from the image of the graph. We may denote an edge $uv \in E$ as $f|g$ if it is incident to faces $f$ and $g$. An embedding is cellular if each of its faces is homeomorphic to the plane; in particular, in any cellular embedding, each component of the boundary of $\Sigma$ must be covered by a cycle of edges in $G$. Euler’s formula implies that any cellularly embedded graph with $n$ vertices, $m$ edges, and $f$ faces lies on a surface with Euler characteristic $\chi = n - m + f$, which implies that $m = O(n + g)$ and $f = O(n + g)$ if the graph is simple. We consider only such cellular embeddings of genus $g = O(n^{1-\varepsilon})$, so that the overall complexity of the embedding is $O(n)$.

Any cellular embedding on an orientable surface can be encoded combina-
Figure 2.2: Graph duality. One edge $uv$ and its dual $(uv)^* = f^*g^*$ are emphasized.

torially by a rotation system, which records the counterclockwise order of edges incident to each vertex. Two paths or cycles 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 redundantly use the term arc to refer to a walk in the graph whose endpoints are boundary vertices. Likewise, we use the term cycle to refer to a closed walk in the graph. Note that cycles may contain the same vertex or edge more than once.

Any undirected graph $G$ embedded on a surface $\Sigma$ without boundary has a dual graph $G^*$, which has a vertex $f^*$ for each face $f$ of $G$, and an edge $e^*$ for each edge $e$ in $G$ joining the vertices dual to the faces of $G$ that $e$ separates. The dual graph $G^*$ has a natural cellular embedding in $\Sigma$, whose faces correspond to the vertices of $G$. See Figure 2.2. For any subgraph $F = (U,D)$ of $G = (V,E)$, we write $G \setminus F$ to denote the edge-complement $(V,E \setminus D)$. We also abuse notation by writing $F^*$ to denote the subgraph of $G^*$ corresponding to any subgraph $F$ of $G$. Further, we may sometimes use $D$ to refer to an edge set or the subgraph $F = (V,D)$, but it should be clear which we mean from context.

A tree-cotree decomposition $(T,L,C)$ of an undirected graph $G$ embedded on a surface without boundary is a partition of the edges into three disjoint subsets; a spanning tree $T$ of $G$, a spanning cotree $C$ (the dual of a spanning tree $C^*$ of $G^*$), and leftover edges $L = G \setminus (T \cup C)$. Euler’s formula implies that in any tree-cotree decomposition, the set $L$ contains exactly $2g$ edges [45]. The definitions for dual graphs and tree-cotree decompositions given above extend to surfaces with boundary, but we do not require these extensions in this dissertation.

For some of the problems we consider in Chapters 4 and 6, the input is actually a directed edge-weighted (or capacitated) graph $G$ with a cellular embedding on some surface. We use the notation $u \rightarrow v$ to denote the directed
dart from vertex $u$ to vertex $v$, and let $\vec{E}$ be the set of darts in $G$. Without loss of generality, we consider only symmetric directed graphs, in which the reversal $v \rightarrow u$ of any dart $u \rightarrow v$ is another dart, possibly with infinite weight (or 0 capacity). We also assume that in the cellular embedding, the images of any edge in $G$ and its reversal coincide (but with opposite orientations). The two darts $u \rightarrow v$ and $v \rightarrow u$ therefore define an edge $uv$ with a canonical orientation $u \rightarrow v$; edge $vu$ does not necessarily exist even though $uv$ does. Thus, like Cabello et al. [15] and Erickson [47], we implicitly model directed graphs as undirected graphs with asymmetric edge weights. We may denote dart $u \rightarrow v$ as $f \uparrow g$ if faces $f$ and $g$ lie to its left and right respectively. The dual of any dart $f \uparrow g$ is $f^* \rightarrow g^*$. Note that the duality of darts is not an involution the way we have specified the orientation of the dual dart here.

Let $p = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_k$ be a simple directed cycle or arc in an embedded graph $G$. We say an edge $u \rightarrow v_i$ enters $p$ from the left (resp. right) if the vertices $v_{i-1}, u,$ and $v_{i+1}$ (module $k$ in the case of a cycle) are ordered clockwise (resp. counterclockwise) around $v_i$, according to the embedding’s rotation system. An edge $v_i \rightarrow u$ leaves $p$ from the left (resp. right) if its reversal $u \rightarrow v_i$ enters $p$ from the left (resp. right). If $p$ is an arc, the above definitions require that $0 < i < k$ and that $u$ is not a vertex in $p$. Recall an arc’s endpoints lie on boundary cycles. Let $t_0v_0$ and $v_0w_0$ be the boundary edges incident to $v_0$ with vertices $t_0$, $v_1$, and $w_0$ appearing in clockwise order around $v_0$. We say $t_0 \rightarrow v_0$ enters $p$ from the left. We say $w_0 \rightarrow v_0$ enters $p$ from the right. Similarly, if $t_kw_k$ and $v_kw_k$ are boundary edges incident to $v_k$ with vertices $t_k$, $w_k$, and $v_{k-1}$ appearing in clockwise order around $v_k$, we say $t_k \rightarrow v_k$ enters $p$ from the left and $w_k \rightarrow v_k$ enters $p$ from the right. Finally, we treat $t_0$ as $v_{-1}$ and $t_k$ as $v_{k+1}$ to define entering from the left (resp. right) for any other edges $u \rightarrow v_0$ or $u \rightarrow v_k$ where $u$ does not appear in $p$.

### 2.3 Homotopy and Homology

Two paths $p$ and $q$ in $\Sigma$ are homotopic if one can be continuously deformed into the other without changing their endpoints. More formally, a homotopy between $p$ and $q$ is a continuous map $h : [0, 1] \times [0, 1] \rightarrow \Sigma$ such that $h(0, \cdot) = p$, $h(1, \cdot) = q$, $h(\cdot, 0) = p(0) = q(0)$, and $h(\cdot, 1) = p(1) = q(1)$. Homotopy defines an equivalence relation over the set of paths with any
Figure 2.3: Left: A contractible cycle on $\Sigma$. Center: A non-contractible but separating cycle on $\Sigma$. Right: A non-contractible and non-separating cycle on $\Sigma$.

fixed pair of endpoints. The set of homotopy classes of loops in $\Sigma$ with basepoint $x_0$ defines a group $\pi_i(\Sigma, x_0)$ under concatenation, called the fundamental group of $\Sigma$. (For all basepoints $x_0$ and $x_1$, the groups $\pi_i(\Sigma, x_0)$ and $\pi_i(\Sigma, x_1)$ are isomorphic.) A cycle is contractible if it is homotopic to a constant map. See Figure 2.3. Given a weight function on the darts of $G$, we say a directed path or cycle is tight if it has minimum total weight (counting edges with multiplicity) for its homotopy class.

Homology is a coarser equivalence relation than homotopy, with nicer algebraic properties, but more subtle definitions. We optionally refer to the vertices of $G$ as cells of dimension 0, the edges as cells of dimension 1, and the faces as cells of dimension 2. A $k$-chain is a function that assigns values from some abelian group $G$ to the cells of dimension $k$. Let $\phi : E \to G$ be a 1-chain. Then, the boundary of $\phi$ is a 0-chain $\partial \phi : V \to G$ defined as $\partial \phi(v) = \sum_{(uv) \in E} \phi(uv) - \sum_{(uv) \in E} \phi(uv)$ for each $v \in V$. A 1-cycle is a 1-chain $\phi$ such that $\partial \phi(v) = 0$ for all $v \in V$. The cycle space of a graph $G$ with respect to a group $G$, denoted by $Z_1(G, G)$, is the vector space of 1-chains that are 1-cycles in $G$. The cycle space $Z_1(G, G)$ is isomorphic to $G^{|E| - |V| + 1}$. Below, we give alternatives to the phrase 1-cycle that depend upon the choice of group $G$ in order to avoid confusion with cycles as defined earlier in this chapter.

The boundary of a 2-chain $\alpha : F \to G$ is a 1-chain $\partial \alpha : E \to G$ such that $\partial \alpha(uv) = \text{right}(uv) - \text{left}(uv)$, where $\text{left}(uv)$ and $\text{right}(uv)$ are the faces to the left and right of the canonical orientation $u \to v$. It is straightforward to verify that the boundary of any 2-chain is a 1-cycle. The boundary space of $G$, denoted by $B_1(G, G)$ is the space of all boundary 1-cycles. It follows from the definition that $B_1(G, G)$ is a linear subspace of $Z_1(G, G)$, and it
is isomorphic to $G^{|F|-1}$ if $b = 0$ or $G^{|F|}$ if $b \geq 1$, where $b$ is the number of boundary components in $\Sigma$.

Two 1-chains $\phi$ and $\psi$ are homologous, or they are in the same homology class, if and only if their piecewise difference $\phi - \psi$ is a boundary 1-cycle. We define the homology space, $H_1(G, G)$, as the vector space of homology classes of 1-cycles. We have $H_1(G, G) \cong \mathbb{Z}_1(G, G) \cong G^2g + \max\{0, b-1\}$ by Euler’s formula. See Figure 2.4.

Depending on our application, we use different choices for $G$. In Chapters 3 and 4, we consider $G = \mathbb{Z}_2$ or cellular homology with coefficients in $\mathbb{Z}_2$. Homology with coefficients in $\mathbb{Z}_2$ is considered in several other works on surface-embedded graph [22,26,47,51], and this choice of coefficients actually simplifies some of the earlier definitions. A 1-chain is simply a subset of edges. We refer to 1-cycles as even subgraphs, and boundaries of 2-chains as null-homologous even subgraphs. An even subgraph is null-homologous if it is the boundary of the closure of the union of a subset of faces of $G$. Therefore, two even subgraphs $\eta$ and $\eta'$ are homologous if their symmetric difference $\eta \oplus \eta'$ is null-homologous. Note that if $b \leq 1$, then a simple cycle $\gamma$ is separating if and only if it is null-homologous; however, when $b > 1$, some separating cycles are not null-homologous. Given a weight function $w : E \to \mathbb{R}$ on the edges, we say an even subgraph $\eta$ is $\mathbb{Z}_2$-minimal if its edges have minimum total weight among all even subgraphs homologous to $\eta$.

In Chapters 5 and 6, we consider $G \in \{\mathbb{Z}, \mathbb{R}\}$ or cellular homology with coefficients from $\{\mathbb{Z}, \mathbb{R}\}$. Here, we say a $k$-chain is trivial if it assigns 0 to all cells, it is non-negative if it assigns non-negative values to all cells, and it is a $(0, 1)$-chain if it assigns values from $\{0, 1\}$ to all cells. Further, for any 1-chain $\phi : E \to G$, we can extent $\phi$ to a function on the darts by letting $\phi(u \to v) = \phi(uv)$ and $\phi(v \to u) = -\phi(uv)$. We refer
to 1-cycles as \textit{circulations} and the boundaries of 2-chains as \textit{boundary circulations} or when the context is clear, simply boundaries. A \textit{trivial circulation/boundary}, a \textit{non-negative circulation/boundary} and a \textit{(0,1)-circulation/boundary} are special cases of such 1-chains. We say a set of cycles \( C \) \textit{trivially generates} a circulation \( \phi \) if \( \phi \) is the 1-chain that assigns a value to each edge equal to the number of times the edge appears in \( C \) oriented along with a cycle minus the number of times it appears oriented opposite a cycle. Note that if \( C \) contains only cycles that respect edge orientations, then it generates a non-negative circulation. We may abuse terminology by equating \( C \) with the circulation trivially generated by \( C \). We sometimes refer to the homology class of a set of cycles, where we more precisely mean the homology class of the circulation trivially generated by that set.

In Chapter 6 we will often refer to \( k \)-chains both in the primal graph and in the dual graph. We have 0, 1, and 2-cochains referring to assignments to the dual faces, dual edges, and dual vertices respectively. To remove ambiguity, we often refer to 1-chains whose duals are circulations in \( G^* \) as \textit{cocirculations}. A \textit{coboundary} is a 1-chain whose dual is a boundary circulation in \( G^* \). Finally, two 1-chains are \textit{cohomologous} or in the same \textit{cohomology class} if their difference is a coboundary.

### 2.4 Flows and Cuts

Consider again homology with coefficients in \( \mathbb{R} \). We will often refer to 1-chains as \textit{flows} or \textit{coflows} depending on if we are focusing on the primal or dual graph. Any flow can be decomposed to a set of weighted paths and cycles. For any capacity function \( c : \vec{E} \to \mathbb{R} \) on the \textit{darts}, we say a flow \( \phi \) is \textit{feasible} if \(-c(v\to u) \leq \phi(uv) \leq c(u\to v)\). Note that capacity functions do not necessarily have to be non-negative for feasibility to be well defined. Given a coflow \( \theta \) we define the capacity of \( \theta \) with respect to \( c \) as follows. Let \( c' : E \to \mathbb{R} \) be a function on the edges such that \( c'(uv) = c(u\to v) \) if \( \theta(uv) \geq 0 \) and \( c'(uv) = -c(v\to u) \) if \( \theta(uv) < 0 \). The capacity of \( \theta \) with respect to \( c \) is the dot product \( \langle \theta, c' \rangle \).

For \( s, t \in V \), an \( s,t \)-flow is a 1-chain \( \phi : E \to \mathbb{R} \) such that \( \partial \phi(v) = 0 \) for all \( v \in V \setminus \{s, t\} \). The value of an \( s,t \)-flow \( \phi \) is \( \sum_{s\to v} \phi(s\to v) \). A \textit{maximum}
**s, t-flow** with respect to a capacity function \( c \) is a feasible flow of highest value. For a flow \( \phi \), the **residual capacity** function \( c_\phi : \vec{E} \to \mathbb{R} \) is defined as \( c_\phi(u \to v) = c(u \to v) - \phi(u \to v) \). Given a capacity function \( c \), we may refer to the **residual graph** \( G_\phi \) when discussing the graph \( G \) coupled with residual capacity function \( c_\phi \). The **dual residual graph** \( G^*_\phi \) is simply the dual graph \( G^* \) coupled with the residual capacity function \( c_\phi \).

An **cut** in \( G = (V, E) \) is defined as a subset of vertices \( S \subseteq V \); we refer to \( S \) and \( T = V \setminus S \) as different **sides** of the cut. Given two vertices \( s \) and \( t \) with \( s \in S \) and \( t \in T \), we say \( S \) is an **s,t-cut**. A dart \( u \to v \) (and its associated edge \( uv \) or \( vu \)) **crosses** a cut \( S \) if exactly one of \( u \) and \( v \) lie in \( S \). In particular, \( u \to v \) crosses \( S \) in the **forward** direction if \( u \in S \) and in the **backward** direction if \( v \in S \). For a cut \( S \) we use the notation \( \Gamma^+(S) \) to denote the set of all darts that cross \( S \) in the forward direction. We define \( \Gamma^-(S) \) as the set of darts that cross \( S \) in the backward direction. A walk \( W \) crosses a cut \( S \) \( k \) times if there are \( k \) edges of \( W \) that cross \( S \).

Given a dart capacity function \( c : \vec{E} \to \mathbb{R} \), the value or capacity of a cut \( S \) is \( \sum_{u \to v \in \Gamma^+(S)} c(u \to v) \). Equivalently, the capacity of a cut \( S \) is equal to the capacity of a coflow \( \theta \) such that \( \theta(u \to v) = 1 \) if \( u \to v \in \Gamma^+(S) \) and \( \theta(u \to v) = 0 \) if \( uv \) does not cross \( S \). A **minimum s,t-cut** of \( G \) with respect to capacity function \( c \) is an \( s,t \)-cut of minimum value. The well known maximum-flow/minimum-cut theorem of Ford and Fulkerson [57] states that for any non-negative capacity function \( c \), the value of a maximum feasible \( s,t \)-flow is equal to the value of a minimum \( s,t \)-cut.

### 2.5 Covering Spaces and Cutting

A continuous map \( \pi : \Sigma' \to \Sigma \) between two surfaces is called a **covering map** if each point \( x \in \Sigma \) lies in an open neighborhood \( U \) such that (1) \( \pi^{-1}(U) \) is a countable union of disjoint open sets \( U_1 \cup U_2 \cup \cdots \) and (2) for each \( i \), the restriction \( \pi|_{U_i} : U_i \to U \) is a homeomorphism. If there is a covering map \( \pi \) from \( \Sigma' \) to \( \Sigma \), we call \( \Sigma' \) a **covering space** of \( \Sigma \). The **universal cover** \( \tilde{\Sigma} \) is the unique simply-connected covering space of \( \Sigma \) (up to homeomorphism). The universal cover is so named because it covers every path-connected covering space of \( \Sigma \).

For any path \( p : [0, 1] \to \Sigma \) such that \( \pi(x') = p(0) \) for some point \( x' \in \Sigma' \),
there is a unique path \( p' \) in \( \Sigma' \), called a **lift** of \( p \), such that \( p'(0) = x' \) and \( \pi \circ p' = p \). We also say that \( p \) lifts to \( p' \). Conversely, for any path \( p' \) in \( \Sigma' \), the path \( \pi \circ p' \) is called a **projection** of \( p' \).

We define a lift of a cycle \( \gamma : S^1 \to \Sigma \) to be the infinite path \( \gamma' : \mathbb{R} \to \Sigma' \) such that \( \pi(\gamma'(t)) = \gamma(t \mod 1) \) for all real \( t \). We call the path obtained by restricting \( \gamma' \) to any unit interval a **single-period lift** of \( \gamma \); equivalently, a single-period lift of \( \gamma \) is a lift of any loop equivalent to \( \gamma \). We informally say that a cycle is the **projection** of any of its single-period lifts.

**Cutting** a combinatorial surface along a cycle or arc modifies both the surface and the embedded graph. For any combinatorial surface \( S = (\Sigma, G) \) and any simple cycle or arc \( \gamma \) in \( G \), we define a new combinatorial surface \( S \bowtie \gamma \) 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 \( \gamma \), each bordering a new boundary. Similar to covering spaces, we define the **projection** of a curve in \( S \bowtie \gamma \) as the natural mapping of points (or vertices and edges) to \( S \).
Chapter 3

Global Minimum Cuts

We begin the novel results portion of the dissertation by describing an algorithm to compute minimum cuts in surface embedded graphs. We may sometimes refer to the problem of computing a minimum cut as the *global* minimum cut problem.

The global minimum cut problem is closely related to the minimum $s,t$-cut problem. Obtaining a polynomial time algorithm for the global minimum cut problem is not too difficult given an algorithm for the minimum $s,t$-cut problem. We can specify a vertex $s$ in $G$ and calculate the minimum $s,t$-cut for every $t \in V \setminus \{s\}$. Using a recent algorithm of Orlin [106], we can compute a global minimum cut deterministically in $O(n^2 m)$ time.

These running times can be significantly improved by avoiding repeated calculations of minimum $s,t$-cuts. Nagamochi and Ibaraki gave a deterministic $O(nm + n^2 \log n)$ time algorithm for the global minimum cut problem. In a series of papers [82, 84, 85], Karger and Stein showed how randomization can be used to speed up the problem further; the best algorithm known runs in $O(m \log^3 n)$ time and returns a correct answer with high probability.

Naturally, planarity speeds up computation further. All planar graphs are sparse (the number of edges is linear in the number of vertices), so Karger’s algorithm [85] mentioned above runs in $O(n \log^3 n)$ time. Chalermsook, Fakcharoenphol, and Nanongkai [19] gave a deterministic algorithm that specifically relies on planarity and runs in $O(n \log^2 n)$ time. Their algorithm was later improved by Łącki and Sankowski [97] who achieved an $O(n \log \log n)$ running time.

In this chapter, we describe the first deterministic near-linear time algorithm to find global minimum cuts in graphs embedded on orientable surfaces of fixed genus. Specifically, given an $n$-vertex graph embedded on a surface of genus $g$, our algorithm runs in $g^{O(g)} n \log \log n$ time. When the genus $g$ is constant, our algorithm’s running time matches the recent algorithm of
Łącki and Sankowski for planar graphs [97]. In fact, our algorithm invokes both Łącki and Sankowski’s algorithm and the recent planar minimum $s, t$-cut algorithm of Italiano et al. [77]. Further improvements in both of these algorithms would immediately improve our algorithm as well.

It is well known that the edges crossing minimum cuts in planar graphs are dual to minimum length simple cycles. At a very high level, the planar-graph algorithms of Chalermsook et al. [19] and Łącki and Sankowski [97] use a divide-and-conquer strategy to find a minimum length simple cycle in the dual graph. Their algorithms split the dual graph into two pieces of roughly equal size with a simple cycle, and then recursively compute the minimum length cycle inside each piece. For the conquering step, Chalermsook et al. prove that the minimum-length cycle crosses the boundary between the two pieces at most twice. They use this fact to argue that a minimum-length cycle crossing the boundary must separate two specific faces and can therefore be found using an algorithm for minimum $s, t$-cuts (in the original primal graph).

Unfortunately, this divide-and-conquer approach does not immediately generalize to surfaces with positive genus. First, the edges crossing a minimum cut are no longer necessarily dual to a single cycle; second, because not all cycles in surface graphs bound disks, we cannot so easily divide the graph into equal-sized pieces that maintain the necessary crossing properties. To work around these obstacles, as in previous work on minimum $s, t$-cuts [22], we rely on properties of subgraphs that have minimum weight in their $\mathbb{Z}_2$-homology class. In particular, in Sections 3.2 and 3.3, we prove several properties of $\mathbb{Z}_2$-minimal subgraphs that can possibly cross the dual of the minimum cut; these properties may be of independent interest. Throughout the remainder of this chapter, we only refer to homology with coefficients in $\mathbb{Z}_2$.

**Algorithm Summary**: Our algorithm relies on a similar observation to that given for computing minimum $s, t$-cuts [22]. Namely, the edges crossing a minimum cut are dual to a minimum weight null-homologous subgraph that is separating. We say a null-homologous even subgraph $\eta$ is a *separating subgraph* if it contains at least one edge. We consider two cases for minimum weight separating subgraphs. Assuming there exists some minimum weight separating subgraph consisting of a single contractible cycle, we use observations of Cabello [12, Lemma 4.1] to cut the surface into a planar graph that
still contains a minimum weight separating subgraph. Our algorithm finds the subgraph using the minimum cut algorithm of Łacki and Sankowski [97]. In case there are no minimum weight separating subgraphs consisting of a single contractible cycle, our algorithm computes several $\mathbb{Z}_2$-minimal even subgraphs from different homology classes. We prove that some of these $\mathbb{Z}_2$-minimal even subgraphs can be used to find a pair of primal vertices $s$ and $t$ that are separated by a minimum cut. The minimum $s,t$-cut algorithm of Italiano et al. [77] is then be used to find a minimum cut.

3.1 Homology Cuts

We now give the high level overview of our algorithm for computing minimum cuts. To work with topology in computing a minimum cut, we use the following modification of a lemma of Chambers et al. [22, Lemma 3.1]. Recall that a null-homologous even subgraph $\eta$ is a separating subgraph if it contains at least one edge.

Lemma 3.1.1. Let $G$ be an undirected graph with non-negative edge capacities, cellularly embedded on a surface $\Sigma$ without boundary, let $S$ be a minimum cut in $G$, and let $C$ be the edges crossing $S$. Then $C^*$ is a minimum weight separating subgraph of $G^*$.

Proof: Let $C$ be the edges crossing an arbitrary cut in $G$. The cut partitions the vertices of $G$ into two disjoint subsets $S$ and $T$. Therefore, the dual subgraph $C^*$ partitions the faces of $G^*$ into two disjoint subsets $S^*$ and $T^*$. Further, $C^*$ is the boundary of the union of faces in $S^*$, implying that $C^*$ is null-homologous in $\Sigma$ and therefore separating.

Conversely, let $C^*$ be an arbitrary separating subgraph of $G^*$. As $C^*$ is null-homologous, it is the boundary of a subset of the faces of $G^*$. Moreover, because $C^*$ is non-empty, it must be the boundary of a proper, non-empty subset of faces. Let $s^*$ and $t^*$ be faces of $G^*$ on either side of $C^*$. Any path from $s$ to $t$ in the primal graph $G$ must traverse at least one edge of $C$. We conclude that $C$ crosses a cut (in particular, an $s,t$-cut). □

Fix an undirected graph $G = (V,E)$, a non-negative weight function $w: E \to \mathbb{R}$, and a cellular embedding of $G$ on a surface $\Sigma$ of genus $g$ with at
least two faces. In light of Lemma 3.1.1, we focus our attention on finding a minimum weight separating subgraph of $G$.

Our algorithm separately considers two cases, illustrated in Figure 3.1. Exactly one of these cases must apply to the minimum weight separating subgraph.

1. Some minimum weight separating subgraph consists of a single contractible simple cycle.

2. Every minimum weight separating subgraph can be decomposed into non-contractible simple cycles.

In the following sections, we describe two subroutines to find minimum weight separating subgraphs that are designed with their corresponding condition in mind. If the corresponding condition does hold, the subroutine will return a separating subgraph with weight at most that of the minimum weight separating subgraph. Otherwise, the subroutine may return a higher weight separating subgraph. By running both subroutines and returning the best result, we find a minimum weight separating subgraph no matter which category it falls into.

3.2 Contractible Cycle

We begin by describing an algorithm to handle the case where some minimum weight separating subgraph is a contractible simple cycle. We begin by borrowing a result of Cabello [12, Lemma 4.1].

**Lemma 3.2.1 (Cabello [12]).** Let $\alpha$ be a tight arc or tight cycle on $G$. There exists a shortest contractible simple cycle that does not cross $\alpha$. 

Figure 3.1: Two types of minimum weight separating subgraphs: a contractible cycle and otherwise.
Corollary 3.2.2. The shortest contractible simple cycle and the shortest non-separating cycle in $G$ do not cross.

Cabello [12] uses these observations in order to compute a shortest contractible simple cycle in a surface embedded graph. Unfortunately his algorithm takes $\Omega(n^2)$ time, because his algorithm must return a shortest contractible simple cycle even if it is not a minimum weight separating subgraph. Cabello et al. [17] use a similar procedure to find a shortest enclosing cycle which bounds a non-empty set of faces. While this procedure can be modified to run in $g^{O(g)}n\log\log n$ time, it may return a cycle that is actually trivial after taking the symmetric difference over all its edges with multiplicity. The cycle returned may not be a separating subgraph as per our definition.

Our algorithm will make use of the cutting operation ($\mathcal{C}$) along tight cycles and arcs in $G$. The following lemma implies it is safe for our algorithm to find minimum weight separating subgraphs in snipped copies of $\Sigma$.

Lemma 3.2.3. Let $\alpha$ be an arbitrary simple cycle or arc in $G$. Let $\Sigma_{\mathcal{C}} = \Sigma \cup \alpha$ and let $G_{\mathcal{C}} = G \cup \alpha$. Any null-homologous even subgraph $\gamma_{\mathcal{C}}$ in $G_{\mathcal{C}}$ projects to a null-homologous even subgraph in $G$.

Proof: Let $\gamma_{\mathcal{C}}$ be an arbitrary null-homologous even subgraph in $G_{\mathcal{C}}$ and let $\gamma$ be its projection in $G$. Subgraph $\gamma$ bounds a subset of faces $F_{\mathcal{C}}$ in $G_{\mathcal{C}}$. Let $F$ be the projection of $F_{\mathcal{C}}$ into $G$. We will argue that $\gamma$ bounds $F$, proving the lemma.

Consider any edge $e = f \uparrow g$ on the boundary of $F$. If $f$ and $g$ still lie adjacent along $e$ in $G_{\mathcal{C}}$, then $e$ bounds $F_{\mathcal{C}}$ and appears in $\gamma$. If $e$ separates a face $f$ from the boundary of $G_{\mathcal{C}}$, then $e$ still appears in $\gamma$.

Now consider any edge $e$ in $\gamma_{\mathcal{C}}$. Suppose $e$ does not lie along $\alpha$ so that its projection appears in $\gamma$. Edge $e$ separates two faces $f$ and $g$ in $G_{\mathcal{C}}$, and
exactly one of those faces appears in $F$. Now suppose $e$ does lie along $\alpha$.

Edge $e$ separates face $f \in F$ from the boundary of $G_{\mathcal{F}}$. The projection of $e$ may separate $f$ from another face $g$. If $g$ exists and is also in $F$, then there exists another edge $e'$ in $G_{\mathcal{F}}$ that separates $g$ from the boundary of $G_{\mathcal{F}}$. The projections of $e$ and $e'$ cancel each other when taking the symmetric difference so their projection does not appear in $\gamma$. Finally, if $g$ does not exist or $g$ is not a member of $F$, then there is not another edge that shares a projection with $e$. The projection of $e$ will exist in $\gamma$. \hfill \Box

We now present the main result of this section.

**Lemma 3.2.4.** There exists a $g^{O(g)} n \log \log n$ time algorithm that computes a minimum weight separating subgraph if any such subgraph is a contractible simple cycle. If not, the algorithm either returns some separating subgraph (that may not be minimum weight) or nothing.

**Proof:** Our algorithm begins by computing a shortest non-separating cycle $\alpha$ in $G$ in $g^{O(g)} n \log \log n$ time, using a modification of an algorithm of Kutz [95] by Italiano *et al.* [77] or using our own $2^{O(g)} n \log \log n$ time modification given in Chapter 4. The surface $\Sigma \mathcal{F} \alpha$ has two boundary cycles $\alpha'$ and $\alpha''$.

It then computes a system $P$ of tight arcs connecting $\alpha'$ and $\alpha''$ in $O(n)$ time using the shortest-path algorithm of Henzinger *et al.* [73], as described by Erickson and Nayyeri [51]. Let $G_{\mathcal{F}}$ denote the planar graph $G_{\mathcal{F}} (\alpha \cup P)$; this graph has $O(gn)$ vertices.

Pick an arbitrary edge $e$ of $\alpha$, and let $e_1$ and $e_2$ be distinct copies of $e$ in $G_{\mathcal{F}}$. Let $\gamma_1$ and $\gamma_2$ be the shortest simple cycles in the subgraphs $G_{\mathcal{F}} \setminus e_1$ and $G_{\mathcal{F}} \setminus e_2$, respectively. Our algorithm computes both $\gamma_1$ and $\gamma_2$ in $O(gn \log \log n)$ time using the algorithm of Läcki and Sankowski [97]. Note that graphs $G_{\mathcal{F}} \setminus e_1$ and $G_{\mathcal{F}} \setminus e_2$ may not contain any cycles. In this case, $G_{\mathcal{F}}$ contains no simple cycles. Corollary 3.2.2 and Lemma 3.2.1 imply $G$ does not contains any contractible simple cycles to begin with and our algorithm returns nothing. For the rest of this section, we assume $\gamma_1$ and $\gamma_2$ are well defined.

Let $\gamma$ be the shorter of the cycles $\gamma_1$ and $\gamma_2$. By multiple instantiations of Lemma 3.2.3, cycle $\gamma$ projects to a null-homologous closed walk $\gamma'$ in the original graph $G$, which may or may not be simple. Our algorithm returns the symmetric difference over all edges in $\gamma'$. The outer face of $G_{\mathcal{F}}$ is the only
Figure 3.3: At least one copy of $e$ is forbidden in the planarized graph.

face that is not also a face of $G$. It follows that the only separating cycle in $G_P$ that is not a separating subgraph in $G$ is the boundary of outer face. Because $\gamma$ avoids at least one edge of the outer face, the carrier of $\gamma'$ must be non-empty. If our algorithm returns anything, it must return a separating subgraph.

Now, suppose some minimum weight separating subgraph of $G$ is a contractible simple cycle. Corollary 3.2.2 and Lemma 3.2.1 imply that some shortest contractible simple cycle $\sigma$ in $G$ crosses neither $P$ nor $\alpha$. (We emphasize that our algorithm does not necessarily compute $\sigma$.) This cycle $\sigma$ appears as a simple cycle in $G_P$ that avoids at least one of the edges $e_1$ or $e_2$. Thus, $\sigma$ cannot be shorter than $\gamma$, and our algorithm returns a minimum weight separating subgraph. \[\square\]

3.3 Non-contractible Components

Next, we consider the case where all minimum weight separating subgraphs contain components that are non-contractible. The following lemma is the key result of this section and could likely have applications beyond this work.

**Lemma 3.3.1.** Let $\sigma$ be a minimum weight separating subgraph, and let $f$ be any face of $G$. Let $\gamma$ be a closed walk on $G$ that lies in the closure of the opposite component of $f$ in $\Sigma \setminus \sigma$, and let $\eta$ be a shortest even subgraph homologous to $\gamma$. There exists a minimum weight separating subgraph $\sigma'$ (possibly $\sigma$) such that $\eta$ lies in the closure of the opposite component of $f$ in $\Sigma \setminus \sigma'$. (See Figure 3.4.)

**Proof:** If $\sigma$ fits the requirement that $\eta$ lies in the closure of the opposite component of $f$ in $\Sigma \setminus \sigma$, then we are done. Assume otherwise. The sub-
Figure 3.4: The setting of Lemma 3.3.1. A $\mathbb{Z}_2$-minimal even subgraph $\eta$ is separated from face $f$ by a minimum weight separating subgraph $\sigma'$.

Graph $\sigma$ separates the faces of $G$ into two non-empty sets. Call the faces in the component of $\Sigma \setminus \sigma$ containing $f$ the far faces and call the rest of the faces near. Similarly, the even subgraph $\eta \oplus \gamma$ is null-homologous and separates the faces of $G$ into two subsets; call the faces in the subset containing $f$ black and the others white.

Let $\sigma'$ be the boundary of the union of the far black faces in $G$. By definition, $\sigma'$ is a null-homologous even subgraph. By assumption, $\eta$ has edges that are incident to two far faces, but $\gamma$ does not; thus, there is at least one far black face $f$. Since there is also at least one near face, $\sigma'$ is non-empty. No edge of $\eta$ lies between two far black faces so $\eta$ lies in the closure of the opposite component of $f$ in $\Sigma \setminus \sigma'$. We claim $\sigma'$ is a minimum weight even subgraph.

For the sake of contradiction, suppose $\sigma'$ is not a minimum weight even subgraph. Because both $\sigma'$ and $\sigma$ are null-homologous, the even subgraph $\eta' = \eta \oplus \sigma' \oplus \sigma$ is homologous to $\eta$, and therefore to $\gamma$.

For any subgraph $H$ of $G$, let $w(H)$ denote the sum of the weights of the edges of $H$. We now prove that $w(\sigma') + w(\eta') \leq w(\eta) + w(\sigma)$ by bounding the contribution of each edge $e \in E(G)$ to both sides of the inequality. Note that both $\sigma'$ and $\eta'$ are subgraphs of $\sigma \cup \eta$; moreover, $\sigma' \oplus \eta' = \sigma \oplus \eta$. There are three cases to consider.

- If $e \not\in \eta \cup \sigma$, then $e$ contributes 0 to both sides of the inequality.
- If $e \in \sigma \oplus \eta$, then $e \in \sigma' \oplus \eta'$. In this case, $e$ contributes $w(e)$ to both sides of the inequality.
- If $e \in \sigma \cap \eta$, then $e$ contributes exactly $2w(e)$ to the right side of the inequality. Trivially, $e$ contributes at most $2w(e)$ to the left side.

On the other hand, because $\sigma'$ is not a minimum weight separating sub-
graph, we must have $w(\sigma') > w(\sigma)$. It immediately follows that $w(\eta') < w(\eta)$, which contradicts the minimality of $\eta$. □

We now present the main result of this section, concluding the description of our algorithm for computing minimum weight separating subgraphs and minimum cuts.

**Lemma 3.3.2.** There exists a $O(g)n \log \log n$ time algorithm that computes a minimum weight separating subgraph if every minimum weight separating subgraph can be decomposed into non-contractible simple cycles. If not, the algorithm either returns some separating subgraph (that may not be minimum weight) or nothing.

**Proof:** Our algorithm begins by picking an arbitrary face $f$ of $G$. Let $\sigma$ be an arbitrary minimum weight separating subgraph. We argue that there exists a non-separating closed walk $\gamma$ in $G$ that lies in the closure of the opposite component of $f$ in $\Sigma \setminus \sigma$.

By assumption, $\sigma$ can be decomposed into simple cycles, each of which is non-contractible. Suppose $\sigma$ consists of more than one cycle. None of the cycles are null-homologous, because we could remove a single null-homologous cycle from $\sigma$ to lower its cost without changing its homology class. In this case, $\gamma$ is simply one of the cycles in $\sigma$’s decomposition. Now, assume $\sigma$ consists of a single simple cycle. Cycle $\sigma$ is not contractible by assumption, so neither component of $\Sigma \setminus \sigma$ is planar. The closure of the component opposite $f$ has non-zero genus and therefore contains a non-separating closed walk in $\Sigma$.

Let $\eta$ be a shortest even subgraph homologous to $\gamma$. By Lemma 3.3.1, we may assume without loss of generality that $\eta$ lies in the closure of the opposite component of $f$ in $\Sigma \setminus \sigma$. Assume for now that our algorithm knows $\eta$. We will remove this assumption later in the proof.

Our algorithm picks an arbitrary edge $e = h_1 \uparrow h_2$ of $\eta$. At least one of $h_1$ and $h_2$ lies in the closure of the opposite component of $f$ in $\Sigma \setminus \sigma$. Our algorithm computes minimum weight subgraphs separating $h_1$ from $f$ and $h_2$ from $f$ using the minimum $s,t$-cut algorithm of Italiano et al. [77]. It then returns the cheaper of these two subgraphs, which weighs no more than $\sigma$.

We now remove the assumption that our algorithm knows $\eta$. Non-separating even subgraph $\eta$ is shortest for one of $2^g - 1$ homology classes. Our algorithm
enumerates all $2^{2g} - 1$ homology classes by sampling subsets of cycles from a homology basis [45]. For each homology class $x$, it finds the shortest even subgraph $\eta_x$ and runs the subroutine described in the previous paragraph assuming $\eta = \eta_x$. If there exists no subgraph separating the arbitrarily picked edge $e \in \eta_x$ from $f$ (in other words, $e = f \uparrow f$), then the subroutine correctly returns nothing for that choice of homology class. The algorithm returns the least weight separating subgraph returned by any instantiation of the subroutine or nothing if no instantiation returns a separating subgraph. One of the homology classes contains $\eta$, so the algorithm will eventually find the minimum weight separating subgraph assuming every minimum weight separating subgraph can be decomposed into non-contractible simple cycles. □

By running the algorithms described in Lemmas 3.2.4 and 3.3.2, we get the main results of this chapter.

**Theorem 3.3.3.** A minimum weight separating subgraph of an undirected $n$-vertex graph embedded on an orientable surface of genus $g$ can be computed in $g^{O(g)}n \log \log n$ time.

**Corollary 3.3.4.** A minimum cut in an undirected $n$-vertex graph embedded on an orientable surface of genus $g$ can be computed in $g^{O(g)}n \log \log n$ time.

### 3.4 Conclusions and Open Problems

Our algorithm repeatedly applies three recent $O(n \log \log n)$-time algorithms for planar and surface graphs as black boxes: one due to Lacki and Sankowski for global minimum cuts [97], one due to Italiano et al. for minimum $(s,t)$-cuts [77], and our own algorithm given in Chapter 4 for shortest non-separating cycles. Indeed, these are the only subroutines in our algorithm that require more than linear time when the genus is fixed. Thus, improvements to any of these algorithms would immediately improve our algorithm as well.

Although our algorithm works in near-linear time for graphs of constant genus, the complexity dependence on the genus is exponential. This exponential dependence is unavoidable with our current technique, as our algorithm calls a subroutine that solves an NP-hard problem: finding the minimum
weight subgraph in a given $\mathbb{Z}_2$-homology class [22]. We optimistically conjecture that global minimum cuts in surface graphs can be computed in $O(g^k n \log \log n)$ time for some small constant $k$, using different techniques.
Chapter 4

Shortest Non-trivial Cycles

We now turn our attention to computing shortest non-trivial cycles. Let $G = (V, E)$ be an arbitrary connected graph embedded on a surface $\Sigma$ of genus $g$ with $b$ boundary cycles. We are primarily interested in computing shortest cycles in $G$ whose images on $\Sigma$ are either non-separating or non-contractible. Cabello and Mohar [18] claim that finding short non-trivial cycles is arguably one of the most natural problems for graphs embedded on a surface. Additionally, finding these cycles has many benefits both for theoretical combinatorial problems [6,36,87,102] and more practical applications in areas such as graphics and graph drawing [10,49,69,75,89,128].

The history of finding non-trivial cycles in undirected graphs goes back several years to a result of Itai and Shiloach [76]. They give an $O(n^2 \log n)$ time algorithm to find a shortest non-trivial cycle in an annulus as a subroutine for computing minimum $s,t$-cuts in planar graphs. Their result has seen several improvements, most recently by Italiano et al. [59,77,112]. Thomassen [124] gave the first efficient algorithm for computing short non-trivial cycles on surfaces with arbitrary genus. His algorithm runs in $O(n^3)$ time and relies on a property of certain families of cycles known as the 3-path condition; see also Mohar and Thomassen [102, Chapter 4]. Erickson and Har-Peled [49] gave an $O(n^2 \log n)$ time algorithm, which remains the fastest known for graphs of arbitrary genus. Cabello and Mohar [18] gave the first results parameterized by genus, and Kutz [95] showed it is possible to find short non-trivial cycles in time near-linear in the number of vertices if we allow an exponential dependence on the genus. Kutz’s algorithm requires searching $g^{O(g)}$ subsets of the universal cover. Cabello, Chambers, and Erickson [14] later showed the near-linear time dependence is possible with only a polynomial dependence on the genus by avoiding use of the universal cover. The current best running time in terms of the number of vertices is $g^{O(g)} n \log \log n$ due to a modification to Kutz’s algorithm by Italiano...
et al. [77].

Unfortunately, all of the above results rely on properties that exist only in undirected graphs; shortest paths intersect at most once (assuming uniqueness), and the reversal of any shortest path is a shortest path. Due to the difficulty in avoiding these assumptions, there are few results for finding shortest non-trivial cycles in directed surface graphs, and all of these results are relatively recent. Befittingly, the short history of these results appears to coincide nicely with the history given above for undirected graphs.

Janiga and Koubek [78] gave the first near-linear time algorithm for computing a shortest non-trivial cycle in a directed graph embedded on an annulus as an attempt to find minimum $s,t$-cuts in planar graphs\(^1\). Their result can also be achieved using recent maximum flow algorithms for planar graphs [9, 46, 126].

Cabello, Colin de Verdière, and Lazarus [15] gave the first efficient algorithms for computing shortest non-trivial cycles in directed surface graphs of arbitrary genus. Their algorithms run in $O(n^2 \log n)$ time and $O(\sqrt{g}n^{3/2} \log n)$ time, and rely on a variant of the 3-path condition and balanced separators, respectively. Erickson and Nayyeri [51] gave a $2^{O(g)}n \log n$ time algorithm for computing the shortest non-separating cycle that relies on computing the shortest cycle in each of $2^{O(g)}$ homology classes. The latest results for these problems are two algorithms of Erickson [47]. The first algorithm computes shortest non-separating cycles in $O(g^2 n \log n)$ time by computing shortest paths in several copies of a linear sized covering space. The second algorithm computes shortest non-contractible cycles (which may be separating) in $g^{O(g)}n \log n$ time in a manner similar to Kutz’s algorithm [95], by lifting the graph to a finite (but large) subset of the universal cover.

In both the undirected and directed graph settings, researchers presented near-quadratic time algorithms for computing shortest non-separating and non-contractible cycles, and others supplemented them with algorithms with exponential dependence in the genus, but near-linear dependence in the complexity of the embedded graph. Similar trends appear in the computation of maximum flows and minimum cuts in surface embedded graphs [22, 23, 48, 51, 64, 118]. For the problems mentioned in this paragraph, we ideally would like algorithms with a near-linear dependency on graph com-

---

\(^1\)Unfortunately, their minimum cut algorithm has a subtle error [81] which may lead to an incorrect result when the minimum $t, s$-cut is smaller than the minimum $s, t$-cut.
plexity but only a polynomial dependence on genus. Of course, we are still interested in pushing down the dependence on graph complexity even if it means sacrificing a bit in the genus dependency when $g$ is sufficiently small.

4.1 New Results

Our first result is improved algorithms for computing non-trivial cycles in undirected surface graphs. Our algorithms run in $2^{O(g)}n \log \log n$ time and can be used to find shortest non-separating, non-contractible, or non-null-homologous cycles. These algorithms improve the running times achieved by Italiano et al. [77] for finding shortest non-separating and non-contractible cycles and show that it is possible to take advantage of the universal cover as in Kutz’s algorithm in order to minimize the dependency on $n$, without searching a super-exponential in $g$ number of subsets of the covering space. For surfaces with $b$ boundary cycles, the shortest non-contractible and non-null-homologous cycle algorithms run in time $2^{O(g+b)}n \log \log n$, while the shortest non-separating cycle algorithm continues to run in $2^{O(g)}n \log \log n$ time. The main idea behind these algorithms is to construct fewer subsets of the universal cover by only constructing subsets corresponding to certain weighted triangulations of a dualized polygonal schema as in [20, 22]. These algorithms are described in Section 4.2.

Next, we describe an algorithm to compute a shortest non-null-homologous cycle in a directed surface graph in $O((g^2 + gb)n \log n)$ time. This algorithm is actually a straightforward extension to Erickson’s algorithm for computing shortest non-separating cycles [47], but we must work out some non-trivial details for the sake of completeness. This algorithm is given in Section 4.3. Along with being an interesting result in its own right, we use this algorithm as a subroutine for our primary result described below.

Our final, primary, and most technically interesting result is an $O(g^3n \log n)$ time algorithm for computing shortest non-contractible cycles in directed surface graphs, improving the result of Erickson [47] for all positive $g$ and showing it is possible to have near-linear dependency in graph complexity without suffering an exponential dependency on genus. On a surface with $b$ boundary cycles, our algorithm runs in $O((g^3 + gb)n \log n)$ time. In order to achieve this running time, we choose to forgo using a subset of the universal
cover in favor of subsets of a different covering space known as the infinite cyclic cover. Our description of the infinite cyclic cover and its properties appears in Sections 4.4 and 4.5. The algorithm is given in Section 4.6.

**Algorithm Summaries:** Our algorithms for computing shortest non-trivial cycles in undirected graphs are heavily based on the improvements by Italiano et al. [77] to the algorithm of Kutz [95]. As in Kutz’s algorithm, our algorithm finds a set of loops that crosses the shortest non-trivial cycles few times. Cutting along these loops turns the surface into a disk, and the shortest non-trivial cycles cross the disk as non-self-crossing arcs. Our algorithm enumerates the ways these arcs can cross the disk, and finds the shortest cycles for each set of crossings.

Our algorithm for computing shortest non-null-homologous cycles in directed graphs is a straightforward extension to Erickson’s algorithm for computing shortest non-separating cycles [47]. If the surface contains no boundary, our algorithm just returns the same result as Erickson’s algorithm. If the surface does contain boundary, it uses appropriate covering spaces to find the shortest cycles that cross paths between boundary components an odd number of times.

Finally, our algorithm for computing shortest non-contractible cycles in directed graphs takes advantage of a covering space called the infinite cyclic cover. Assuming all shortest non-contractible cycles are separating and that the surface has exactly one boundary, our algorithm computes a set of non-separating cycles similar to the algorithm for undirected graphs. We argue that for at least one of these non-separating cycles, we can build an instance of the infinite cyclic cover where a shortest non-contractible cycle lifts to a shortest non-null-homologous cycle; if it lifts to a separating cycle, then it separates a finite set of faces and a boundary component from the rest of the covering space. Our algorithm builds $O(n)$ subsets of the infinite cyclic cover and uses the non-null-homologous cycle algorithm to search each of these subsets.
4.1.1 Assumptions and Tools

To simplify our presentation and analysis for the algorithms on directed graphs, we assume that any two vertices \( x \) and \( y \) in \( G \) are connected by a unique shortest directed path, denoted \( \sigma(x, y) \). The Isolation Lemma [104] implies that this assumption can be enforced (with high probability) by perturbing the edge weights with random infinitesimal values [49].

Our algorithms rely on a result by Cabello et al. [14] which generalizes a result of Klein [93] for planar graphs.

**Lemma 4.1.1 (Cabello et al. [14]).** Let \( G \) be a directed graph with non-negative edge weights, cellularly embedded on a surface \( \Sigma \) of genus \( g \), and let \( f \) be an arbitrary face of \( G \). We can preprocess \( G \) in \( O(g n \log n) \) time and \( O(n) \) space with high probability, so that the length of any shortest path from any vertex incident to \( f \) to any other vertex can be retrieved in \( O(\log n) \) time.

Our algorithms rely on the fact that the families of trivial cycles defined here follow a variant of the \( 3 \)-path condition [102, Sec. 4.3]. Given three \( x, y \)-paths \( \alpha, \beta, \) and \( \gamma \) such that \( \alpha \cdot \text{rev}(\beta) \) and \( \beta \cdot \text{rev}(\gamma) \) are both trivial, the cycle \( \alpha \cdot \text{rev}(\gamma) \) is also trivial.

4.2 Shortest Non-trivial Cycles in Undirected Graphs

Let \( G \) be an undirected graph with positive edge weights, cellularly embedded on an orientable surface \( \Sigma \) of genus \( g \). We sketch an algorithm to compute a shortest non-separating, non-contractible, or non-null-homologous cycle in \( G \). We assume the surface has no boundary, and consider the case with boundary at the end of this section. Recall any shortest non-null-homologous cycle is a shortest non-separating cycle in a surface without boundary.

We begin by reviewing Kutz’s [95] algorithm for computing shortest non-trivial cycles. Kutz begins by computing a greedy system of loops \( \Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{2g}\} \) using a construction of Erickson and Whittlesey [53]. The construction can be performed in \( O(g n) \) time using our assumption that \( g = O(n^{1-\varepsilon}) \) [95]. The surface \( D = \Sigma \setminus \Lambda \) is a topological disk with each loop \( \lambda_i \in \Lambda \) appearing twice upon its boundary. See Figure 4.1. Kutz argues that there exists some shortest non-trivial cycle \( \gamma \) that meets three criteria: (1) \( \gamma \)
Given a cycle $\gamma$, there exists a sequence of crossings between $\gamma$ and the loops of $\Lambda$. Kutz uses the above observations to find shortest cycles corresponding to $g^{O(g)}$ crossing sequences of length $O(g)$ where at least one of the crossing sequences corresponds to a shortest non-trivial cycle. For each crossing sequence $X$, he describes how to determine if a cycle corresponding to $X$ meets the criteria above and, if so, how to find a shortest cycle corresponding to $X$ in $O(gn \log n)$ time using an algorithm of Colin de Verdière and Erickson [32]. Italiano et al. [77] later improved the running time of Colin de Verdière and Erickson’s algorithm to $O(n \log \log n)$. The final running time for Kutz’s algorithm with the modification by Italiano et al. is therefore $g^{O(g)}n \log \log n$.

In order to improve the running time, we show how to reduce the number of crossing sequences that need to be considered by Kutz’s algorithm using a similar strategy to that seen in [20, 22]. See also Section 5.5 in Chapter 5. As mentioned, the greedy system of loops $\Lambda$ used by Kutz cuts the surface into a topological disk $D$. By replacing each loop in $\Lambda$ with a single edge in $D$, we transform $D$ into an abstract polygonal schema. Each loop of $\Lambda$ corresponds to two edges of the polygon. Any non-self-crossing cycle $\gamma$ in $\Sigma$ is cut into arcs by the polygon where an arc exists between two edges if $\gamma$ consecutively crosses the corresponding loops of $\Lambda$. We dualize the polygonal schema by replacing each edge with a vertex and each vertex with an edge.
Cycle $\gamma$ now corresponds to a *weighted triangulation* of the dualized polygonal schema where each pair of consecutive crossings by $\gamma$ between loops of $\Lambda$ is represented by an edge between the corresponding vertices. Each edge of the triangulation receives a weight equal to the number of times $\gamma$ performs the corresponding consecutive crossings. Some shortest non-trivial cycle crosses each member of $\Lambda$ at most twice, so the edge weights on its triangulation are all between 0 and 2.

Our algorithm for computing a shortest non-trivial cycle in $G$ enumerates all weighted triangulations of the dualized polygonal schema with weights between 0 and 2 by brute force. There are $2^{O(g)}$ weighted triangulations considered. For each triangulation, the algorithm then checks if it corresponds to a single cycle in $O(g)$ time by brute force. If the triangulation does correspond to a single cycle, then its crossing sequence is calculated. The algorithm uses Italiano et al.’s [77] modification to Kutz’s [95] algorithm to determine if the crossing sequence meets the aforementioned criteria and, if so, to calculate a shortest cycle corresponding to that crossing sequence. Our algorithm will eventually return a shortest cycle corresponding to the correct crossing sequence for some shortest non-trivial cycle. The overall running time is $2^{O(g)}n \log \log n$.

### 4.2.1 Surfaces with Boundary

We now extend the above algorithm to work on surfaces with boundary. For computing a shortest non-separating cycle, we reduce to the case without boundary by pasting disks into each of the boundary components. This transformation does not change the set of non-separating cycles. Our algorithm still runs in time $2^{O(g)}n \log \log n$.

In order to compute a shortest non-contractible cycle or non-null-homologous cycle, we use a *greedy system* $\Lambda$ of $O(g + b)$ arcs [20, 31, 32, 47, 51] instead of a greedy system of loops. The necessary properties of the greedy system still hold as detailed in the following lemmas.

**Lemma 4.2.1.** All shortest non-contractible and non-null-homologous cycles are simple.

**Proof:** Both contractible and null-homologous cycles follow the 3-path condition, implying the lemma [15, Lemma 3].
Lemma 4.2.2. There exist shortest non-contractible and non-null-homologous cycles that cross each arc $\lambda$ of a greedy system of arcs at most twice.

Proof: Our proof closely follows that of [95, Lemma 1]. We prove the lemma for any family of cycles following the 3-path condition. Let $\gamma$ be a shortest non-trivial cycle that minimizes the total number of crossings with arcs in a greedy system of arcs $\Lambda$. If no arc is crossed more than two times, we are done. Otherwise, let $\lambda$ be an arc crossed more than two times. Arc $\lambda$ consists of the concatenation of two shortest paths $p_1$ and $p_2$ with an additional edge [22]. One of the two shortest paths is crossed at least twice by $\gamma$ by assumption. We assume $p_1$ is crossed twice without loss of generality.

Let $u$ and $v$ be vertices of $p_1$ that lie in distinct crossings with $\gamma$. Let $\gamma_1$ and $\gamma_2$ be internally disjoint $u,v$-paths such that $\gamma = \gamma_1 \cdot \text{rev}(\gamma_2)$, and let $\beta = p_1[u,v]$. Path $\beta$ is a shortest path since it is a subpath of shortest path $p$. One of $\gamma_1 \cdot \text{rev}(\beta)$ or $\beta \cdot \text{rev}(\gamma_2)$ must be non-trivial or $\gamma$ would be trivial by the 3-path condition. Further, both $\gamma_1 \cdot \text{rev}(\beta)$ and $\beta \cdot \text{rev}(\gamma_2)$ are no longer than $\gamma$ as $\beta$ is a shortest path. Finally, both $\gamma_1 \cdot \text{rev}(\beta)$ and $\beta \cdot \text{rev}(\gamma_2)$ have at least one fewer crossing with $\Lambda$. \[\square\]

Lemma 4.2.3. There exist shortest non-contractible and non-null-homologous cycles that cross each arc $\lambda$ of a greedy system of arcs at most twice and contain no curls.

Proof: Our proof closely follows that of [95, Lemma 3]. Let $\gamma$ be a non-contractible (non-null-homologous) cycle that crosses each arc $\lambda$ of a greedy system of arcs $\Lambda$ at most twice such that it has a minimal number of crossings. Suppose for contradiction there exists a curl with arc $\lambda$. Let $u$ and $v$ be vertices of $\lambda$ that each lie distinct crossings defining the curl, and let $\gamma'$ be the $u,v$-subpath of $\gamma$ within the curl. Let $\beta$ be the subpath of $\lambda$ between $u$ and $v$. Path $\beta$ is tight, because $\lambda$ is tight [20]. Cutting the surface $\Sigma$ along $\Lambda$ creates a disk containing the cycle $\gamma' \cdot \text{rev}(\beta)$. Therefore, $\gamma'$ and $\beta$ are homotopic. Path $\gamma'$ can be replaced by $\beta$ within $\gamma$ without changing its homotopy or homology class and without increasing its length. Performing the replacement also reduces the number of crossings with $\Lambda$, creating a contradiction. \[\square\]

Given that the three properties of the greedy system still hold, the rest of the algorithm remains essentially unchanged. We still use a dualized polyg-
onal schema, except it now has $O(g+b)$ vertices, and our algorithm must enumerate $2^{O(g+b)}$ weighted triangulations. To test if a crossing sequence corresponds to a non-null-homologous cycle, our algorithm uses techniques shown in [22]. The overall running time is $2^{O(g+b)} n \log \log n$.

With these extensions to surfaces with boundary, we get the following theorem.

**Theorem 4.2.4.** A shortest non-separating cycle in an $n$-vertex undirected graph embedded on an orientable surface of genus $g$ with $b$ boundary cycles can be computed in $2^{O(g)} n \log \log n$ time. Further, a shortest non-contractible or non-null-homologous cycle can be computed in $2^{O(g+b)} n \log \log n$ time.

### 4.3 Shortest Non-null-homologous Cycles in Directed Graphs

Now let $G$ be a symmetric directed graph with positive dart weights, cellularly embedded on an orientable surface $\Sigma$ of genus $g$ with $b$ boundary cycles. We continue by giving an overview of an algorithm to compute a shortest cycle in $G$ that is not null-homologous.

In [47], Erickson describes a system $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{2g}\}$ of $2g$ non-separating cycles where each cycle $\lambda_i$ is composed of two shortest paths in $G$ along with an extra edge. We actually describe and use this construction explicitly in Section 4.5. For each cycle $\lambda_i \in \Lambda$, Erickson gives an $O(gn \log n)$ time algorithm to find a shortest cycle that crosses $\lambda_i$ an odd number of times. Any non-separating cycle must cross at least one member of $\Lambda$ an odd number of times, so an $O(g^2n \log n)$ time algorithm for finding a shortest non-separating cycle follows immediately.

In a similar vain, we claim it is possible to compute in $O(gn \log n)$ time a shortest cycle crossing any non-separating arc $\lambda$ an odd number of times assuming $\lambda$ is a shortest path. Our algorithm for finding a shortest non-null-homologous cycle begins by calling Erickson’s algorithm as a subroutine in case any shortest non-null-homologous cycles are non-separating. We then perform the following steps in case all the shortest non-null-homologous cycles are separating. Arbitrarily label the boundary cycles of $G$ as $B_0, B_1, \ldots, B_{b-1}$. Let $s$ be an arbitrary vertex on $B_0$. We compute the
shortest path tree $T$ from $s$ using Dijkstra’s algorithm in $O(n \log n)$ time. For each index $i \geq 1$, let $\lambda_i$ be a shortest directed path in $T$ from $B_0$ to $B_i$ that contains exactly one vertex from each boundary cycle $B_0$ and $B_i$. Let $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{b-1}\}$ be the set of shortest paths computed above. Each path must be non-separating as it connects two distinct boundary cycles. We can easily compute $\Lambda$ in $O(bn)$ time once we have the shortest path tree $T$. If a shortest non-null-homologous cycle is separating, then it must separate $B_0$ from some other boundary cycle $B_i$ with $i \geq 1$.

**Lemma 4.3.1.** If a simple cycle $\gamma$ separates boundary cycle $B_0$ from a different boundary cycle $B_i$, then $\lambda_i$ crosses $\gamma$ an odd number of times.

**Proof:** Cycle $\gamma$ separates $\Sigma$ into two components $A$ and $B$ containing boundary cycles $B_0$ and $B_i$ respectively. Arc $\lambda_i$ must cross $\gamma$ from $A$ to $B$ one more time than it crosses from $B$ to $A$. Therefore, $\lambda_i$ crosses $\gamma$ an odd number of times. \[\Box\]

Lemma 4.3.1 implies that any shortest non-null-homologous cycle $\gamma$ crosses some arc $\lambda_i$ an odd number of times if $\gamma$ is separating. In order to use the above lemma, we describe an extension to the cyclic double cover of Erickson [47] that works with simple arcs instead of cycles. Let $\lambda$ be an arbitrary simple non-separating arc in $\Sigma$.

Define the covering space $\Sigma^2_\lambda$, which we call the **cyclic double cover**\(^2\) as follows. Cutting the surface $\Sigma$ along $\lambda$ gives us a new surface $\Sigma'$ with at least one boundary cycle. One boundary cycle of $\Sigma'$ contains two copies of $\lambda$ denoted $\lambda^+$ and $\lambda^-$. Let $(\Sigma',0)$ and $(\Sigma',1)$ denote two distinct copies of $\Sigma'$. For any point $p \in \Sigma'$, let $(p,0)$ and $(p,1)$ denote the corresponding points in $(\Sigma',0)$ and $(\Sigma',1)$, respectively. In particular, let $(\lambda^+,0)$ and $(\lambda^-,0)$ denote the copies of $\lambda^+$ and $\lambda^-$ in $(\Sigma',0)$. Finally, let $\Sigma^2_\lambda$ be the surface obtained by identifying $(\lambda^-,0)$ and $(\lambda^+,1)$ to a single arc, denoted $(\lambda,0)$, and identifying $(\lambda^-,1)$ and $(\lambda^+,0)$ to a single arc, denoted $(\lambda,1)$. To remove ambiguity, for any vertex $s$ in $\lambda$, we say $(\lambda,0)$ contains $(s,0)$ and $(\lambda,1)$ contains $(s,1)$. Any graph $G$ that is cellularly embedded in $\Sigma$ lifts to a graph $G^2_\lambda$ with twice as many vertices and edges that is cellularly embedded in $\Sigma^2_\lambda$. There are also twice as many faces in the embedding of $G^2_\lambda$ on $\Sigma^2_\lambda$ and at least $2b-2$ boundary cycles, so Euler’s formula implies the genus of $\Sigma^2_\lambda$ is at most $2g$. See Figure 4.2.

\(^2\)Named for the cyclic group of order 2.
Figure 4.2: Left: An arc $\lambda$ between two boundary on a torus. Right: The cyclic double cover $\Sigma^2_{\lambda}$.

For combinatorial surfaces, we can equivalently define the cyclic double cover using a standard voltage construction [66, Chapters 2,4]. Here, we assume $\lambda$ is an arc in $G$. For any directed edge $u \to v$, we define $\varepsilon_{\lambda}(u \to v)$ to be 1 if $u \to v$ enters $\lambda$ from the left or leaves $\lambda$ from the left, and 0 otherwise.

Let $G^2_{\lambda}$ be the graph whose vertices are the pairs $(v, z)$, where $v$ is a vertex of $G$ and $z$ is a bit, and whose edges are the ordered pairs

$$(u \to v, z) := (u, z) \to (v, z \oplus \varepsilon_{\lambda}(u \to v))$$

for all edges $u \to v$ of $G$ and both bits $z$. Here, $\oplus$ denotes addition modulo 2.

Let $\pi : G^2_{\lambda} \to G$ denote the obvious covering map $\pi(v, z) = v$. We declare that a cycle in $G^2_{\lambda}$ bounds a face of $G^2_{\lambda}$ if and only if its projection to $G$ bounds a face of $G$. The resulting embedding of $G^2_{\lambda}$ defines the cyclic double cover $\Sigma^2_{\lambda}$. For any directed cycle $\gamma$, we define the crossing parity $\varepsilon_{\lambda}(\gamma)$ to be 1 if $\gamma$ crosses $\lambda$ an odd number of times and 0 otherwise. Equivalently, we have

$$\varepsilon_{\lambda}(\gamma) = \bigoplus_{u \to v \in \gamma} \varepsilon_{\lambda}(u \to v).$$

As in [47], the following lemmas are immediate.

**Lemma 4.3.2.** Let $\lambda$ be any simple non-separating arc in $\Sigma$; let $\gamma$ be any cycle in $\Sigma$; and let $s$ be any vertex of $\gamma$. Then $\gamma$ is the projection of a unique path in $\Sigma^2_{\lambda}$ from $(s, 0)$ to $(s, \varepsilon_{\lambda}(\gamma))$.

**Lemma 4.3.3.** Let $\lambda$ be any simple non-separating arc in $\Sigma$. Every lift of a shortest directed path in $G$ is a shortest directed path in $G^2_{\lambda}$.

**Lemma 4.3.4.** Let $\lambda$ be any simple non-separating arc in $\Sigma$; let $\gamma$ be the shortest cycle in $\Sigma$ that crosses $\lambda$ an odd number of times; and let $s$ be any
vertex of $\gamma$. Then $\gamma$ is the projection of a shortest path in $\Sigma^2_\lambda$ from $(s,0)$ to $(s,1)$.

All that remains is to present a slightly modified lemma of Erickson [47, Lemma 3.4].

**Lemma 4.3.5.** Let $\lambda$ be any arc in $\Lambda$. The shortest cycle $\gamma$ that crosses $\lambda$ an odd number of times can be computed in $O(gn\log n)$ time.

**Proof:** The proof remains essentially unchanged for our version of the lemma, but we present it here for completeness. Let $\lambda = s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_k$, and let $s_i$ be the lowest index vertex of $\lambda$ that lies on $\gamma$. By Lemma 4.3.5, $\gamma$ lifts to a shortest path $\gamma_\lambda$ in $\Sigma^2_\lambda$ from $(s_i,0)$ to $(s_i,1)$. If $\gamma_\lambda$ uses any other vertex $(s_j,0)$ then $\gamma_\lambda$ uses the entire shortest path $(s_i,0)$ to $(s_j,0)$ which is a subpath of $(\lambda,0)$. Path $\gamma_\lambda$ begins with a subpath of $(\lambda,0)$ and is disjoint from $(\lambda,0)$ otherwise.

Consider the surface $\Sigma^2_\lambda \setminus (\lambda,0)$ obtained by not identifying $(\lambda^-,0)$ and $(\lambda^+,1)$ in the construction of $\Sigma^2_\lambda$. The previous paragraph implies that $\gamma_\lambda$ is the shortest path between some vertex in $(\lambda^-,0)$ or $(\lambda^+,1)$ and some vertex $(s_i,1)$. All such vertices in $(\lambda^-,0)$ and $(\lambda^+,1)$ lie on the same face of $\Sigma^2_\lambda \setminus (\lambda,0)$, and we can compute all these shortest paths in $O(gn\log n)$ time with high probability according to Lemma 4.1.1. \qed

Applying Lemma 4.3.5 to each arc $\lambda \in \Lambda$ and comparing the results to the shortest non-separating cycle found by Erickson’s algorithm, we immediately get Theorem 4.3.6.

**Theorem 4.3.6.** A shortest non-null-homologous cycle in an $n$-vertex directed graph embedded on an orientable surface of genus $g$ with $b$ boundary cycles can be computed in $O((g^2 + gb)n\log n)$ time with high probability.

### 4.4 The Infinite Cyclic Cover

As in the previous section, let $G$ be a symmetric directed graph with positive dart weights, cellularly embedded on an orientable surface $\Sigma$ of genus $g$ with $b$ boundary cycles. We begin to describe our algorithm for computing a shortest non-contractible cycle in $G$. Our job is easy if any shortest
non-contractible cycle is non-null-homologous; we can just run the algorithm given in Section 4.3 in $O((g^2 + gb) n \log n)$ time. We must work harder, though, to find a shortest non-contractible cycle $\gamma$ if every shortest non-contractible cycle is null-homologous. Our high-level strategy is to construct $O(g)$ subsets of a covering space we call the infinite cyclic cover. In Lemma 4.6.1, we show at least one of the subsets contains a non-null-homologous cycle that projects to $\gamma$.

Let $\lambda$ be an arbitrary simple non-separating cycle in $\Sigma$. We define the covering space $\Sigma_\lambda$, which we call the **infinite cyclic cover**\textsuperscript{3}, as follows. Cutting the surface $\Sigma$ along $\lambda$ gives us a new surface $\Sigma'$ with $b + 2$ boundary cycles where two of the boundary cycles are copies of $\lambda$ denoted $\lambda^+$ and $\lambda^-$. The infinite cyclic cover is obtained by pasting together an infinite number of copies of $\Sigma'$ along corresponding boundary cycles $\lambda^\pm$. Specifically, we have a copy $(\Sigma', i)$ of $\Sigma'$ for each integer $i$. Let $(\lambda^+, i)$ and $(\lambda^-, i)$ denote copies of $\lambda^+$ and $\lambda^-$ in $(\Sigma', i)$. The infinite cyclic cover is defined by identifying $(\lambda^+, i)$ and $(\lambda^-, i + 1)$ for every $i$. Any graph $G$ cellularly embedded on $\Sigma$ lifts to an infinite graph $G_\lambda$ embedded in $\Sigma_\lambda$. Note that for any pair of simple non-separating cycles $\lambda$ and $\mu$, the infinite cyclic covers $\Sigma_\lambda$ and $\Sigma_\mu$ are homeomorphic, but the lifted graphs $G_\lambda$ and $G_\mu$ may not be isomorphic.

We would like to use the infinite cyclic cover to aid us in finding a shortest non-contractible cycle. As explained in Section 4.5, it is possible to consider only a finite portion of $\Sigma_\lambda$ if we choose $\lambda$ carefully. We call this subset the **restricted infinite cyclic cover**. Again, let $\lambda$ be an arbitrary simple non-separating cycle in $\Sigma$ and define $\Sigma'$ as above with boundaries $\lambda^+$ and $\lambda^-$. Instead of pasting together an infinite number of copies of $\Sigma'$, we only paste together five copies. Specifically, we have a copy $(\Sigma', i)$ of $\Sigma'$ for each integer $i \in \{1, \ldots, 5\}$. Again, let $(\lambda^+, i)$ and $(\lambda^-, i)$ denote copies of $\lambda^+$ and $\lambda^-$ in $(\Sigma', i)$. The restricted infinite cyclic cover is defined by identifying $(\lambda^+, i)$ and $(\lambda^-, i + 1)$ for every $i \in \{1, \ldots, 4\}$. See Figure 4.3. Now any graph $G$ cellularly embedded on $\Sigma$ lifts to a finite graph $G^r_\lambda$ embedded in $\Sigma^r_\lambda$ with at most six times as many vertices and edges. Note that $\Sigma^r_\lambda$ still has two lifts of $\lambda$ acting as boundary cycles. We continue to refer to these boundary cycles as $\lambda^+$ and $\lambda^-$ when it is clear from context that we are referring to the restricted infinite cyclic cover. Euler’s formula implies the genus of $\Sigma^r_\lambda$.

\textsuperscript{3}Named for the infinite cyclic group.
Further restrict $\lambda$ to be a simple non-separating cycle in $G$. For any path or cycle $p$, we define the **crossing count** $c_\lambda(p)$ to be the number of times $p$ crosses $\lambda$ from left to right *minus* the number of times $p$ crosses $\lambda$ from right to left. Equivalently, we have

$$c_\lambda(p) = \sum_{u \rightarrow v \in p} c_\lambda(u \rightarrow v)$$

where for any directed edge $u \rightarrow v$, we define $c_\lambda(u \rightarrow v)$ to be 1 if $u \rightarrow v$ *enters* $\lambda$ from the left, $-1$ if $u \rightarrow v$ *leaves* $\lambda$ from the left, and 0 otherwise. We can define the restricted infinite cyclic cover using a *voltage construction* [66, Chapters 2, 4] for combinatorial surfaces. Let $G_\lambda^r$ be the graph whose vertices are the pairs $(v, i)$, where $v$ is a vertex of $G$ and $i$ is an integer in $\{1, \ldots, 6\}$ if $v$ lies along $\lambda$ or $\{1, \ldots, 5\}$ if $v$ does not lie along $\lambda$. The edges of $G_\lambda^r$ are the ordered pairs

$$(u \rightarrow v, i) := (u, i) \rightarrow (v, i + c_\lambda(u \rightarrow v))$$

for all edges $u \rightarrow v$ of $G$ and all $i \in \{1, \ldots, 6\}$. Let $\pi : G_\lambda^r \rightarrow G$ denote the obvious covering map $\pi(v, i) = v$. We declare that a cycle in $G_\lambda^r$ bounds a face of $G_\lambda^r$ if and only if its projection to $G$ bounds a face of $G$. The resulting
embedding of $G_r^\lambda$ defines the restricted infinite cyclic cover $\Sigma_r^\lambda$.

4.5 Lifting Shortest Non-contractible Cycles

Consider the following procedure also used in [47]. We construct a greedy tree-cotree decomposition $(T, L, C)$ of $G$, where $T$ is a shortest path tree rooted at some arbitrary vertex of $G$. Euler’s formula implies that $L$ contains exactly $2g$ edges; label these edges arbitrarily as $u_1v_1, u_2v_2, \ldots, u_2gv_2g$. For each index $i$, let $\lambda_i$ denote the unique cycle in the undirected graph $T \cup u_iv_i$ oriented so that it contains the directed edge $u_i \rightarrow v_i$. If there are no boundary cycles in $\Sigma$, then the set of cycles $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{2g}\}$ is a basis for the first homology group of $\Sigma$ [45]. We refer to the construction as a partial homology basis. Every non-separating cycle in $\Sigma$ crosses at least one cycle in $\Lambda$ an odd number of times [18, Lemma 3]. The greedy tree-cotree decomposition $(T, L, C)$ can be constructed in $O(n \log n)$ time using Dijkstra’s algorithm. Afterward, we can easily compute the partial homology basis in $O(gn)$ time.

Recall that a single period lift of a cycle $\gamma$ to a covering space refers to any lift of a loop equivalent to $\gamma$. Let $\sigma$ be an arbitrary shortest path in $G$. Erickson [47] argues that a lift of any shortest non-contractible cycle to the universal cover does not intersect many lifts of $\sigma$. This observation applies to the infinite cyclic cover as well. The following lemma and its corollary are essentially equivalent to Lemma 4.6 and Corollary 4.7 of Erickson [47], but modified for our setting.

**Lemma 4.5.1.** Let $\gamma$ be a shortest non-contractible cycle in $\Sigma$; let $\lambda$ be any simple non-separating cycle in $\Sigma$; and let $\sigma$ be any shortest path in $\Sigma$. Any single-period lift of $\gamma$ to the infinite cyclic cover $\Sigma_\lambda$ intersects at most two lifts of $\sigma$.

**Proof:** The covering space $\Sigma_\lambda$ is path connected, so it is itself covered by the universal cover $\tilde{\Sigma}$. Any single period lift of $\gamma$ to $\Sigma_\lambda$ in turn has one or more lifts in $\tilde{\Sigma}$. Any one of these single period lifts of $\gamma$ to $\tilde{\Sigma}$ intersects at most two lifts of $\sigma$ [47, Lemma 4.6]. Covering maps are functions, so lifting from $\Sigma_\lambda$ to $\tilde{\Sigma}$ cannot decrease the number of intersecting lifts of $\sigma$. \qed
Corollary 4.5.2. Let $\Lambda$ be a partial homology basis in $\Sigma$; let $\lambda$ be any cycle in $\Lambda$; and let $\gamma$ be a shortest non-contractible cycle in $\Sigma$. Any single-period lift of $\gamma$ to $\Sigma_\lambda$ intersects at most four lifts of $\lambda$.

Proof: Every vertex of $\lambda$ belongs to one of two directed shortest paths. By Lemma 4.5.1, any single-period lift of $\gamma$ intersects at most two lifts of either shortest path. \qed

Recall the restricted infinite cyclic cover defined in Section 4.4 is constructed by pasting together five copies of the surface cut along the simple non-separating cycle $\lambda$. We immediately get the following lemma stating the restricted infinite cyclic cover is large enough to contain a lift of any shortest non-contractible cycle.

Lemma 4.5.3. Let $\Lambda$ be a partial homology basis in $\Sigma$; let $\lambda$ be any cycle in $\Lambda$; and let $\gamma$ be a shortest non-contractible cycle in $\Sigma$. There exists a single period lift of $\gamma$ to $\Sigma_r^\lambda$.

In fact, we show below that $\gamma$ lifts to be a shortest non-contractible cycle in $\Sigma^\lambda_r$ if $\gamma$ is separating. This statement actually holds for any non-separating cycle $\lambda$ made of two shortest paths optionally connected by an edge. In Lemma 4.6.1, we explain that the correct choice of $\lambda$ guarantees the lift of $\gamma$ to be non-null-homologous.

We continue by noting that every shortest non-contractible cycle is simple [15, Lemma 3]. We show that if any shortest non-contractible cycle $\gamma$ is separating, then it lifts to a cycle in $\Sigma^\lambda_r$ for any $\lambda$ in the partial homology basis. Recall the definition of the crossing count $c_\lambda(\gamma)$.

Lemma 4.5.4. Let $\lambda$ be any simple non-separating cycle in $\Sigma$, and let $\gamma$ be a loop in $\Sigma$ with a lift in $\Sigma^\lambda_r$. Then, $\gamma$ lifts to a loop in $\Sigma^\lambda_r$ if and only if $c_\lambda(\gamma) = 0$.

Proof: Let $\Sigma'$ be the surface $\Sigma$ cut along $\lambda$. By construction, $\Sigma^\lambda_r$ is composed of five copies of $\Sigma'$, denoted $(\Sigma', i)$ for each integer $i \in \{1, \ldots, 5\}$. Each copy is separated by a lift of $\lambda$. Consider a lift of $\gamma$ contained in $\Sigma^\lambda_r$ which we denote $\gamma^\lambda$. For every instance of $\gamma$ crossing $\lambda$ from left to right, there is an instance of $\gamma^\lambda$ crossing a lift of $\lambda$ from $(\Sigma', i)$ to $(\Sigma', i + 1)$ for some $i$. Likewise, every time $\gamma$ crosses $\lambda$ from right to left, $\gamma^\lambda$ crosses $\lambda$ from $(\Sigma', i)$
Lemma 4.5.5. Let $\lambda$ be any simple non-separating cycle in $\Sigma$ and let $\gamma$ be any simple separating cycle. We have $c_\lambda(\gamma) = 0$.

Proof: Cycle $\gamma$ separates $\Sigma$ into two components denoted $A$ and $B$ so that a path crossing $\gamma$ exactly once starts in $A$ and ends in $B$ if it crosses from left to right. Let $x$ be an arbitrary point on $\lambda$ and consider the loop $\ell$ equivalent to $\lambda$ based at $x$. Every time $\ell$ crosses $\gamma$ from left to right, we see $\ell$ goes from $A$ to $B$. Further $\gamma$ crosses $\ell$ once from right to left. Similarly, every time $\ell$ crosses $\gamma$ from right to left, we see $\ell$ goes from $B$ to $A$ and $\gamma$ crosses $\ell$ once from left to right. Loop $\ell$ must cross from $A$ to $B$ the same number of times it crosses from $B$ to $A$. Therefore, $\gamma$ crosses $\ell$ and $\lambda$ from right to left the same number of times it crosses left to right. By definition, $c_\lambda(\gamma) = 0$. □

Corollary 4.5.6. Let $\Lambda$ be a partial homology basis in $\Sigma$; let $\lambda$ be any cycle in $\Lambda$; and let $\gamma$ be a shortest non-contractible cycle in $\Sigma$. If $\gamma$ is separating, then $\gamma$ lifts to a loop in $\Sigma_\lambda^r$.

We can finally show that if any shortest non-contractible cycle $\gamma$ is separating, then it actually lifts to a shortest non-contractible cycle in $\Sigma_\lambda^r$ for any $\lambda$ in a partial homology basis.

Lemma 4.5.7. Let $\gamma_\lambda$ be a loop in $\Sigma_\lambda^r$ that projects to a simple loop $\gamma$ in $\Sigma$. Loop $\gamma_\lambda$ is contractible if and only if $\gamma$ is contractible.

Proof: Suppose $\gamma_\lambda$ is contractible. There exists a homotopy $h$ from $\gamma_\lambda$ to a constant map. The paths in $h$ can be projected to $\Sigma$, yielding a homotopy from $\gamma$ to a constant map. Therefore, $\gamma$ is contractible.

Now, suppose $\gamma$ is contractible. There exists a homotopy $h$ from $\gamma$ to a constant map. There exists a unique homotopy $h_\lambda$ of $\gamma_\lambda$ that lifts the paths in $h$ to the infinite cyclic cover $\Sigma_\lambda$ [72, Proposition 1.30]. Homotopy $h_\lambda$ finishes with a constant map, so $\gamma_\lambda$ is contractible in $\Sigma_\lambda$. Loop $\gamma_\lambda$ must be simple to project to a simple loop $\gamma$, so it bounds a disk $D$ in $\Sigma_\lambda$. Disk $D$ contains no faces outside of $\Sigma_\lambda^r$, because $\gamma_\lambda$ contains no edges outside of $\Sigma_\lambda^r$ to bound those outside faces. Therefore, $\gamma_\lambda$ bounds a disk ($D$) in $\Sigma_\lambda^r$ implying $\gamma_\lambda$ is contractible in $\Sigma_\lambda^r$. □
Lemma 4.5.8. Let $\Lambda$ be a partial homology basis in $\Sigma$; let $\lambda$ be any cycle in $\Lambda$; and let $\gamma$ be a shortest non-contractible cycle in $\Sigma$. If $\gamma$ is separating, then $\gamma$ lifts to a shortest non-contractible cycle in $\Sigma^*_\lambda$.

4.6 Shortest Non-contractible Cycles in Directed Graphs

We now describe our algorithm for computing a shortest non-contractible cycle. We assume the surface has genus $g \geq 1$. Otherwise, every non-contractible cycle is non-null-homologous, and we can simply use the algorithm given in Section 4.3. Further, we begin by assuming the surface has exactly one boundary cycle. Instances where $\Sigma$ has more than one boundary cycle or no boundary cycles are handled as simple reductions to the one boundary cycle case given at the end of this section.

Let $\partial \Sigma$ denote the one boundary cycle on $\Sigma$. We compute a partial homology basis $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{2g}\}$ in $O(n \log n + gn)$ time as described in Section 4.5. The following lemma states that one of the cycles in the homology basis can be used to build a restricted infinite cyclic cover that is useful for our computation. Surprisingly, the boundary introduced by restricting the infinite cyclic cover plays a key role in the proof of the lemma.

**Lemma 4.6.1.** Let $\gamma$ be a shortest non-contractible cycle in $\Sigma$. If $\gamma$ is separating, then there exists a non-separating cycle $\lambda \in \Lambda$ such that $\gamma$ lifts to a shortest non-null-homologous cycle in the restricted infinite cyclic cover $\Sigma^*_\lambda$.

**Proof:** Every shortest non-contractible cycle is simple [15, Lemma 3]. So by assumption, $\gamma$ is a simple separating cycle. There is exactly one boundary $\partial \Sigma$, so $\gamma$ bounds the closure $A$ of a set of faces. The component $A$ must have genus, or $\gamma$ would bound a disk and be contractible. There exists a simple non-separating cycle $\omega$ on $\Sigma$ contained entirely within $A$. Cycle $\omega$ must cross some other cycle $\lambda \in \Lambda$ an odd number of times [18, Lemma 3]. See Figure 4.3. Consider the infinite cyclic cover $\Sigma^*_{\lambda}$ and its restriction $\Sigma^*_\lambda$.

Let $p$ be a path in $\Sigma$ from $\partial \Sigma$ to $\omega$ such that $p$ does not cross $\lambda$. Path $p$ must exist, because $\lambda$ is non-separating. Further, $p$ crosses $\gamma$ an odd number of times. Let $\partial \Sigma_{\lambda}$ be a lift of $\partial \Sigma$ to $\Sigma_{\lambda}$, and let $p_{\lambda}$ be the lift of $p$ to $\Sigma_{\lambda}$ that begins on $\partial \Sigma_{\lambda}$. Let $\gamma_{\lambda}$ be a lift of $\gamma$ to $\Sigma_{\lambda}$ such that $p_{\lambda}$ crosses $\gamma_{\lambda}$ an odd
number of times. By symmetry and Lemma 4.5.8, we may assume \( \gamma_\lambda \) is a cycle in \( \Sigma_\lambda^r \). We note \( \gamma_\lambda \) is simple as it projects to simple cycle \( \gamma \).

Suppose that \( \gamma_\lambda \) is separating. Let \( \omega_\lambda \) denote a lift of cycle \( \omega \) to \( \Sigma_\lambda \) such that \( p_\lambda \) ends on \( \omega_\lambda \). Curve \( \omega_\lambda \) is not a cycle in \( \Sigma_\lambda \), because \( \omega \) crosses \( \lambda \) an odd number of times in \( \Sigma \) (see Lemma 4.5.4). Therefore, \( \omega_\lambda \) is a simple infinite path that does not cross any lift of \( \gamma_\lambda \). Path \( \omega^r_\lambda = \omega_\lambda \cap \Sigma^r_\lambda \) is a simple arc from \( \lambda^- \) to \( \lambda^+ \) in \( \Sigma^r_\lambda \) which does not cross \( \gamma_\lambda \). Path \( p \) does not cross \( \lambda \), implying that \( p_\lambda \) is a path in \( \Sigma^r_\lambda \) with endpoints on \( \partial \Sigma_\lambda \) and \( \omega^r_\lambda \). Further, \( p_\lambda \) crosses \( \gamma_\lambda \) an odd number of times, implying that \( \gamma_\lambda \) separates \( \partial \Sigma_\lambda \) from \( \omega^r_\lambda \) and \( \lambda^- \).

We see either \( \gamma_\lambda \) is non-separating or it separates a pair of boundary cycles. Therefore, \( \gamma_\lambda \) is non-null-homologous in \( \Sigma^r_\lambda \). Lemma 4.5.8 implies \( \gamma_\lambda \) is actually a shortest non-null-homologous cycle in \( \Sigma^r_\lambda \). \( \square \)

In the above proof, it would actually be preferable if \( \gamma_\lambda \) was separating. In this case, we could find \( \gamma_\lambda \) in \( O(gn \log n) \) time by applying Lemma 4.3.5 along shortest paths between \( \lambda^- \) and each lift of \( \partial \Sigma \). As written, the lemma requires us to apply the full algorithm of Section 4.3 in \( O(g^2n \log n) \) time if we wish to find \( \gamma^r_\lambda \).

We now finish considering the case where \( \Sigma \) has one boundary cycle. Applying Lemmas 4.5.8 and 4.6.1, we construct the restricted infinite cyclic cover \( \Sigma^r_\lambda \) and find a shortest non-null-homologous cycle in \( \Sigma^r_\lambda \) once for each cycle \( \lambda \in \Lambda \) using the algorithm of Section 4.3. This procedure gives us a shortest non-contractible cycle in \( O(g^3n \log n) \) time if any are separating. We apply the algorithm of Section 4.3 (or Erickson’s [47] algorithm) once to \( G \) directly to account for the case where every shortest non-contractible cycle is non-separating. All that remains is to consider the cases where \( \Sigma \) has several boundary cycles or no boundary cycles.

4.6.1 Surfaces with Several Boundary

We now consider the case where \( \Sigma \) has \( b > 1 \) boundary cycles. We apply the algorithm of Section 4.3 to find any shortest non-contractible cycles that are non-null-homologous. Next, we paste disks into all but one of the boundary cycles. This transformation does not introduce any non-contractible cycles, because it does not remove any paths from any homotopies. Further, it does
not restrict the set of non-contractible null-homologous cycles. Every such cycle $\gamma$ still separates a subset of faces (with genus) from the one remaining boundary cycle. We now apply the algorithm as given for one boundary cycle to find any shortest non-contractible cycles that happen to be null-homologous.

4.6.2 Surfaces without Boundary

Finally, we consider the case where $\Sigma$ has no boundary. We apply the algorithm of Section 4.3 to find any shortest non-contractible cycles that are non-null-homologous (we can also apply Erickson’s [47] algorithm as every non-null-homologous cycle is also non-separating on a surface without boundary). We then perform the following reduction in case every shortest non-contractible cycle is null-homologous. We compute one cycle $\lambda$ of a greedy homology basis using a greedy tree-cotree decomposition in $O(n \log n)$ time and reduce the problem of finding the shortest non-contractible cycle for the surface $\Sigma$ with genus $g$ and no boundary to the same problem on the larger surface $\Sigma^r_\lambda$, which has two boundary cycles and genus $5g - 5$. Note that the shortest non-contractible cycle in $\Sigma^r_\lambda$ may be non-separating. The reduction is correct according to Lemma 4.5.8. We then apply the algorithm for several boundary on the new surface $\Sigma^r_\lambda$. Using both extensions and the algorithm as given above, we get our desired theorem.

**Theorem 4.6.2.** A shortest non-contractible cycle in an $n$-vertex directed graph embedded on an orientable surface of genus $g$ with $b$ boundary cycles can be computed in $O((g^3 + gb)n \log n)$ time.

4.7 Conclusions and Open Problems

We gave algorithms to compute shortest non-trivial cycles in both directed and undirected surface embedded graphs. In undirected graphs, our algorithms find shortest non-contractible and non-null-homologous cycles in $2^{O(g+b)}n \log \log n$ time and shortest non-separating cycles in $2^{O(g)}n \log \log n$ time. For directed graphs, our algorithms find shortest non-null-homologous cycles in $O((g^2 + gb)n \log n)$ time and shortest non-contractible cycles in $O((g^3 + gb)n \log n)$ time.
The most obvious question remaining is whether we can reduce these times further. In particular, it is natural to ask if we can compute a shortest non-contractible cycle in a directed surface graph in $O((g^2 + gb)n \log n)$ time, matching the algorithm of Cabello et al. [14] for undirected surface graphs. The main bottleneck appears to be the need to compute shortest non-null-homologous cycles in the restricted infinite cyclic cover. If the proof of Lemma 4.6.1 can be improved to show an appropriate arc or cycle of $\Sigma_\lambda$ is crossed an odd number of times by the lift of a shortest non-contractible cycle, then we can easily reduce the cost of searching each cover to $O(gn \log n)$.

Another question is whether or not the $O(n \log \log n)$ running time achieved by Italiano et al. [77] can be achieved in directed graphs and if its use requires lifting to subsets of the universal cover.
Chapter 5

Counting Minimum $s, t$-cuts

Let $G = (V, E)$ be a weighted directed graph on an orientable surface $\Sigma$ of genus $g$ with two vertices $s, t \in V$. We now consider the problem of counting the minimum $s, t$-cuts of $G$. This problem in general graphs is $\#P$-complete, and can be reduced to the problem of counting maximal antichains in a poset [111]. Ball and Provan [111] first considered the problem of counting minimum cuts and gave an algorithm to compute the number of minimum $s, t$-cuts in an $s, t$-planar graph (where the source and sink are on the same face). Later, Bezáková and Friedlander [4] generalized the algorithm for arbitrary locations of $s$ and $t$ in a planar graph.

Counting the minimum cuts is of interest due to connections with many other areas. For example, the number of minimum cuts is closely related to the probabilistic connectedness of a stochastic graph, where each edge may fail with a certain probability [3], and so it is fundamentally important in several network reliability problems [3, 30, 83, 105].

In addition, cuts have strong connections to problems from computer vision. In image segmentation, the image is represented as a (generally planar) graph on the pixels with edges connecting neighboring pixels weighted according to how similar the pixels are; a minimum cut between two locations corresponds to a good segmentation of the original image [11]. Being able to count minimum cuts is closely related to sampling such cuts [79], implying that our ability to count minimum cuts allows us to sample from the collection of high quality segmentations of an image.

For graphs on surfaces, good segmentation algorithms are key for problems such as texture mapping, metamorphosis, simplification, and compression; see [116] for a recent survey of techniques and applications. Many of these algorithms wish to minimize stretch so that patches of the surface are separated via small separators and local distances within each patch stay close to the original distance. Minimum cuts have strong potential here; by looking
at flow along the mesh, separating along these cuts will again result in a good segmentation. Even an algorithm with large dependence on the genus is useful, because many meshing algorithms attempt to keep the genus small as a way of reducing noise in the mesh.

5.1 New Results

In this chapter, we describe a quadratic time algorithm to compute the number of minimum $s,t$-cuts for a weighted graph embedded on a surface of constant genus. Specifically, our algorithm runs in $2^{O(g)}n^2$ time assuming a cellular embedding is given onto a surface of genus $g$. If no embedding is given, then we can compute one in $2^{O(g)}n$ time [88, 127]. Since counting the number of cuts is generally #P-complete [111], finding a fixed parameter tractable algorithm to compute the number of cuts for a surface embedded graph represents a significant and perhaps optimal improvement in known results for a large family of graphs. Our algorithm requires only a few simple assumptions on the input graph; every edge has positive capacity, and there exists a directed path from $s$ to every vertex in $G$ and a directed path from every vertex in $G$ to $t$.

Our approach uses a connection between cuts and (co-)homology in a non-trivial way, as well as generalizing tools from [4] to more general surfaces. As in [4], our algorithm first reduces the problem of counting minimum cuts in $G$ to the problem of counting forward $t,s$-cuts in a directed acyclic graph. Our algorithm then uses a bijection between forward $t,s$-cuts and circulations of a certain homology class in the dual graph. These reductions are described in Sections 5.3 and 5.4, respectively.

The characterization of cuts using circulations in the dual is not original to the work in this chapter [22,51,108] (see also Chapter 3), but there are some key changes in our characterization and how we use it for counting cuts. In [22,51] and Chapter 3, minimum cuts in undirected graphs are characterized using homology with coefficients in $\mathbb{Z}_2$. However, in our case, $G$ has directed edges, so we must use coefficients in $\mathbb{Z}$. Integer coefficients are used in [108] to compute edge expansion in genus $g$ graphs. However, the algorithm used to compute edge expansion has running time $n^{O(g^2)}$ and is therefore not fixed parameter tractable. To the best of our knowledge, there
is no fixed parameter tractable algorithm that deals with integer homology directly.

In Section 5.5 we describe an algorithm to compute the number of dual circulations in a certain homology class if the primal directed acyclic graph is triangulated. Finally, in Section 5.6 we generalize our algorithm to work for non-triangulated primal graphs as well. Unlike many other problems where triangulating an input graph without changing the output is trivial, we must actually change the surface itself to form a triangulation without affecting the number of forward $t, s$-cuts. Fortunately, some surprising properties of non-crossing cycles allows us to limit the complexity of our modified surface.

Our algorithm can also be used to sample a minimum $s, t$-cut uniformly at random. The sampling algorithm follows almost as an immediate consequence of our main result and the sampling technique given in [4]. After running the counting cut algorithm, we only need $O(n \log n)$ time per sampling, so several samples can be computed quickly. We describe the sampling algorithm in Section 5.7.

**Algorithm Summaries:** As in [4], our counting algorithm first reduces the problem of counting minimum cuts in $G$ to the problem of counting forward $t, s$-cuts in a directed acyclic graph. We show that for directed acyclic graphs embedded on a surface, the edges leaving forward $t, s$-cuts form cycles in the dual graph in a particular homology class. Our algorithm is able to count collections of cycles in this homology class by considering the different ways they cross a set of loops in the primal graph. Counting these collections of cycles only gives the correct answer if the primal DAG is triangulated, so our algorithm adds a small number of handles to the surface first to guarantee the DAG is triangulated without affecting the number of forward $t, s$-cuts.

Our counting algorithm generates a lot of data about the DAG including how many dual paths there are between pairs of dual vertices that cross the surface in different ways. Our sampling algorithm uses this information to randomly pick dual cycles corresponding to forward $t, s$-cuts without having to rerun the entire counting algorithm.
5.2 Directed Graphs and Definitions

As mentioned above, our algorithm begins by reducing the problem of counting minimum $s,t$-cuts to a related problem in a directed acyclic graph. In order to easily take advantage of the structure of a directed acyclic graph, we use a different model of directed edges in this chapter than we do for Chapters 4 and 6. Let $G = (V,E)$ be an arbitrary directed graph embedded on a surface $\Sigma$ of genus $g$. Here, each oriented edge $uv \in E$ coincides with exactly one dart, $u \rightarrow v$. We will use the words edge and dart interchangeably to refer to $u \rightarrow v$. The dual of each edge $u \rightarrow v$ is $f^* \rightarrow g^*$ where $f = left(u \rightarrow v)$ and $g = right(u \rightarrow v)$.

For each edge $e = u \rightarrow v \in E$ we define its tail and head to be the vertices $u$ and $v$, respectively. If $u = v$ then $e$ is a loop. The indegree of a vertex $v$ is the total number of edges in $E$ whose head are $v$. Similarly, the outdegree of $v$ is the total number of edges in $E$ whose tail are $v$. A directed $u,v$-walk in $G$ is a sequence of vertices $W = u = w_1, w_2, \ldots, w_k = v$ such that $w_i \rightarrow w_{i+1} \in E$ for all $1 \leq i < k$; we sometimes denote a $u,v$-walk by $u \leadsto v$; we also use $u \leadsto v$ to designate there is a directed walk from $u$ to $v$. A $u,v$-walk is closed if $u = v$, it is a (directed) path if it has no repeated vertices, and it is a (directed) cycle if it is a path with $u = v$ its only repeated vertex. These definitions of path and cycle differ from the ones used in the other chapters in that paths and cycles here must be simple. An undirected walk, path, or cycle has the same definition as above, except we allow either $w_i \rightarrow w_{i+1} \in E$ or $w_{i+1} \rightarrow w_i \in E$ for any consecutive vertices in the vertex sequence. Note that we still respect the ordering of vertices in an undirected walk even though we are ignoring the orientation of its edges within $G$.

A directed graph $G$ is a directed acyclic graph (DAG) if it does not contain any directed cycle. A vertex $v \in V$ of a DAG is a source if its indegree is zero and a sink if its outdegree is zero.

A spanning tree $\tau$ of a connected graph $G = (V,E)$ is a maximal subgraph of $G$ that contains no undirected cycles. The tree $\tau$ is a forward spanning tree with root $r \in V$ if and only if $\tau$ contains a directed path from $r$ to any vertex $u \in V$; we will also say that $\tau$ is a directed tree with root $r$. Similarly, $\tau$ is a backward spanning tree with root $r$ if it contains a directed path from every vertex $u \in V$ to $r$. 

51
5.3 Minimum Cuts and Forward Cuts

Following Bezáková and Friedlander [4], we begin by reducing the problem of counting minimum cuts to the problem of counting forward cuts. Let $G$ be a directed acyclic (multi-)graph. Let $a$ and $b$ be vertices of $G$. Finally, let $S$ be a subset of vertices such that $a \in S$ and $b \notin S$. We say $S$ is a forward $a,b$-cut of $G$ if there is no edge $u \rightarrow v$ such that $v \in S$ and $u \notin S$. We begin by considering the following theorem:

**Theorem 5.3.1 (Bezáková and Friedlander [4, Theorem 4]).** Let $G = (V, E, c)$ be a (directed) flow network with edge capacities $c : E \rightarrow \mathbb{R}^+$. Let $s \in V$ be the source and $t \in V$ be the sink. There exists a directed acyclic graph (DAG) $\tilde{G} = (\tilde{V}, \tilde{E})$ and vertices $\tilde{s}, \tilde{t} \in \tilde{V}$ such that the number of minimum $s,t$-cuts in $G$ is equal to the number of forward $\tilde{t}, \tilde{s}$-cuts in $\tilde{G}$.

The proof of Theorem 5.3.1 relies upon constructing a maximum flow $f$ for the graph $G$ and removing directed cycles containing flow from $f$. The graph $\tilde{G}$ is then formed by taking the residual graph $G_f$, removing 0 weight edges, and contracting the strongly connected components. The **edge contraction** operation is defined as removing an edge $u \rightarrow v$ of the graph $G$ and identifying its endpoints, $u$ and $v$. Corollary 5.3.2 follows immediately.

**Corollary 5.3.2 (Bezáková-Friedlander [4, Corollary 5]).** Suppose there exits a path from $s$ to every vertex of $G$ and a path from every vertex of $G$ to $t$. Then $\tilde{t}$ is the only vertex of indegree 0 and $\tilde{s}$ is the only vertex of outdegree 0 in $\tilde{G}$.

Bezáková and Friedlander use the fact that contraction of an edge in a planar graph yields a planar graph with an inherited embedding in their algorithm to compute the number of minimum cuts [4].

In our setting, we must similarly exploit the embedding of the graph on a surface. Let $G$ be embedded on a surface $\Sigma$ of genus $g$. The current best running time for computing a maximum flow in an arbitrary sparse graph is $O(n^2 / \log n)$ due to a recent algorithm of Orlin [106]. The proof of Theorem 5.3.1 implies $\tilde{G}$ can be computed in $O(n^2)$ time, because computing a maximum flow, forming the residual graph, and removing cycles from $\tilde{G}$ takes $O(n^2)$ time total in a sparse graph. The proof for Theorem 5.3.1 will not quite get the correct structure for the DAG in our setting, however, since
we will still require the graph to be cellularly embedded on the surface Σ, and contracting strongly connected components may destroy the topology of the underlying surface which is essential to the algorithm.

We therefore modify the original construction as follows. First, our algorithm finds strongly connected components in the residual graph as in the original construction. All of the edges are contracted iteratively unless the edge is a loop; these loops are not contracted. The contraction of a single edge can be done in linear time while maintaining the same embedding on the underlying surface, so the overall contraction algorithm is still $O(n^2)$. Note that the underlying topology is maintained, since a loop will remain in the graph for each handle of the surface. Theorem 5.3.3 follows immediately.

**Theorem 5.3.3.** Let $G = (V, E, c)$ be a (directed) flow network with edge capacities $c : E \to \mathbb{R}^+$ embedded on a surface of genus $g$. Let $s \in V$ be the source and $t \in V$ be the sink. There exists a directed acyclic graph (DAG) $\tilde{G} = (\tilde{V}, \tilde{E})$, possibly with self-loops, embedded on a surface of genus at most $g$ and vertices $\tilde{s}, \tilde{t} \in \tilde{V}$ such that the number of minimum $s, t$-cuts in $G$ is equal to the number of forward $\tilde{t}, \tilde{s}$-cuts in $\tilde{G}$. Moreover, $\tilde{G}$ can be computed in $O(n^2)$ time.

While this graph is not a DAG, our algorithm is resilient to the existence of self loops. As an alternative to the above procedure, our algorithm can perform the original contraction procedure of Bezáková and Friedlander [4] and then compute a new embedding of $\tilde{G}$ without loops onto a new surface of genus at most $g$ [88], but we must consider loops anyway due to technicalities introduced by the procedure in Section 5.6. Note the minimum embedding of a connected graph is known to be cellular [127].

5.4 Forward Cuts and Cocirculations

Based on Theorem 5.3.3 and Corollary 5.3.2, we focus on the problem of counting forward $t, s$-cuts in a directed acyclic graph $G$ possibly with self loops embedded on a surface $\Sigma$ of genus $g$ where $t$ is the only source in $G$ and $s$ is the only sink. Let $\Sigma' = \Sigma \setminus (s^* \cup t^*)$. Simply knowing that $G$ is a DAG immediately gives us the following lemma which generalizes Claim 1 of Bezáková and Friedlander [4]. While the original claim deals with single
cycles in planar graphs, our lemma describes more general circulations. This lemma helps us characterize the edges leaving forward $t, s$-cuts as particular circulations in the dual and makes it possible to count these circulations.

**Lemma 5.4.1.** There exist no non-trivial non-negative boundary circulations of $G^*$ in the surface $\Sigma'$.

**Proof:** For the sake of contradiction, let $\phi$ be a non-trivial non-negative boundary circulation of $G^*$ in the surface $\Sigma'$. We note that no edge dual to a loop in $G$ can have non-zero value in any boundary circulation, since its dual is bordered by the same face on both sides.

Let $(u \rightarrow w)^*$ be a directed edge with $\phi((u \rightarrow w)^*) > 0$. Vertex $u$ is reachable from $t$ since $t$ is the only source in $G$. Likewise, $w$ can reach $s$. Therefore, there exists a simple directed path $p = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ from $t$ to $s$ through $u \rightarrow w$, where $v_0 = t$, $v_k = s$; also for some $0 \leq i < k$, $v_i = u$ and $v_{i+1} = w$.

Boundary circulation $\phi$ is equal to $\partial\alpha$ for some 2-chain $\alpha$ of $G^*$ in the surface $\Sigma$ where $\alpha(v_0^*) = \alpha(v_k^*) = 0$, because $v_0^* = t^*$ and $v_k^* = s^*$ are boundaries in $\Sigma'$. For each edge $v_i \rightarrow v_{i+1}$ of $p$, we have $\alpha(v_{i+1}^*) - \alpha(v_i^*) \leq 0$, because there exists no edge $e$ in $G$ with $\phi(e^*) < 0$. Further, we have $\alpha(w^*) - \alpha(u^*) < 0$. Therefore, either $\alpha(v_0^*) > 0$ or $\alpha(v_k^*) < 0$, a contradiction. 

Theorem 5.3.3 reduces the problem of counting minimum cuts to the problem of counting forward cuts in a DAG that possibly contains self loops. The following results reduce counting forward cuts to the problem of counting circulations in a certain homology class. These results borrow ideas from minimum cut algorithms in surface embedded graphs [22,51] (see also Chapter 3), but require substantially more technical detail to work with integer homology.

**Lemma 5.4.2.** Let $T$ be a forward $t, s$-cut in $G$, and let $\phi_T$ be the 1-chain in $G^*$ where $\phi_T(e) = 1$ if $e^*$ crosses $T$ and $\phi_T(e) = 0$ otherwise. Then, $\phi_T$ is a $(0,1)$-circulation of $G^*$ homologous to $\partial t^*$ in the surface $\Sigma'$.

**Proof:** We define a 2-chain $\alpha$ of $G^*$ in the surface $\Sigma'$. For each vertex $v \in V \setminus \{t, s\}$, let $\alpha(v^*) = 1$ if $v \in T \setminus \{t\}$, and let $\alpha(v^*) = 0$ otherwise. Consider the circulation $\partial\alpha$ and any directed edge $e = u \rightarrow v$ of $G$. If $e$ is a loop so $v = u$, then $\alpha(v^*) - \alpha(u^*) = 0$. If $u = t$ and $v \in T$, then $\partial\alpha(e^*) = -1$. If $u = t$
and \( v \notin T \), then \( \partial \alpha(e^*) = 0 \). If \( u \in T \setminus \{t\} \) and \( v \notin T \), then \( \partial \alpha(e^*) = 1 \). In all other situations, \( \partial \alpha(e^*) = 0 \). We see \( \partial \alpha = \phi_T - \partial t^* \).

**Lemma 5.4.3.** Let \( \phi \) be a non-negative circulation in \( G^* \) that is homologous to \( \partial t^* \) in the surface \( \Sigma' \). Then there exists a forward \( t, s \)-cut \( T \) of \( G \), such that for each edge \( e \in E \), \( e \) crosses \( T \) if and only if \( \phi(e^*) = 1 \). Further, \( \phi \) is a \((0,1)\)-circulation in \( G^* \).

**Proof:** We begin by showing the existence of some (not necessarily forward) \( t, s \)-cut \( T \) where for every edge \( e \) that crosses \( T \) in the forward direction we have \( \phi(e^*) \geq 1 \); i.e., the dual of the collection of edges with value at least 1 in \( \phi \) separates \( t \) from \( s \).

Let \( \phi_t \) be the circulation trivially generated by \( \partial t^* \). By assumption, there exists a 2-chain \( \alpha \) of \( G^* \) in the surface \( \Sigma \) such that \( \partial \alpha = \phi - \phi_t \) and \( \alpha(t^*) = \alpha(s^*) = 0 \). Let \( p = (v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_k) \) be any directed path in \( G \) such that \( v_0 = t \) and \( v_k = s \). Suppose there exists no edge \( e \) of \( p \) with \( \phi(e^*) \geq 1 \). We have \( \alpha(v_i^*) - \alpha(v_{i-1}^*) \geq 1 \) and \( \alpha(v_{i+1}^*) - \alpha(v_i^*) \geq 0 \) for all \( 0 < i < k \). We immediately have a contradiction on \( \alpha(1) = \alpha(v_k^*) = 0 \). Thus, any simple \( t, s \)-path contains an edge \( e \in E \) such that \( \phi(e^*) \geq 1 \). The collection of such edges separates \( t \) from \( s \).

Now, let \( T \subset V \) be the set of vertices reachable from \( t \) using only edges \( e \) with \( \phi(e^*) = 0 \). Recall \( \Gamma^+(T) \) and \( \Gamma^-(T) \) are the sets of edges that cross \( T \) in the forward and backward directions, respectively. Let \( \phi_T \) be the 1-chain of \( G^* \) with \( \phi_T(e^*) = 1 \) for every directed edge \( e \in \Gamma^+(T) \), \( \phi_T(e^*) = -1 \) for every directed edge \( e \in \Gamma^-(T) \), and \( \phi_T(e^*) = 0 \) for every other edge in \( G \). Let \( G' \) be the graph \( G \) with every edge of \( \Gamma^-(T) \) reversed. We see \( T \) is a forward \( t, s \)-cut in \( G' \). By Lemma 5.4.2, \( \phi_T \) is a \((0,1)\)-circulation homologous to \( \partial t^* \) in \( G^* \) on the surface \( \Sigma' \). By assumption, \( \phi \) is homologous to \( \partial t^* \) on \( \Sigma', \) too. Therefore, \( \phi_T \) is also homologous to \( \phi \) on \( \Sigma' \).

Now, let \( \phi' = \phi - \phi_T \). The 1-chain \( \phi' \) is a non-negative boundary circulation on \( G^* \). Thus, Lemma 5.4.1 implies that \( \phi' \) should be trivial. It follows that \( \phi = \phi_T \). Circulation \( \phi_T \) is non-negative, so \( T \) is a forward \( t, s \)-cut. Further, \( \phi \) is a \((0,1)\)-circulation. \( \square \)

Lemmas 5.4.2 and 5.4.3 imply a bijection between forward \( t, s \)-cuts in \( G \) and \((0,1)\)-circulations in \( G^* \) of a particular homology class. We immediately get the following theorem which drives the remaining algorithm design and analysis.
Theorem 5.4.4. Let $G = (V, E)$ be a directed graph embedded on an orientable surface $\Sigma$ with vertices $s, t \in V$ such that there exist no directed cycles in $G$ other than single edge loops and where every vertex of $G$ is reachable from $t$ and every vertex can reach $s$, and let $\Sigma' = \Sigma \setminus (t^* \cup s^*)$. The number of forward $t,s$-cuts in $G$ is equal to the number of $(0,1)$-circulations of $G^*$ homologous to $\partial t^*$ in the surface $\Sigma'$.

5.5 Counting Cuts in Triangulations

In this section, we give our algorithm for counting forward $t,s$-cuts of $G$ assuming $G$ is embedded in $\Sigma$ as a triangulation. We relax our assumption that $G$ is a triangulation in Section 5.6. We first recall that $G^*$ is 3-regular. Therefore, any set of edge-disjoint directed cycles in $G^*$ must also be vertex disjoint. We immediately see every $(0,1)$-circulation $\phi$ of $G^*$ is trivially generated by a unique set of edge-disjoint directed cycles in $G^*$ found by tracing along edges $e$ of $G$ where $\phi(e^*) = 1$. Theorem 5.4.4 then implies we can count the forward $t,s$-cuts of $G$ by counting such collections of cycles in a particular homology class. In fact, the second part of Lemma 5.4.3 implies we can safely count all sets of cycles that trivially generate a circulation in the correct homology class without explicitly checking if they are edge disjoint or simple. We focus on counting these collections of cycles.

Our algorithm begins with the following construction. It creates a tree-cotree decomposition $(\tau, L, C)$ where $\tau$ is an arbitrary directed spanning tree of $G$ rooted at $t$. Euler’s formula implies that $L$ contains exactly $2g$ directed edges which we label $u_1 \rightarrow v_1, \ldots, u_{2g} \rightarrow v_{2g}$. Let $\tau[v]$ denote the directed path from $t$ to $v$ in $\tau$. For each $i \in \{1, \ldots, 2g\}$, let $p_i^+$ denote the directed path $\tau[u_i] \cdot (u_i \rightarrow v_i)$, and let $p_i^- = \tau[v_i]$. Let $p_0^+ = \tau[s]$ and let $p_0^-$ denote the trivial walk from $s$ to itself. Let $p_i$ denote the directed path $p_i^+ \cdot \text{rev}(p_i^-)$. Finally, let $P = \{p_0, p_1, \ldots, p_{2g}\}$. The intersection of $P$ and $\Sigma'$ forms a system of arcs in $\Sigma'$; cutting $\Sigma'$ along $P$ creates a topological disk [20].

Lemma 5.5.1. Let $G$ be a directed acyclic graph with single edge loops, $T$ be a forward $t,s$-cut, and $p$ be a simple directed path in $G$. At most one edge of $p$ crosses $T$.

Proof: Since $T$ is a forward $t,s$-cut, by definition there is no edge going from
V \setminus T to T. Therefore, once any directed walk enters V \setminus T, it cannot cross to T again, giving at most one edge on the walk crossing the cut. □

**Corollary 5.5.2.** At most one edge of each path $p_i^+, p_i^-$ crosses T.

Let $C$ be an arbitrary set of undirected cycles in $G^*$ and let $\phi$ be the circulation trivially generated by $C$. We show how to determine the homology class of $\phi$ in $\Sigma'$ by computing the net number of times the cycles in $C$ cross members of $P$. For any undirected cycle $\gamma = (f_1, f_2, \cdots, f_k)$ in $G^*$ and index $0 \leq i \leq 2g$, let $x_i^+(\gamma)$ be the number of edges $f_j \uparrow f_{j+1}$ in $p_i^+$ minus the number of edges $f_j \uparrow f_{j+1}$ in $p_i^-$. Let $x_i^-\gamma)$ be defined similarly for $p_i^-$. Let $x_i(\gamma) = x_i^+(\gamma) - x_i^-(\gamma)$. Similar to [22, 51], we define the **subdivided crossing vector** $x(\gamma)$ to be $(x_0^+(\gamma), x_0^-(\gamma), \ldots, x_{2g}^+(\gamma), x_{2g}^-(\gamma))$. We define the **arc crossing vector** $x(\gamma)$ to be $(x_0(\gamma), \ldots, x_{2g}(\gamma))$. The subdivided/arc crossing vector of $x(C)$ is the sum of the respective crossing vectors of $C$’s individual elements. Observe that $x(\partial t^*) = (1, 0, 0, \ldots)$. The following lemma and its proof are based on Erickson and Nayyeri [51, Lemma 3.2].

**Lemma 5.5.3.** A set of undirected cycles $C$ in $G^*$ trivially generates a boundary circulation $\phi$ in $\Sigma'$ if and only if $x(C) = 0$.

**Proof:** Suppose $C$ trivially generates boundary circulation $\phi$. By definition, $\phi = \partial \alpha$ for some 2-chain $\alpha$ of $G^*$ in the surface $\Sigma'$. The boundary of any face of $G^*$ has arc crossing vector 0. We see

$$x(C) = x\left(\sum_{v \in V} \alpha(v^*) \cdot \partial v^*\right) = \sum_{v \in V} \alpha(v^*) x(\partial v^*) = 0.$$ 

Now, suppose $x(C) = 0$. We create a graph $G^+$ by modifying $G^*$ as follows. For every arc $p_i \in P$ and for every adjacent pair of edge $e_1, e_2$ in $p_i$, we subdivide $e_1^*$ and $e_2^*$ by replacing $e_1^*$ (respectively $e_2^*$) with a vertex $v e_1^*$ ($v e_2^*$) adjacent to both endpoints of $e_1^*$ ($e_2^*$). We then add an edge $v e_1^* v e_2^*$ to $G^+$, subdividing a face incident to both $e_1^*$ and $e_2^*$. Essentially, we augment $G^*$ with paths that follow the images of loops in $P$. We then prove the lemma by considering the cycles $C$ in $G^+$. Subdividing edges and faces does not change homology.

Let $a$ and $b$ be two intersection (crossing) points between a cycle $\gamma \in C$ and the image of some arc $p_i$ where $\gamma$ crosses $p_i$ from left to right through $a$.
and \( \gamma \) crosses \( p_i \) from right to left through \( b \). If such crossing points do not exist each cycle of \( C \) lies in the disk \( \Sigma' \setminus P \), and the lemma follows. Let \( p_i[a, b] \) be the path added to \( G^+ \) between \( a \) and \( b \) along the image of \( p_i \). Alter \( C \) by replacing \( \gamma \)'s crossings through \( a \) and \( b \) with copies of \( p_i[a, b] \) and \( \text{rev}(p_i[a, b]) \). The transformation does not change the homology class of \( C \) and it reduces the number of crossings of \( P \). By induction, we see \( \phi \) is homologous to a circulation trivially generated from a set of cycles \( C' \) that do not cross \( P \). Each cycle of \( C' \) lies in the disk \( \Sigma' \setminus P \), meaning they trivially generate a boundary circulation. \( \square \)

**Corollary 5.5.4.** Two sets of undirected cycles \( C \) and \( C' \) are homologous if and only if \( x(C) = x(C') \).

Let \( \gamma \) be a directed cycle in \( G^* \). We define the **crossing sequence** of \( \gamma \) to be its cyclic order of crossings of \( \{p_0^-, p_0^+, \ldots, p_{2g}^-, p_{2g}^+\} \). We compute the total number of forward \( t, s \)-cuts in \( G \) (or the total number of sets of cycles in \( G^* \) that trivially generate a circulation homologous to \( \partial t^* \)), by enumerating sets of **abstract** cycles in the dual, where an abstract cycle is specified by a crossing sequence. For any set of abstract cycles \( C_A \), we compute the total number of corresponding circulations in \( G^* \).

We use a method based on previous works [20, 22] (see also Section 4.2 in Chapter 4) to enumerate abstract sets of cycles. Our algorithm cuts \( \Sigma' \) along \( P \) and replaces each copy of \( p_i^+, p_i^- \) with a single edge to obtain an abstract polygonal schema which we denote as \( S \). We emphasize that our algorithm replaces each path \( p_i^+, p_i^- \) by a single edge and **not** each arc \( p_i \) of \( \Sigma' \) as in previous works. Each path \( p_i^+, p_i^- \) corresponds to two edges of \( S \).

Now consider a set of cycles \( C \) that trivially generates a circulation homologous to \( \partial t^* \) in \( \Sigma' \), and let \( C_A \) be its corresponding abstract set of cycles. In polygonal schema \( S \), each of the cycles of \( C_A \) is cut into arcs which cross the schema; by Lemma 5.4.3, the arcs will be non-crossing in the interior of \( S \), and by Corollary 5.5.2 each edge of the schema contains at most one endpoint of any arc; in particular, no two arcs have endpoints on the same pair of boundary.

Our algorithm dualizes the polygonal schema by taking the original abstract schema and replacing each edge with a vertex and each vertex with an edge. It then connects two vertices in the dual if there is an arc between their corresponding edges in the original schema. Now, each arc from \( C_A \)
represents an edge between vertices in the $8g+2$-gon. (Note that we can ignore copies of $p_0^-$ because $p_0^-$ is a trivial walk and no directed paths cross it.) Since none of the arcs can cross, this abstraction gives a subdivision of the dualized schema with no parallel edges. We triangulate this subdivision by adding edges of weight zero. We say that two triangulations of the abstract polygonal schema are *equivalent* if and only if they are identical ignoring zero weight edges. See Figure 5.1 for illustration.

Every equivalence class of weighted triangulations of the dualized schema corresponds to a collection of non-crossing abstract cycles in $G^*$. Our algorithm enumerates such equivalence classes. The weight of each edge is either zero or one, which implies that our algorithm needs to consider only $2^{O(g)}$ different triangulations.

Our algorithm first checks whether a triangulation corresponds to a set of cycles that trivially generate the homology class of $\partial t^*$ using Corollary 5.5.2 and Corollary 5.5.4. We note that each edge of the triangulation has a clear direction specified entirely by the vertices it goes through, because each vertex corresponds to a directed path in $G$. Our algorithm then computes an abstract collection of cycles corresponding to the triangulation by brute force and then computes the *crossing sequence* for each cycle in the collection and $P$. All that remains is to compute the total number of cycle collections in $G^*$ with a given set of crossing sequences.

### 5.5.1 Counting cycles with a given crossing sequence

Let $X$ be an arbitrary crossing sequence corresponding to an abstract directed cycle $\gamma$. We represent the elements of $X$ using the integers $0, \ldots, 2g+1$ along with their negations (where the existence of element $-0$ is optional). Ele-
ment \(i\) represents \(\gamma\) crossing \(p_i^+\) and element \(-i\) represents \(\gamma\) crossing \(p_i^-\). Suppose we perturb the arcs of \(P\) so they are disjoint except at \(t\). Abusing notation, we assign each edge \(e^*\) of \(G^*\) a computed crossing sequence \(X(e^*)\) for the perturbed arcs using the following linear time recursive procedure. For the sake of definition, we add a vertex \(s'\) to \(G\) within an arbitrary face incident to \(s\) along with a directed edge \(s \rightarrow s'\). Set \(X((s \rightarrow s')^*) = 0\). Recall that we use the spanning tree \(\tau\) and the set \(L\) of distinct extra edges in the tree-cotree decomposition to construct \(P\). For each edge \(u_i \rightarrow v_i\) in \(L\), set \(X((u_i \rightarrow v_i)^*) = i\). For each edge \(e^*\) outside \(\tau\) or \(L\) set \(X(e^*) = \varepsilon\). Finally, for each edge \(u \rightarrow v\) in \(\tau\) in bottom up order from the leaves, set \(X((u \rightarrow v)^*)\) using the following iterative subroutine. The crossing sequence \(X((u \rightarrow v)^*)\) begins as the empty sequence \(\varepsilon\). For each edge \(e'\) incident to \(v\) in clockwise order starting with the edge immediately clockwise to \(u \rightarrow v\), append to \(X((u \rightarrow v)^*)\) the crossing sequence \(X(e'^*)\) if \(e'\) has tail \(v\) and the reversal of \(X(e'^*)\) otherwise. Note that \(v\) will be the head of \(e'\) only if \(e' \in L\). As each edge appears at most twice in each arc of \(P\), the above procedure runs in \(O(gn)\) time.

**Lemma 5.5.5.** For any dual cycle \(c\), we have \(X(c)\) equal to the concatenation of crossing sequences for \(c\)'s individual edges in order.

**Proof:** It suffices to prove that for any edge \(e\), the crossing sequence \(X(e^*)\) accurately lists the arcs of \(P\) crossed by \(e^*\) in order. The statement is trivially true for any \(e\) outside \(\tau\) or \(L\). Each edge \(u_i \rightarrow v_i\) in \(L\) appears in exactly one member of \(P\) so the statement is true for those edges as well. Finally, for any edge \(e = u \rightarrow v\) of \(\tau\), assume the computation is accurate for all descendants of \(e\) in the rooted tree \(\tau\). If \(u \rightarrow v\) or \(v \rightarrow u\) appears in any arc \(P\) it must be immediately followed by or proceeded by an edge incident to \(v\). Let \(e'\) be the first edge in the clockwise rotation system of \(v\) following \(e\). Edge \(e'\) is one of \(s \rightarrow s'\), a member of \(L\), a decedent of \(e\) in \(\tau\), or an edge outside of \(L\) and \(\tau\) so we may assume \(X(e')\) is accurate (and possibly empty). If \(X(e'^*)\) is non-empty and \(e'\) has \(v\) as its tail then the arcs passing through \(e'\) must be the leftmost arcs passing through \(u \rightarrow v\) to avoid a crossing and \(e'^*\) passes through them in the same order and direction as \(e^*\). If \(X(e'^*)\) is non-empty and \(e'\) has \(v\) as its head, then the arcs passing through \(e'\) must still be the leftmost arcs passing through \(u \rightarrow v\). However, \(e'^*\) passes through the arcs in the opposite order and direction as \(e^*\). In all cases, we see the concatenation
of $X(e^*)$ or its reversal is correct. The remainder of $X(e^*)$’s computation is correct by induction.

We construct the following graph $G_X$ along with a mapping from vertices and edges of $G_X$ to $G^*$. The vertices of $G_X$ are pairs $(f^*, X')$ where $f^*$ is a vertex of $G^*$ and $X'$ is a prefix of $X$ (including the empty sequence $\varepsilon$). Graph $G_X$ contains edges $(f^*, X') \rightarrow (h^*, X'')$ where $X'$ is a proper prefix of $X$ (not including $X$ itself), $f^* \rightarrow h^*$ is an edge of $G^*$, and $X'' = X' \cdot X(f^* \rightarrow h^*)$. Vertices and edges of $G_X$ map to vertices and edges of $G^*$ by simply dropping the second component of their pairs. We can also define $G_X$ along with an embedding on a disk $\Sigma_X$ using a standard construction [18,95]. Cut along $P$’s image in $\Sigma'$ to create a disk $D$ we call the fundamental domain. Create one copy of $D$ denoted $D_X$ for every prefix $X'$ of $X$. For every pair of prefixes $X'$ and $X''$ where $X'' = X' \cdot i$, paste together $D_X$ and $D_{X''}$ along $p_i^+$. If $X'' = X' \cdot -i$, then paste along $p_i^-$. Finally, remove all outgoing edges from any vertex in $D_X$. Here $\Sigma_X$ is a subset of the universal cover of $\Sigma'$. The next lemma follows from our construction.

**Lemma 5.5.6.** Let $f$ be a face of $G$ on the right side of an edge in path $p_i^+$ ($p_i^-$). If $X$ contains $i$ ($-i$), then there exists a bijection between cycles in $G^*$ with crossing sequence $X$ with first vertex $f^*$ and paths in $G_X$ from $(f^*, \varepsilon)$ to $(f^*, X)$.

Lemmas 5.5.3 and 5.4.1 imply the following lemma.

**Lemma 5.5.7.** Graph $G_X$ is a directed acyclic graph.

A simple dynamic programming algorithm computes the number of paths from a vertex $u$ in a DAG to a vertex $v$ in linear time [4, Observation 6]. For each face $f$ of $G$ on the right side of an edge in path $p_i^+$ ($p_i^-$) where $X$ contains $i$ ($-i$), our algorithm computes the number of paths in $G_X$ from $(f^*, \varepsilon)$ to $(f^*, X)$. It then sums the results.

**Lemma 5.5.8.** Let $X$ be a non-empty crossing sequence of an abstract cycle. Then, there is an $O(|X|n^2)$ time algorithm to compute the number of directed cycles with crossing sequence $X$.

In order to compute the total number of cycle collections in $G^*$ with a given set of crossing sequences, our algorithm simply needs to multiply the number
of cycles for each individual crossing sequence. It then adds the number of cycles corresponding to each equivalence class of weighted triangulations of the dualized polygonal schema. We get the following lemma.

**Lemma 5.5.9.** Let $G$ be a triangulated DAG with possible self-loops embedded on a surface $\Sigma$ of genus $g$ with $t$ and $s$ the only source and sink, respectively. There is a $2^{O(g)}n^2$ time algorithm to compute the total number of forward $t, s$-cuts.

5.6 Handling Non-triangulations

In this section, we remove our assumption that $G$ is embedded in $\Sigma$ as a triangulation. We sketch an algorithm to build a triangulated graph $G_\Delta$ embedded on a surface $\Sigma_\Delta$ of genus $O(g)$ with the same total number of forward $t, s$-cuts. We can then use the algorithm of Section 5.5 to count the forward cuts in $G_\Delta$ and so in $G$. The following lemma has a key role in constructing $G_\Delta$.

**Lemma 5.6.1.** Let $G = (V, E)$ be a DAG with possible self-loops and $t$ and $s$ the only source and sink vertices, respectively. Assume $u, v \in V$, $u \not\rightarrow v \notin E$ and $u \not\leftrightarrow v$. Then, $G \cup u \rightarrow v$ is a DAG with possible self-loops that has the same number of forward $t, s$-cuts as $G$.

**Proof:** If $u = v$ then the lemma is trivial, because a self-loop never shows up in a forward cut. We may assume $u \neq v$.

Assume $G \cup u \rightarrow v$ has a directed cycle $\gamma$ of length at least two. Since $G$ does not contain any directed cycles, we know $u \rightarrow v \in \gamma$. It immediately follows that $[\gamma \setminus u \rightarrow v] \cup u \not\leftrightarrow v$ contains a closed walk with at least two distinct vertices in $G$, which contradicts the lemma assumption.

Now consider any forward $t, s$-cut $T$ in $G$. Since $u \not\leftrightarrow v$ it cannot be the case that $u \in S$ and $v \in T$; it follows that $T$ is a forward cut in $G \cup u \rightarrow v$ as well.

On the other hand, a forward cut in $G \cup u \rightarrow v$ is indeed a forward cut in its subgraph $G$, and the proof is complete.

A face $f$ of $G$ is **irreducible** if and only if for any two vertices $u, v \in V$ that are not adjacent on $f$ (1) $u \neq v$, (2) $u \not\leftrightarrow v$, and (3) $v \not\leftrightarrow u$. In particular,
Figure 5.2: Irreducible faces of degree four, six and eight; circles: outgoing vertices; bullets: incoming vertices.

the boundary of an irreducible face is composed of an even number of edges with alternating clockwise and counterclockwise directions; see Figure 5.2.

Lemma 5.6.2. Let \( f \) be an irreducible face. Then, all vertices on the boundary of \( f \) are distinct.

Proof: Any two non-adjacent vertices on the boundary of \( f \) are distinct by the definition of an irreducible face.

Assume that \( u \rightarrow v \) appears on the boundary of \( f \) and that \( u = v \). Let \( w \) be the other neighbor of \( v \) on the boundary of \( f \), and so \( w \rightarrow v \in E \). Because \( u \) and \( v \) are identical, it follows that \( w \rightarrow u \in E \), in particular \( w \sim u \). Since the degree of an irreducible face is at least 4, \( w \) and \( u \) cannot be adjacent on the boundary of \( f \), which implies that \( f \) is not irreducible. \( \Box \)

An embedded DAG with possible self loops is maximally triangulated if and only if any non-triangle face of it is irreducible. Applying Lemma 5.6.1 and adding self-loops let us create a maximally triangulated graph \( G_\delta \).

Lemma 5.6.3. Let \( G \) be a DAG with possible self-loops embedded on a surface \( \Sigma \) such that \( t \) and \( s \) are the only source and sink. Then, there is an \( O(n^2) \) time algorithm to compute a maximally triangulated DAG \( G_\delta \) embedded on \( \Sigma \) that has the same number of forward \( t,s \)-cuts as \( G \).

Proof: Lemma 5.6.1 implies that for any pair of vertices \( u, v \in V \) such that \( u \not\sim v \), we can add \( u \rightarrow v \) without changing the total number of forward \( t,s \)-cuts. In particular, we can also add self-loops without changing the total number of forward \( t,s \)-cuts.

First, in \( O(n^2) \) time, for all pairs of vertices \( u, v \in V \), our algorithm figures out whether \( u \not\sim v \) by running breadth first searches from all vertices of \( G \). Then, for each vertex \( u \in V \), we add all \( u \rightarrow v \) edges such that \( u \) and \( v \) are on a same face and \( u \not\sim v \). It is straightforward to check that the resulting graph \( G_\delta \) is maximally triangulated. \( \Box \)
Unfortunately, this process does not necessarily result in a triangulation. We cannot add edges on an irreducible face without possibly changing the number of forward $t,s$-cuts. Fortunately, we can prove that the total number of irreducible faces is $O(g)$ in any maximally triangulated DAG.

Let $v$ be a vertex on the boundary of an irreducible face $f$ of the maximally triangulated graph $G_δ$. Then, $v$ is *incoming* on $f$ if and only if both incident edges to $v$ on $f$ are incoming. Similarly, $v$ is *outgoing* on $f$ if and only if both incident edges to $v$ on $f$ are outgoing. Observe that any vertex $v$ on the boundary of any irreducible face $f$ is either incoming or outgoing on $f$; see Figure 5.2.

Let $S$ be a backwards spanning tree of $G$ with root $s$. Let $S[v]$ denote the directed path from vertex $v \in V$ to $s$.

**Figure 5.3:** The setting for Lemma 5.6.4.

**Lemma 5.6.4.** Let $f$ be an irreducible face and $u$ and $v$ be outgoing and incoming vertices on $f$, respectively. Then, there is no directed path from any vertex of $S[v]$ to $u$; in particular, no directed $t,u$-path intersects $S[v]$.

**Proof:** Let $\gamma = S[v]$ and assume, for the purpose of contradiction, that there exists a directed path $\tau$ from a vertex $x \in \gamma$ to $u$. It follows that there exists a directed path, $\gamma[v,x] \cdot \tau[x,u]$, from $v$ to $u$. Since $G$ does not contain a directed cycle whose length is larger than 1, we have $u \rightarrow v \notin E$, which implies that $f$ is reducible; see Figure 5.3. \[\square\]

Let $u$ be an outgoing vertex on an irreducible face $f$ and $v_1$ and $v_2$ be $u$'s neighbors on $f$. We define $C(S,f,u)$ to be the undirected cycle that is composed of $u \rightarrow v_1 \cdot S[v_1]$ and $u \rightarrow v_2 \cdot S[v_2]$; see Figure 5.4, left.

**Lemma 5.6.5.** Let $f$ be an irreducible face and $u$ be an outgoing vertex on $f$. Then, $C(S,f,u)$ is a non-separating cycle on $\Sigma$.  

64
Figure 5.4: Left: The definition of $C(S, f, u)$. Right: The proof of Lemma 5.6.5.

Proof: Assume, for the purpose of contradiction, that $C(S, f, u)$ is separating. Let $v_1$ and $v_2$ be the neighbors of $u$ on $f$ and $\alpha$ be a $v_1, v_2$-path (in the surface and not in the graph), which is strictly inside $f$ except for its endpoints that are on the boundary of $f$. Also, let $\gamma_1 = S[v_1]$ and $\gamma_2 = S[v_2]$.

(See Figure 5.4, right.)

Since $C(S, f, u)$ and $\alpha \cup uv_1 \cup uv_2$ are separating, $\gamma = \alpha \cup \gamma_1 \cup \gamma_2$ is separating as well. Cycle $\gamma$ separates $\Sigma$ into two surfaces $\Sigma_u$ and $\Sigma_w$. Let $w$ be the other neighbor of $v_1$ on $f$ and observe that $u$ and $w$ are on two different sides of $\gamma$. Without loss of generality assume that $u \in \Sigma_u$ and $w \in \Sigma_w$.

Since $t$ is the source of the DAG and its indegree is zero it cannot be on $\gamma$, so it is either in $\Sigma_u$ or in $\Sigma_w$. In the former case any $t, w$-path has to intersect $\gamma$ and in the latter case any $t, u$-path has to intersect $\gamma$. In either case there is a directed path from $\gamma_1 = S[v_1]$ or $\gamma_2 = S[v_2]$ to $u$ or $w$, which contradicts Lemma 5.6.4.

Lemma 5.6.6. Let $f$ and $f'$ be irreducible faces and $u$ and $u'$ be two outgoing vertices on $f$ and $f'$, respectively, where $f \neq f'$. Then, $C(S, f, u) \cup C(S, f', u')$ does not separate $\Sigma$; in particular, $C(S, f, u)$ and $C(S, f', u')$ are not homotopic.

Proof: Let $v_1$ and $v_2$ be the neighbors of $u$ on $f$ and $\alpha$ be a directed $v_1, v_2$-path (in the surface and not in the graph), which is strictly inside $f$ except for its endpoints that are on the boundary of $f$. Similarly, let $v'_1$ and $v'_2$ be the neighbors of $u'$ on $f'$ and $\alpha'$ be a directed $v'_1, v'_2$-path, which is strictly inside $f'$ except for its endpoints that are on the boundary of $f'$. Let $w, w' \in V$ be the other neighbors of $v_1$ and $v'_1$ on $f$ and $f'$, respectively. Further, let $\gamma_1 = S[v_1]$, $\gamma_2 = S[v_2]$, $\gamma'_1 = S[v'_1]$, $\gamma'_2 = S[v'_2]$, $\gamma = \alpha \cup \gamma_1 \cup \gamma_2$ and $\gamma' = \alpha' \cup \gamma'_1 \cup \gamma'_2$. 
Assume, for the purpose of contradiction, that $C(S, f, u) \cup C(S, f', u')$ is separating. Since $\alpha \cup uv_1 \cup uv_2$ and $\alpha' \cup u'v_1' \cup u'v_2'$ are contractible cycles and $C(S, f, u) \cup C(S, f', u')$ is separating, $\gamma \cup \gamma'$ is also separating. Subspace $\gamma \cup \gamma'$ separates $\Sigma$ into two surfaces $\Sigma_u$ and $\Sigma_w$. Observe that $u$ and $w$ are on two different sides of $\gamma \cup \gamma'$. Without loss of generality assume that $u \in \Sigma_u$ and $w \in \Sigma_w$.

There are four possible cases to consider depending on whether $t \in \Sigma_u$ and whether $u' \in \Sigma_u$.

First we suppose that $t \in \Sigma_u$ and $u' \in \Sigma_u$. In this case, observe that $w' \in \Sigma_w$. If we assume also that $t \in \Sigma_u$, any $t, w'$-directed path $\tau'$ intersects $\gamma \cup \gamma'$. Lemma 5.6.4 implies that $\tau'$ cannot intersect $\gamma'$. So, it intersects $\gamma_1$ or $\gamma_2$. Without loss of generality assume that $\tau'$ intersects $\gamma_1$, and $x'$ is any vertex in the intersection. Then, $\delta = \gamma_1[v_1, x'] \cdot \tau'[x', w'] \cdot w' \rightarrow v_1'$ is a directed $v_1v_1'$ path. See Figure 5.5.

Similar to above, any directed $t, w$-path, $\tau$, intersects $\gamma_1'$ or $\gamma_2'$.

**Subcase 1:** If there is a vertex $x \in \tau \cap \gamma_1'$, then there is a $v_1', v_1$-directed path $\delta' = \gamma_1'[v_1', x] \cdot \tau[x, w] \cdot w \rightarrow v_1$, and $\delta \cdot \delta'$ is a closed walk with at least two distinct vertices contradicting the assumption that $G$ is a DAG.

**Subcase 2:** If there is a vertex $x \in \tau \cap \gamma_2'$, then there is a $v_2', v_1$-directed path $\delta' = \gamma_2'[v_2', x] \cdot \tau[x, w] \cdot w \rightarrow v_1$, and $\delta' \cdot \delta$ contains a $v_2', w'$-directed path contradicting either the assumption that $G$ has no cycle of length larger than one (if edge $w' \rightarrow v_2'$ exists) or the assumption that $f'$ is irreducible (otherwise).

Now if $t \in \Sigma_u$ and $u' \notin \Sigma_u$ (so $u' \in \Sigma_w$), we can instead consider a $t, w$-directed path $\tau'$ intersecting $\gamma \cup \gamma'$ and $\gamma_1 \cup \gamma_1'$, and the argument proceeds analogously.
directed path \( \tau \). Similarly to the last case, we know \( \tau \) must intersect \( \gamma \cup \gamma' \), giving an intersection with either \( \gamma'_1 \) or \( \gamma'_2 \) from Lemma 5.6.4. From this point, the argument is identical to the above case.

The cases when \( t \in \Sigma_w \) are completely symmetric, with the same intersection patterns occurring as above.

Finally, we note that if \( u = u' \), we are simply in the case where \( u' \in \Sigma_u \). If \( v_1, v_2, v_3, \) and \( v_4 \) are distinct, the above cases work entirely. If not (so that \( f \) and \( f' \) share vertices), then we simply have for example \( v_1 = v'_1 \), and know that \( \tau \) cannot cross \( \gamma_1 \) or \( \gamma'_1 \), leaving the forced intersection from above to be with \( \gamma'_2 \). Finally, if we have \( v_1 = v'_1 \) and \( v_2 = v'_2 \), then any such \( \tau \) could not cross either \( \gamma'_1 \) or \( \gamma'_2 \) by Lemma 5.6.4, and we have an immediate contradiction to the assumption of \( C(S, f, u) \cup C(S, f', u') \) separating the surface.

\[ \square \]

Using Lemmas 5.6.5 and 5.6.6 we obtain an upper bound on the number of irreducible faces.

**Lemma 5.6.7.** There are at most \( 6g \) irreducible faces in \( G \).

**Proof:** Consider a backward spanning tree \( S \) rooted at \( s \). Let \( u_1, \ldots, u_k \) be a list of outgoing vertices (with possible multiplicity) with one vertex in each irreducible face of \( G \); for \( 1 \leq i \leq k \) assume \( u_i \) is on the irreducible face \( f_i \). By construction, the closed walks \( C(S, f_i, u_i) \) (\( 1 \leq i \leq k \)) are mutually non-crossing. Lemma 5.6.5 implies that for any \( 1 \leq i \leq k \), \( C(S, f_i, u_i) \) is non-separating, and Lemma 5.6.6 implies that for any pair \( 1 \leq i < j \leq k \), \( C(S, f_i, u_i) \) and \( C(S, f_j, u_j) \) are non-homotopic. It follows that \( k \leq 6g \); see Chambers et al. [20, Lemma 2.1].

\[ \square \]

To get rid of irreducible faces we further triangulate \( G_\delta \) by connecting the vertices that appear on the boundary of irreducible faces to \( s \); Lemma 5.6.1 implies that we can always add edges to \( s \) without changing the total number of forward cuts. However, adding edges with endpoints in different faces results in changing the underlying surface \( \Sigma \); intuitively, we need to glue more handles to the surface to avoid edge crossings. The following lemma shows that all irreducible faces can be triangulated by adding only \( O(g) \) handles to \( \Sigma \).
Lemma 5.6.8. Let $G_\delta$ be a maximally triangulated DAG with possible self-loops embedded on a surface $\Sigma$ of genus $g$, and $t$ and $s$ be the only source and sink, respectively. Then, there exists a triangulated supergraph $G_\Delta$ of $G_\delta$ embedded on a surface $\Sigma_\Delta$ of genus $O(g)$ such that the number of forward $t,s$-cuts in $G_\delta$ and $G_\Delta$ are equal. Further, $G_\Delta$ can be computed in $O(n)$ time.

Proof: Lemma 5.6.3 implies $G_\delta$ has $O(g)$ irreducible faces. Lemma 5.6.1 implies $s$ is not on the boundary of any irreducible face.

Let $f$ be an irreducible face and $f'$ be a triangle incident to $s$. Let the vertices on the boundary of $f$ and $f'$ be $(v_0, v_1, \ldots, v_{k-1})$ and $(s, s', s'')$ respectively, in clockwise order. We add a handle to connect $f$ and $f'$, and use it to add edges from all $v_i$'s to $s$. Combinatorially, for all $0 \leq i < k$, we add edge $e_i = v_i \rightarrow s$ such that (1) for all $0 \leq i < k$, $e_i$ is between $v_i \oplus 1 v_i$ and $v_i \ominus 1 v_i$ in the clockwise rotation system edge list of $v_i$, where $\oplus$ and $\ominus$ are addition and subtraction modulo $k$, (2) for all $1 \leq i < k - 1$, $e_i$ is between $v_i \ominus 1 s$ and $v_i \oplus 1 s$ in the list of $s$, (3) $v_0 s$ is between $s'' s$ and $v_1 s$ in the list of $s$, and (4) $v_{k-1} s$ is between $v_{k-2} s$ and $s' s$ in the list of $s$; see Figure 5.6.

It is easy to check that for any $0 \leq i < k - 1$, the triangle $(v_i, s, v_{i+1})$ is a face of the new graph. The only face that is not a triangle is $(v_0, s, s'', s', s, v_{k-1})$, which can be triangulated by adding the following edges: $s'' \rightarrow s$, $s \rightarrow s$ and $v_{k-1} \rightarrow s$; see Figure 5.6. Since all new edges are towards $s$, Lemma 5.6.1 implies that adding them does not change the number of forward $t,s$-cuts.

Each irreducible face of degree $d$ can be triangulated by adding one handle in $O(d)$ time. It follows that we can iteratively triangulate $G_\delta$ to obtain $G_\Delta$ by adding $O(g)$ handles in $O(n)$ time.

□

Theorem 5.3.3 reduces the problem of counting minimum cuts in a surface embedded graph of genus $g$ to the problem of counting forward cuts in a graph embedded on the same surface. Lemmas 5.6.3 and 5.6.8 reduce the latter problem to counting forward cuts in a triangulation of a surface of genus $O(g)$. Finally, Lemma 5.5.9 provides an algorithm to count forward cuts in embedded triangulations. Thus, we derive our main theorem.

Theorem 5.6.9. Let $G = (V, E, c)$ be a (directed) flow network with edge capacities $c : E \rightarrow \mathbb{R}^+$ embedded on an orientable surface $\Sigma$ of genus $g$. Let $s \in V$ be the source and $t \in V$ be the sink where there exists a path from $s$ to every vertex in $V$ and a path from every vertex in $V$ to $t$. There
exists a $2^{O(g)}n^2$ time algorithm to calculate the number of minimum $s,t$-cuts in $G$. 

5.7 Sampling Minimum Cuts

In this section, we given an algorithm to sample a minimum $s,t$-cut from a graph uniformly at random. Let $G = (V,E)$ be a directed acyclic graph plus a set of loops embedded on a surface $\Sigma$ of genus $g$ with a unique source $t$ and unique sink $s$. Let $G^*$ be the dual graph of $G$ and let $\Sigma' = \Sigma \setminus (t^* \cup s^*)$. By Theorem 5.3.3, it suffices to give an algorithm to sample forward $t,s$-cuts in $G$. Our sampling algorithm combines the ideas from earlier in this paper with the algorithm given in [4]. We assume $G$ is embedded as a triangulation without loss of generality (see Section 5.6) and run the counting algorithm given in Section 5.5. We assume familiarity with the counting algorithm as given.

Our counting algorithm enumerates weighted triangulations of a dualized polygonal schema with a particular arc crossing signature relative to a system of $2g+1$ arcs. For each such triangulation, it counts the directed cycles in $G^*$ that correspond to the crossing sequences represented in the triangulation. See Section 5.5 for details. For each such triangulation $\Delta_i$, let $c_i$ be the number of collections of directed cycles corresponding to $\Delta_i$. Our sampling algorithm samples a single weighted triangulation where each triangulation $\Delta_i$ is picked with probability $c_i / \sum_k c_k$. 

69
Let $\Delta$ be the triangulation sampled. Our *counting* algorithm computes an abstract collection of cycles corresponding to $\Delta$. For each cycle in the abstract collection, it computes the number of real cycles with the same *crossing sequence* relative to a system of $4g+2$ paths. Our sampling algorithm picks a cycle uniformly at random for *each* of these crossing sequences. Let $X$ be one such crossing sequence.

Given $X$, our counting algorithm creates a directed acyclic graph $G_X$. It then counts the directed paths in $G_X$ between several pairs of endpoints. For each pair of endpoints $((f^*_{i}, \varepsilon), (f^*_{i}, X))$, let $d_i$ be the number of directed paths between $(f^*_{i}, \varepsilon)$ and $(f^*_{i}, X)$ in $G_X$. Our sampling algorithm picks a pair of endpoints where each pair $((f^*_{i}, \varepsilon), (f^*_{i}, X))$ is picked with probability $d_i / \sum_k d_k$.

Finally, we describe how to sample a directed path between a pair of endpoints $(f^*, \varepsilon)$ and $(f^*, X)$. Let $x_0 = (f^*, X)$. For every $k = 1, 2, \ldots$, our sampling algorithm selects $x_k$ from the set of immediate predecessors to $x_{k-1}$ with probability proportional to the number of paths between $(f^*, \varepsilon)$ and the predecessor. The reverse of $x_0, x_1, \ldots$ gives us a randomly sampled path in $G_X$ or equivalently a randomly sampled cycle in $G^*$.

All of the information required for the sampling algorithm is computed by the counting algorithm. When sampling, a random base vertex for each dual loop is chosen (without explicitly building the subset of the universal cover); this takes $O(\log n)$ time since we are sampling among $n$ vertices. As we walk backwards along a random directed path, it takes at most $O(\log n)$ time to pick $x_k$ from the set of predecessors, where lookups are done in the table for the dynamic programming. Since there are at most $n$ vertices on the directed path, the total time is at most $O(n \log n)$. (Note that if one builds the universal cover for the weighted triangulation explicitly rather than storing the information in a dynamic programming table, it results in an extra factor of $g$, giving $O(gn \log n)$ instead.)

Finally, we get the following result:

**Theorem 5.7.1.** Let $G = (V, E, c)$ be a (directed) flow network with edge capacities $c : E \rightarrow \mathbb{R}^+$ embedded on an orientable surface $\Sigma$ of genus $g$. Let $s \in V$ be the source and $t \in V$ be the sink where there exists a path from $s$ to every vertex in $V$ and a path from every vertex in $V$ to $t$. There exists an algorithm to sample minimum $s$, $t$-cuts uniformly at random in $O(n \log n)$
time per sample after running our algorithm to count minimum $s, t$-cuts in $G$ once.
In this chapter, we describe ongoing work in the computation of maximum flows in surface embedded graphs. Let $G = (V, E)$ be a positively capacitated directed graph embedded on a surface $\Sigma$ of genus $g$ and no boundary. Let $s, t \in V$ be the designated source and sink vertices. Our goal is compute a maximum $s, t$-flow in $G$.

The fastest algorithms for computing maximum flows in general graphs are due to Orlin [106] and King, Rao, and Tarjan [92]. Together their algorithms can be used to solve the maximum flow problem in $O(nm)$ time if $G$ has $m$ edges. For sparse $G$, Orlin [106] describes an algorithm to solve the problem in only $O(n^2 / \log n)$ time. If $G$ has integer capacities, then an algorithm of Goldberg and Rao [63] can compute the maximum $s, t$-flow in $O(n^{3/2} \log n \log U)$ time.

Ford and Fulkerson gave the first maximum flow algorithm that considers planarity in their seminal paper [57] on maximum flows and minimum cuts, although their algorithm assumes $s$ and $t$ lie on the same face (the graph is $s,t$-planar). Itai and Shiloach [76] consider $s, t$-planar graphs as well, and showed how to compute the maximum flow in $O(n \log n)$ time if the graph is undirected. Their result was generalized to directed graphs by Hassin [70], whose algorithm can be made to run in linear time by applying a linear time shortest path algorithm of Henzinger et al. [73] as a subroutine.

When $G$ is planar but $s$ and $t$ lie on different faces, the problem becomes more difficult. Itai and Shiloach [76] describe how to compute a minimum cut in $O(n^2 \log n)$ time assuming $G$ is undirected by computing a shortest cycles separating $s^*$ from $t^*$ in the dual graph. Reif [112] improved the running time to $O(n \log^2 n)$ by employing a divide-and-conquer strategy for finding the shortest separating cocycle. Reif’s algorithm was further improved by Frederickson [59] and eventually Italiano et al. [77] who gave an $O(n \log \log n)$ time algorithm for the problem. All of these algorithm can also be used to
compute a maximum $s,t$-flow without an asymptotic increase in running time by using a procedure of Hassin and Johnson [71].

For directed $G$, the problem has a shorter history. Johnson and Venkatesan [80] described an $O(n^{3/2} \log n)$ time algorithm that uses a divide-and-conquer strategy through recursive separator decompositions. Venkatesan [125] observed that a feasible flow of value $\lambda$ could be computed in $O(n^{3/2})$ time if one exists by computing a single-source shortest path tree in the dual residual graph of any flow with value $\lambda$. Here, the lengths of the darts used in computing the shortest path tree are equal to darts’ capacities. The feasible flow exists if and only if the dual residual graph contains no negative length cycles. This result implies a binary search strategy to compute a maximum $s,t$-flow assuming the darts have integer capacities. Using the recent shortest path algorithm of Mozes and Wulff-Nilsen [103], this strategy runs in $O(n \log^2 n \log C / \log \log n)$ time, where $C$ is the sum of the dart capacities. Weihe [126] described an algorithm to compute maximum $s,t$-flows in directed planar graphs with arbitrary real dart capacities in $O(n \log n)$ time assuming a strong connectivity assumption about $G$. Borradaile and Klein [9] described a different algorithm that removes the connectivity assumption while still running in $O(n \log n)$ time.

The observations of Itai and Shiloach [76] have been generalized to undirected surface embedded graphs for the purpose of computing minimum $s,t$-cuts [22,51,77] and global minimum cuts as described in Chapter 3. The authors of those works interpret finding minimum length cocycles in the plane as finding minimum weight members of certain $\mathbb{Z}_2$ homology classes. This interpretation yields algorithms that work with arbitrary real edge capacities and run in as little as $g^{O(g)} n \log \log n$ time in higher genus surfaces.

However, little is known about computing maximum flows in undirected or directed surface embedded graphs. The only results that are better than those already known for more general graphs are two algorithms of Chambers, Erickson, and Nayyeri [23]. These algorithms respectively run in $O(g^8 n \log^2 n \log^2 C)$ time if $G$ has integer capacities summing to $C$ and $g^{O(g)} n^{3/2}$ time for arbitrary real capacities. Both of these algorithms use a generalization of Venkatesan’s [80] observations noted above. Namely, there exists a feasible $s,t$-flow in $G$ homologous to a given flow $f$ if and only if the dual residual graph $G_f^*$ contains no negative length cycles. The two algorithms both rely on solving an implicit linear programming problem using
a shortest path solver as a separation oracle. The algorithm for integer dart capacities uses the central-cut ellipsoid method [67, 68, 90, 91, 117, 130, 131] while the other uses a variant of multidimensional parametric search [1].

Both of these methods for solving the linear program are extremely complicated, and the authors admit that existing algorithms for more general classes of graphs should outperform the two algorithms above. It appears there is more work to be done, and they conjecture that there exists an $O(g^k n \log n)$ time algorithm for computing maximum $s,t$-flows where $k$ is some small constant.

We note that in contrast to the results on minimum cuts, neither algorithm for maximum flows on surfaces generalizes techniques for the best known results for maximum flows in planar graphs. A natural plan of attack is to find an interpretation of Borradaile and Klein’s [9] algorithm that works on surface with genus. Erickson [46] tried this strategy with his reinterpretation of Borradaile and Klein’s algorithm. Recall that there exists a maximum $s,t$-flow of value $\lambda$ in the plane if and only if shortest path trees exist in the dual residual graph of any $s,t$-flow of value $\lambda$. Erickson describes computing a maximum $s,t$-flow as a parametric shortest path problem where the algorithm continuously pushes increasing amounts of flow along a single $s,t$-path while maintaining a shortest path tree of the dual residual graph. He argues that individual changes to the shortest path tree can be computed in $O(\log n)$ amortized time each given the correct data structures. Further, the shortest path tree changes at most $O(n)$ times before it is no longer well defined and the flow from $s$ to $t$ is maximized. However, it is not clear how this interpretation can lead to an efficient maximum flow algorithm for graphs on higher genus surfaces. There exist graphs even in the torus where pushing flow along certain primal $s,t$-paths can cause $\Omega(n^2)$ changes to a shortest path tree in the dual residual graph [46]. Also, a homology class for a maximum $s,t$-flow may never be reached if flow is only augmented along a single $s,t$-path. The dual residual graph of a flow $f$ may contain negative length cycles even though there exist feasible flows of higher value than $f$ [23].
6.1 New Results

We describe a new algorithm to compute a maximum $s,t$-flow in $G$. This algorithm is considerably simpler than the algorithms of Chambers et al. [23] and relies only on data structures that are now standard when working with flow and cuts in planar and surface embedded graphs. The algorithm is based on a new interpretation of Venkatesan’s [80] observations on dual shortest path trees and a new interpretation of Borradaile and Klein’s [9] algorithm for maximum flow in planar graphs. The high level idea behind our algorithm is that instead of asking whether a shortest path tree exists in the dual residual of certain $s,t$-flows, we ask whether a minimum capacity coflow for a particular cohomology class exists. We compute a minimum capacity coflow for a particular cohomology class, and then update the coflow as primal flow is continuously pushed along arbitrary $s,t$-paths. Only the value of the primal flow has an effect on the minimum capacity coflow, so we can choose $s,t$-paths convenient for our algorithm design and analysis. The cohomology class is chosen so that these updates can be performed in $O(g \log n + g^2)$ amortized time each.

We feel this is in some sense the “correct” interpretation Borradaile and Klein’s [9] algorithm as the high level details of the algorithm for higher genus surfaces derive naturally from the high level premise sketched above. In fact, other than some additional bookkeeping details, our algorithm is actually identical to Borradaile and Klein’s [9] when the input graph $G$ is embedded in the plane.

We are able to show that our algorithm will always terminate with a maximum $s,t$-flow after a finite number of operations, even when the capacities of $G$ are arbitrary real numbers. Unfortunately, we are unable to prove that our algorithm has better than a near-quadratic running time when the genus is fixed. We still believe that the hoped for $O(g^k n \log n)$ running time might hold for our algorithm, as proving the $O(n \log n)$ bound is highly non-trivial even for Borradaile and Klein’s [9] algorithm. To our knowledge, our algorithm is the only generalization of Borradaile and Klein’s that takes advantage of surface topology while also being guaranteed to terminate with a maximum flow.

**Algorithm Summary:** As stated above, our algorithm essentially main-
tains a minimum capacity coflow in the dual residual graph as primal flow is continuously pushed from \( s \) to \( t \). In addition, our algorithm uses a consequence of linear programming duality to maintain the property that the \( s,t \)-flow is also the maximum cost feasible flow for its value where the cost function for the flow is precisely the minimum capacity coflow the algorithm maintains. The changes to the flow and coflow take place over several iterations. In each iteration, the flow is updated to have as high a value as possible without changing the coflow. Then, the coflow is updated so that more flow can be pushed from \( s \) to \( t \). Our algorithm uses the grove data structure of Cabello, Chambers, and Erickson [14] to perform these iterations quickly. The algorithm terminates when the coflow can no longer be updated so that more flow can be pushed from \( s \) to \( t \).

6.2 Assumptions and Data Structures

We assume without loss of generality that \( s \) and \( t \) have degree 1. To enforce this assumption, we add new vertices \( s' \) and \( t' \) to \( G \), connecting them only to \( s \) and \( t \) via edges with very large capacity for both darts. We make the new vertices \( s' \) and \( t' \) the new source and sink respectively in \( G \).

We now describe the data structures used to efficiently implement our algorithm for maximum flows. The first of these is a dynamic forest data structure. For each edge \( uv \) in the forest, the data structure maintains two separate real values \( \text{val}(u \rightarrow v) \) and \( \text{val}(v \rightarrow u) \) for \( uv \)'s darts. Sentinel value \( \infty \) is also a possible value for each dart. The data structure supports the following operations:

- **CREATE(\( u \))**: Return a new tree with vertex \( u \).
- **CUT(\( uv \))**: Remove edge \( uv \) from the forest.
- **LINK(\( u \rightarrow v, \alpha, \beta \))**: Add edge \( uv \) to the forest and set \( \text{val}(u \rightarrow v) = \alpha \) and \( \text{val}(v \rightarrow u) = \beta \). Sentinel value \( \infty \) is a valid choice for \( \alpha \) or \( \beta \). The operation assumes \( u \) and \( v \) initially lie in different trees of the dynamic forest.
- **GETDARTVALUE(\( u \rightarrow v \))**: Returns the value \( \text{val}(u \rightarrow v) \). This operation assumes either \( uv \) or \( vu \) is an edge in the forest.
• **AddPath**($\Delta, u, v$): For each dart $x \rightarrow y$ on the directed path from $u$ to $v$, add $\Delta$ to $\text{val}(x \rightarrow y)$ and subtract $\Delta$ from $\text{val}(y \rightarrow x)$. The value $\text{val}(x \rightarrow y)$ does not change if it is $\infty$ before the operation, and $\text{val}(y \rightarrow x)$ does not change if it is $\infty$ before the operation. This operation assumes $u$ and $v$ lie in the same tree.

• **MinPath**($u, v$): Return a dart $x \rightarrow y$ on the directed path from $u$ to $v$ such that $\text{val}(x \rightarrow y)$ is minimized. Sentinel $\infty$ is cannot be the minimum value unless $\text{val}(x \rightarrow y) = \infty$ for every dart on the directed path from $u$ to $v$. This operation assumes $u$ and $v$ lie in the same tree.

• **Junction**($u, v, w$): Return the unique node that lies on the paths from $u$ to $v$, from $v$ to $w$, and from $w$ to $u$ in the forest. This operation assumes that nodes $u$, $v$, and $w$ lie in the same component of the forest.

With the proper modifications [62, 121], many dynamic forest data structures can be used to support these operations in $O(\log n)$ amortized arithmetic operations each, including link-cut trees [118] and self-adjusting top trees [122]. Supporting $\infty$ as a valid value on edges can be supported by simply declaring it larger during all comparisons with real values and always declaring $\infty$ the solution in addition and subtracting operations involving $\infty$. The only operation not standard to dynamic tree data structures is **Junction**, but its implementation is described by Cabello, Chambers, and Erickson [14].

We also require the use of a grove data structure as used by Cabello, Chambers, and Erickson [14] for their algorithm to compute multiple-source shortest paths in surface embedded graphs. Decompose the edges into two sets $R$ and $D$. We optionally denote two degree-1 vertices $a_0$ and $b_0$ as **designated anchors**, and guarantee that either $R$ is connected or every connected component of $R$ contains a designated anchor. Let $R$ connect the primal vertices into $\kappa$ components and let $D$ connect the dual vertices into $\kappa^*$ components. For our algorithm, both $\kappa$ and $\kappa^*$ are between 1 and 2. Subgraph $R$ contains at most $n - 1 - \kappa + \kappa' + 2g$ edges. We separate $R$ into $O(g)$ edge-disjoint subtrees as follows.

Let $\bar{R}$ be the subgraph of $R$ formed by repeatedly removing vertices of degree 1 that are not $a_0$ or $b_0$ until none remain. We call $\bar{R}$ the **2-core** of $R$. Let $H$ (for ‘hair’) be the subgraph removed from $R$ in the formation of $\bar{R}$. 

77
Subgraph $H$ forms a forest. Subgraph $\bar{R}$ at least contains every edge $e$ where the endpoints of the dual edge $e^*$ lie in different components of $D$ and if we assign designated anchors, every edge $e$ that lies on a simple path from $a_0$ to $b_0$.

**Lemma 6.2.1.** Subgraph $\bar{R}$ consists of up to $6g - 3\kappa + 3\kappa^* - 2 = O(g)$ paths $\pi_1, \pi_2, \ldots$ with up to $4g - 2\kappa + 2\kappa^* = O(g)$ endpoints on anchor vertices or vertices of degree at least 3.

**Proof:** Our proof closely follows one of Erickson and Har-Peled [49, Lemma 4.2]. Let $n_R$ and $m_R$ be the number of paths in a decomposition of $\bar{R}$ where the endpoints of the paths are either anchor vertices or vertices of degree at least 3. Let $\bar{G}$ be the graph of $n_R$ vertices and $m_R$ edges embedded on $\Sigma$ such that there is one edge in $\bar{G}$ per path $\pi_i$ and each edge in $\bar{G}$ shares an embedding with its associated path in $\bar{R}$. At most two vertices of $\bar{G}$ have degree 1, and the others have degree at least 3 so $3n_R - 4 \leq 2m_R$. Further, there are exactly $\kappa^*$ faces in the embedding of $\bar{G}$. However, $\bar{G}$ may not be cellularly embedded on $\Sigma$ because some of the faces may not be disks and $\bar{G}$ may not be connected. Adding up to $\kappa$ additional edges to $\bar{G}$ connects the graph, and adding non-trivial self loop edges to each non-disk face can guarantee a cellular embedding. By Euler’s formula we have $n_R - m_R - \kappa + \kappa^* \geq 2 - 2g$. Straightforward substitutions show $m_R \leq 6g - 3\kappa + 3\kappa^* - 2$ and $n_R \leq 4g - 2\kappa + 2\kappa^*$.

\[ \square \]

Note that up to two of these paths described above may be trivial (contain one vertex) if designated anchors are being used. For each index $i$, let $H_i$ denote the union of $\pi_i$ with all trees in $H$ that share a vertex with $\pi_i$. In the event that a tree $\tau$ in $H$ shares vertices with two paths $\pi_i$, we arbitrarily choose one subtree $H_i$ to contain $\tau$. We refer to each path $\pi_i$ as the anchor path and its endpoints $a_i$ and $b_i$ the anchor vertices of the corresponding subtree $H_i$. We may have $a_i = b_i$ if they are both the same designated anchor. The set of $O(g)$ edge-disjoint subtrees $\{H_1, H_2, \ldots\}$ form a grove.

We maintain the grove by storing the subtrees $H_i$ in a dynamic forest data structure. We maintain separate copies of each anchor vertex in this data structure, one for each $H_i$ that contains it. Note that the number of edges in $R$ is $O(n)$, so the total number of vertex copies is also $O(n)$. We also
record the correct copies of the anchor vertices of each subtree $H_i$. As in the
dynamic forest data structure, our grove data structure stores values on the
edges. Our grove data structure supports the following operations:

- **GROVE\text{LINK}(u\rightarrow v, \alpha, \beta)**: Add edge $uv$ to the grove, and set $\text{val}(x\rightarrow y) = \alpha$ and $\text{val}(v\rightarrow u) = \beta$. This operation assumes $u$ and $v$
are not in the same subtree $H_i$.

- **GROVE\text{Cut}(uv)**: Remove edge $uv$ from the grove.

To implement GROVE\text{LINK}(u\rightarrow v, \alpha, \beta), we first determine the subtrees $H_i$ and $H_j$ that contain $u$ and $v$ respectively. We find the vertices $\hat{a} = \text{JUNCTION}(u, a_i, b_i)$ and $\hat{b} = \text{JUNCTION}(v, a_j, b_j)$. Then we perform \text{LINK}(u\rightarrow v, \alpha, \beta) to merge $H_i$ and $H_j$ into a single tree $\hat{H}$. We split $\hat{H}$ into up to five smaller subtrees by splitting each of $\hat{a}$ and $\hat{b}$ that is not a designated anchor into three copies. Each copy carries up to three edges that lie along anchor paths, and we can perform the split using a constant number of \text{LINK} and \text{Cut} operations. Finally, we store the new anchor vertices for each of the new subtrees. See Figure 6.1.

To implement GROVE\text{Cut}(uv), we begin by determining which tree $H_i$ contains $uv$. Assuming neither $a_i$ nor $b_i$ is a designated anchor, we then merge three copies of anchors $a_i$ and $b_i$ into single vertices, merging five trees into a single tree $\hat{H}$. If one or both of $a_i$ or $b_i$ is a designated anchor, then we do not perform a merger on that anchor vertex. Let $c$ and $c'$ be the anchor vertices shared with the other copies of $a_i$, and let $d$ and $d'$ be the anchor
vertices shared with other copies of $b_i$ before the merger. We remove $uv$ from $\hat{H}$ by performing the dynamic tree operation $\text{CUT}(uv)$, creating two new subtrees of $\hat{H}$. Finally, we set $c$ and $c'$ to be the anchor vertices of one subtree while $d$ and $d'$ are the anchor vertices of the other. Both grove operations listed above take $O(\log n)$ amortized time.

6.3 Homological Maxflow/Mincut

In this section, we present the main motivation behind our algorithm, a homological variant of the classical maxflow/mincut theorem of Ford and Fulkerson [57]. Our version of the theorem generalizes one of Chambers et al. [23, Theorem 4.3] to work with arbitrary coflows and capacity functions that are not positive. We say a dart assignment $x : \vec{E} \to \mathbb{R}$ induces a coboundary if the coflow $\theta$ with $\theta(uv) = x(u \to v) - x(v \to u)$ is a coboundary. We say a graph $G$ contains a negative coboundary with respect to a capacity function $c : \vec{E} \to \mathbb{R}$ if there exists a dart assignment $x_0 : \vec{E} \to \mathbb{R}^+$ inducing a coboundary such that $\langle x_0, c \rangle < 0$.

**Theorem 6.3.1.** Let $G = (V, E)$ be a directed graph embedded on a surface of genus $g$, let $c : \vec{E} \to \mathbb{R}$ be a (not necessarily positive) capacity function, and let $\theta : E \to \mathbb{R}$ be a coflow in $G$. The maximum cost $\langle \phi, \theta \rangle$ of any feasible circulation $\phi$ in $G$ is well defined if and only if $c$ does not contain a negative coboundary. Further, the maximum cost of any feasible circulation is equal to the minimum capacity of any coflow cohomologous with $\theta$.

**Proof:** Our proof is based on arguments given by Chambers et al. [23, Section 4.2]. Consider the following linear programming formulation of the maximum cost circulation problem.

\[
\begin{align*}
\text{max} & \quad \sum_{uv \in E} \phi(uv) \cdot \theta(uv) \\
\text{s.t.} & \quad \sum_{vw \in E} \phi(vw) - \sum_{uw \in E} \phi(uv) = 0 \quad \text{for all } v \in V, \\
& \quad \phi(uv) \leq c(u \to v) \quad \text{for all } uv \in E, \\
& \quad -\phi(uv) \leq c(v \to u) \quad \text{for all } uv \in E.
\end{align*}
\]
The dual of this linear program has a variable \( \alpha(v) \) for each vertex \( v \) and a variable \( x(u \rightarrow v) \) for each dart \( u \rightarrow v \):

\[
\begin{align*}
\text{min} & \quad \sum_{u \rightarrow v} x(u \rightarrow v) \cdot c(u \rightarrow v) \\
\text{s.t.} & \quad \alpha(u) - \alpha(v) + x(u \rightarrow v) - x(v \rightarrow u) = \theta(uv) \quad \text{for all } uv \in E, \\
& \quad x(u \rightarrow v) \geq 0 \quad \text{for all } u \rightarrow v \in \vec{E}.
\end{align*}
\]

If we recast the dual program as in terms of the dual graph, we get the following dual linear program:

\[
\begin{align*}
\text{min} & \quad \sum_{f \uparrow g} x(f \uparrow g) \cdot c(f \uparrow g) \\
\text{s.t.} & \quad \alpha(g) - \alpha(f) + x(f \uparrow g) - x(g \uparrow f) = \theta(f \mid g) \quad \text{for all } f \mid g \in E^*, \\
& \quad x(f \uparrow g) \geq 0 \quad \text{for all } f \uparrow g \in \vec{E}^*.
\end{align*}
\]

The primal program cannot be unbounded because of the capacity constraints. As a consequence, the dual dual program must have at least one feasible solution. Further, if the maximum cost flow in the primal graph is not well defined, it must be because there is no feasible solution, which implies the dual dual program is unbounded.

Now, suppose there exists \( x_0 : \vec{E} \rightarrow \mathbb{R}^+ \) inducing a coboundary of some 0-cochain \( \alpha_0 \) such that \( \langle x_0, c \rangle < 0 \). Let \( \alpha_1(f) \) and \( x_1(f \uparrow g) \) be the variables to any feasible solution to the third linear program. Replacing each variable \( x_1(f \uparrow g) \) with \( x_1(f \uparrow g) + x_0(f \uparrow g) \) and each variable \( \alpha_1(f) \) by \( \alpha_1(f) - \alpha_0(f) \) yields a feasible solution of strictly lower value. The third linear program is unbounded and the primal program is infeasible.

Suppose instead that the primal program is infeasible, and the third linear program is unbounded. Let \( \alpha_1(f) \) and \( x_1(f \uparrow g) \) be the variables to any feasible solution to the third linear program. The set of feasible solutions to the third program lies in an unbounded polytope in \( \mathbb{R}^{2m} \). There must exist an \( \alpha_0 \) and \( x_0 \) such that the ray \( \alpha_\lambda = \alpha_1 + \lambda \alpha_0, \ x_\lambda = x_1 + \lambda x_0 \) specifies a feasible solution for every real \( \lambda \geq 0 \). Further, as \( \lambda \) increases, the objective value of the solutions decrease. Because no point on the ray violates the condition \( x(f \uparrow g) \geq 0 \) for any dart \( f \uparrow g \), we know \( x_0(f \uparrow g) \geq 0 \) for every \( f \uparrow g \). Because the objective decreases as \( \lambda \) increases, \( \langle x_0, c \rangle < 0 \).
Finally, we have
\[ \alpha_1(g) - \alpha_1(f) + x_1(f \uparrow g) - x_1(g \uparrow f) = \theta(f|g) \]
and
\[ \alpha_\lambda(g) - \alpha_\lambda(f) + x_\lambda(f \uparrow g) - x_\lambda(g \uparrow f) = \theta(f|g) \]
by the equalities in the dual dual program. Definition chasing shows us
\[ x_0(f \uparrow g) - x_0(g \uparrow f) = \alpha_0(f) - \alpha_0(g). \]
The dart assignment \( x_0 \) induces the coboundary of the 0-cochain \(-\alpha_0\).

To complete the proof of Theorem 6.3.1, we assume from here on that the dual dual has an optimal feasible solution. Let \( \theta' \) be any coflow cohomologous to \( \theta \). Let \( x' : \vec{E} \rightarrow \mathbb{R}^+ \) be an assignment to the darts such that for each dart \( f \uparrow g \) we have \( x'(f \uparrow g) = \theta'(f \uparrow g) \) if \( \theta'(f \uparrow g) \geq 0 \) and \( x'(f \uparrow g) = 0 \) otherwise. Let \( \alpha' : v \rightarrow \mathbb{R} \) be a 0-cochain such that \( \theta' = \theta - \partial \alpha' \). Setting the dual dual program variables according to \( x' \) and \( \alpha' \) give a feasible solution with the same objective value as \( \theta' \)'s capacity. The optimal solution to the dual dual program as value no more than the minimum capacity of any coflow cohomologous to \( \theta \).

Finally, let \( \alpha_{OPT}(f) \) and \( x_{OPT}(f \uparrow g) \) denote the variables in some optimal solution. The variables \( \alpha_{OPT} \) can be interpreted as a 0-cochain as we have done in the previous paragraphs. We see
\[ x_{OPT}(f \uparrow g) - x_{OPT}(g \uparrow f) = \theta(f|g) + \alpha_0(f) - \alpha_0(g), \]
implying \( x_{OPT} \) induces a coflow \( \theta_{OPT} \) cohomologous with \( \theta \). Consider any dual edge \( f|g \in E^* \). We have \( c(f \uparrow g) + c(g \uparrow f) \geq 0 \). Otherwise, the dart assignment \( x_0 : \vec{E} \rightarrow \mathbb{R}^+ \) where \( x_0(f \uparrow g) = x_0(g \uparrow f) = 1 \) and \( x_0(\vec{e}) = 0 \) for all other \( \vec{e} \) would induce a coboundary while having \( \langle x_0, c \rangle < 0 \). We may assume that exactly one of \( x_{OPT}(f \uparrow g) \) and \( x_{OPT}(g \uparrow f) \) is greater than 0.

The objective of the dual dual program is the capacity of \( \theta_{OPT} \) by definition, meaning the dual dual program has an optimal solution with value no less than the minimum capacity of any coflow cohomologous to \( \theta \).

Complementary slackness immediately implies the following corollary:
Corollary 6.3.2. Let $\phi$ be a feasible circulation of $G$ and let $\theta$ be a coflow of $G$. Circulation $\phi$ maximizes the value $\langle \phi, \theta \rangle$ and $\theta$ is a minimum capacity coflow for some cohomology class of $G$ if and only if $\phi$ saturates every dart $x \rightarrow u = f \uparrow g$ such that $\theta(f \uparrow g) > 0$.

Using this corollary, we can easily show that the optimal coflow is resilient to changes in residual capacities created by circulations.

Lemma 6.3.3. Let $\phi$ be any circulation in $G$, and let $\theta^*$ be a minimum capacity coflow for some cohomology class of $G$. Coflow $\theta^*$ is minimum capacity for its homology class in residual graph $G_\phi$.

Proof: Let $\phi^*$ be a feasible circulation of $G$ that maximizes the value $\langle \phi, \theta^* \rangle$. By Corollary 6.3.2, $\phi^*$ saturates every dart $x \rightarrow u = f \uparrow g$ such that $\theta^*(f \uparrow g) > 0$. Circulation $\phi^* - \phi$ saturates those darts as well when applied to $G_\phi$. \qed

6.4 Parametric Minimum Capacity Coflows

Let $\theta$ be an arbitrary coflow. For any real $\lambda$, let $f_\lambda$ denote an arbitrary (not necessarily feasible) $s,t$-flow of value $\lambda$. Abusing notation, we let $G_\lambda = G_{f_\lambda}$ and $c_\lambda = c_{f_\lambda}$. For all $\lambda$ such that $G_\lambda$ does not contain a negative capacity coboundary, let $\theta_\lambda$ be a minimum capacity coflow cohomologous to $\theta$ in the dual residual graph $G_\lambda$. Coflow $\theta_\lambda$ cannot necessarily stay the same for all choices of $\lambda$. However, Lemma 6.3.3 implies that the choice of $f_\lambda$ does not effect the set of valid $\theta_\lambda$.

Our algorithm is based on the following process on a graph $G$ with positive capacity function $c$. Residual graph $G_0$ contains no negative capacity coboundaries. In fact, residual graph $G_\lambda$ does not contain a negative capacity coboundary for all $\lambda$ between 0 and the value of a maximum $s,t$-flow, because there exist flows of all lessor values that are feasible. However, if $\lambda$ exceeds the value of a maximum $s,t$-flow, all minimum $s,t$-cuts will become over saturated and their outgoing darts will define negative capacity coboundaries.

Set $\lambda := 0$, and consider the well-defined coflow $\theta_\lambda$. We continuously increase $\lambda$ and track changes to $\theta_\lambda$ as we do so until $G_\lambda$ contains a negative coboundary. There are periods during which $\lambda$ increases but $\theta_\lambda$ remains
unchanged. However, there are certain critical values of \( \lambda \) such that \( \theta_\lambda \) must change to remain minimum capacity for its cohomology class.

In order to more easily explain the changes that occur to \( \theta_\lambda \), we invoke Corollary 6.3.2 and Lemma 6.3.3 to assume \( f_\lambda \) is a feasible \( s, t \)-flow of \( G \) that saturates all darts \( u \rightarrow v = f \uparrow g \) such that \( \theta_\lambda(f \uparrow g) > 0 \). These same darts all have 0 residual capacity in \( G_\lambda \), meaning the capacity of \( \theta_\lambda \) in \( G_\lambda \) is equal to 0.

Now fix a value of \( \lambda \) such that \( G_\lambda \) contains no negative capacity coboundaries. Let \( D \subset E \) contain every edge \( e \) such that \( \theta_\lambda(e^*) \neq 0 \). Let \( R = E \setminus D \). If there exists an \( s, t \)-path \( p \) in \( R \) such that every dart on \( p \) is residual, then augmenting \( f_\lambda \) along \( p \) gives us a new feasible flow \( f'_\lambda \) where \( \lambda' > \lambda \) and every dart \( u \rightarrow v = f \uparrow g \) such that \( \theta_\lambda(f \uparrow g) > 0 \) is still saturated. By Corollary 6.3.2, \( \theta_{\lambda'} = \theta_\lambda \). An augmentation step is the act of either augmenting along a path in \( R \) or determining that no residual paths exists in \( R \).

Now, suppose that \( R \) contains no residual \( s, t \)-path. There exists an \( s, t \)-cut \( S \) in \( G \) such that for every dart \( u \rightarrow v = f \uparrow g \) leaving \( S \) either \( \theta_\lambda(f \uparrow g) \neq 0 \) or \( c_\lambda(u \rightarrow v) = 0 \). If there exists at least one dart such that \( \theta_\lambda(f \uparrow g) < 0 \), then let \( w \) be the maximum value of \( \theta_\lambda(f \uparrow g) \) among all such darts. Let \( \alpha : V \rightarrow \mathbb{R}^+ \) be a 0-cochain such that \( \alpha(v^*) = -w \) for all \( v \in S \) and \( \alpha(v^*) = 0 \) for all \( v \notin S \). We create a new coflow \( \theta'_\lambda = \theta_\lambda + \partial \alpha \) cohomologous with \( \theta_\lambda \). See Figure 6.2. Because for every dart \( f \uparrow g \) in \( \partial \alpha \) either \( c_\lambda(f \uparrow g) = 0 \) or \( c_\lambda(g \uparrow f) = 0 \) and \( \theta_\lambda(g \uparrow f) + \partial \alpha(g \uparrow f) \geq 0 \), the capacity of \( \theta'_\lambda \) is 0. By Corollary 6.3.2, coflow \( \theta'_\lambda \) has minimum capacity for its cohomology class. Coflow \( \theta'_\lambda \) also induces a new set of edges with non-zero coflow \( D' \) and a new set of edges \( R' = E \setminus D' \), revealing another chance to augment \( f_\lambda \). We call each such change in \( \theta_\lambda \) a pivot. In Section 6.7, we explain how for certain
choices of $\theta$ we can guarantee a finite number of pivots before either an augmenting path is found or we discover that $\lambda$ cannot be increased without creating a negative capacity coboundary.

Finally, we discuss what occurs if there is no residual $s,t$-path in $R$ and no dart $u \rightarrow v = f \uparrow g$ leaving $S$ such that $\theta_\lambda(f \uparrow g) < 0$. In this case, every dart leaving $S$ must be saturated. By the classic maxflow/mincut theorem, we know that $\lambda$ is the value of a maximum flow, and it cannot increase further without creating a negative capacity coboundary.

The “parametric minimum capacity coflow” process described above is similar to the parametric shortest path process considered by Erickson [46] in his interpretation of Borradaile and Klein’s [9] algorithm for computing maximum flows in planar graphs. In the next section, we describe a choice for $\theta_0$ based on a shortest path tree in $G^*$. If $G$ is embedded in the plane, the darts with positive flow in each $\theta_\lambda$ will match the darts in the shortest path trees of each $G_\lambda$. In some sense, our procedure tracks the same objects as Erickson’s.

### 6.5 Shortest Path Coflows

In the previous section, we sketched a procedure for updating a minimum capacity coflow $\theta_\lambda$ as $\lambda$ continuously increases. However, it is not clear that individual augmentation steps and pivots can be completed efficiently. In fact, saturating all the residual paths of $R$ sounds like a maximum flow problem itself. In this section, we describe an efficiently computable choice for $\theta_0$ such that augmentation steps leading to a pivot and pivots themselves can be performed in $O(g \log n + g^2)$ amortized time each given the correct data structures.

Our algorithm for maximum flows begins as follows. Let $o$ be an arbitrary vertex in the dual graph $G^*$. (To more closely mirror the maximum flow algorithm of Borradaile and Klein [9], we may choose a vertex $o$ on the face $t^*$. ) Let $\tau$ be a shortest path tree on the darts of $G^*$ rooted at $o$ where the length of each dart is equal to its capacity. Our algorithm computes $\tau$ in $O(n \log n)$ time simply by running Dijkstra’s algorithm. It then computes the coflow $\theta_0$ using the following recursive procedure on $\tau$. For every dual edge $e$ with neither of its darts in $\tau$, our algorithm assigns $\theta_0(e) := 0$. For
every vertex \( q \neq o \) in the postorder traversal of \( \tau \) and every dart \( p\rightarrow q \) in \( \tau \), it makes assignments to \( \theta_0 \) such that

\[
\theta_0(p\rightarrow q) = 1 + \sum_{q\rightarrow r\in \tau} \theta_0(q\rightarrow r).
\]

Our algorithm then computes the primal flow \( f_0 \) using a similar procedure to that used by Borradaile and Klein [9]. For each dual vertex \( q \), let \( d(q) \) denote the shortest path distance from \( o \) to \( q \). Define the slack of each dual dart \( r\rightarrow q \) as follows:

\[
\text{slack}(r\rightarrow q) = d(r) + c(r\rightarrow q) - d(q).
\]

We interpret \( d \) as a 2-chain and let \( f_0 = \partial d \). The slack of each dual dart \( r\rightarrow q \) is exactly equal to the residual capacity of that dart in the primal graph. By Ford’s classical formulation of shortest paths [56], we know the slacks and therefore residual capacities are all non-negative. Further, the residual capacities of all darts with positive coflow in \( \theta_0 \) are equal to 0, guaranteeing \( \theta_0 \) is a minimum capacity coflow by Corollary 6.3.2.

Our algorithm follows the procedure given in Section 6.4 by partitioning the edges of \( G \) into two disjoint sets \( (R,D) \). Between every pivot step, set \( D \) contains the edges with non-zero coflow. Our algorithm maintains the invariant that subgraph \( R \) never has a component containing neither \( s \) nor \( t \).

We have the following lemma.

**Lemma 6.5.1.** Let \( \theta_\lambda \) be any minimum capacity coflow computed during over the course of our algorithm. Subgraph \( D \) is connected.

**Proof:** The coflow \( \theta_0 \) is such that for every dual vertex \( q \neq o \), we have

\[
\sum_{rq} \theta_0(rq) - \sum_{qr} \theta_0(qr) = 1.
\]

In other words, \( q \) receives an excess of 1 unit of coflow. Dual vertex \( o \) is the only dual vertex supplying coflow to the other dual vertices. Therefore, a non-zero amount of coflow passes from \( o \) to \( q \). Subgraph \( D \) contains a spanning tree. \( \square \)

As the parametric minimum capacity coflow procedure progresses, our algorithm stores the following values. For each dart \( u\rightarrow v \), our algorithm stores
its residual capacity \( c_\lambda(u\rightarrow v) \) for the current primal flow \( f_\lambda \). For each dual dart \( f \uparrow g \), our algorithm stores \( \theta_\lambda(f \uparrow g) \) if \( \theta_\lambda(f \uparrow g) > 0 \). Otherwise, it stores the sentinel value \( \infty \) for that dart. Note that our algorithm does not store \( f_\lambda \) explicitly. It can easily compute \( f_\lambda \) once the parametric minimum capacity coflow procedure finishes in linear time using the explicitly stored residual capacities of the darts. In the next section, we describe how our algorithm stores the edges and dart values for members of \( R \) and \( D \). We do so while explaining how to efficiently implement augmenting steps and pivots.

### 6.6 Augmentation Steps and Pivots

In this section, we describe the data structures and procedures used by our algorithm to perform augmentation steps and pivots in \( O(g \log n + g^2) \) amortized time each.

#### 6.6.1 Augmentation steps

We begin by describing how our algorithm efficiently performs augmentation steps. Our algorithm explicitly stores the residual capacities and coflow values for edges outside of \( R \) and \( D \) respectively. Otherwise, the edges and their values are stored in more sophisticated data structures as defined below.

In order to perform the augmentation steps, it stores the edges of \( R \) in a grove data structure as described in Section 6.2 which we call the **primal grove**. The primal grove contains \( s \) and \( t \) as designated anchors. For each dart \( u\rightarrow v \) in the primal grove, the value \( \text{val}(u\rightarrow v) = c_\lambda(u\rightarrow v) \). We assume \( R \) is connected at the beginning of each augmentation step.

To efficiently saturate an \( s,t \)-cut in \( R \), our algorithm creates a simple flow model graph \( G' = (V', E') \) with \( O(g) \) vertices and edges. The vertices of \( G' \) are the \( O(g) \) anchor vertices of the primal grove. Graph \( G' \) contains an edge \( uv \) between every pair of anchor vertices \( u \) and \( v \) that share an anchor path in the grove. Let \( c'(u\rightarrow v) \) be the capacity of dart \( u\rightarrow v \) in \( G' \). There are no simple paths between anchor vertices that go through hair in \( R \). Also, no pair of anchor paths share edges. We conclude that any augmentation of \( f_\lambda \) is feasible if and only if it sends up to \( c_\lambda(\text{MinPath}(u_i, v_i)) \) units of
flow along each anchor path with endpoints $u_i$ and $v_i$. Our algorithm sets $c'(u \rightarrow v) = \text{GetDartValue}(\text{MinPath}(u_i, v_i))$ where $u_i$ and $v_i$ are copies of $u$ and $v$ that share an anchor path. The construction of $G'$ takes $O(g \log n)$ amortized time.

With the model graph $G'$ fully defined, our algorithm applies any reasonably efficient method to compute the maximum $s,t$-flow $f'$ in $G'$. This maximum flow can be computed in $O(g^2 / \log g)$ time using a recent algorithm due to Orlin [106]. The flow found in $G'$ can easily be applied along edges of $R$ by performing $\text{AddPath}(f'(u \rightarrow v), u_i, v_i)$ to every anchor path in the primal grove with anchor vertices $u_i$ and $v_i$. Augmenting $f_\lambda$ takes $O(g \log n)$ amortized time.

After applying the flow $f'$ to $R$, our algorithm computes a minimum $s,t$-cut $S'$ of $G'$ and the darts $C'$ leaving $S'$ in $O(g)$ time. Let $C$ be an initially empty set of edges from $R$. For each dart $u \rightarrow v$ in $C'$, our algorithm does the following. It finds a saturated dart $x \rightarrow y$ from the anchor path containing anchor vertices $u_i$ and $v_i$ by performing $\text{MinPath}(u_i, v_i)$. It then adds edge $xy$ to $C$ and then performs $\text{GroveCut}(xy)$ to remove $xy$ from the primal grove and $R$. Once this procedure is performed for each dart $u \rightarrow v$ in $C'$, the subgraph $R$ contains two components with $s$ and $t$ lying in different components. The set of darts leaving $s$’s component $S$ are all saturated, and our algorithm can move on to perform a pivot. Computing $C$ takes $O(g \log n)$ amortized time, so the entire augmentation step takes $O(g \log n + g^2 / \log g)$ amortized time.

### 6.6.2 Pivots

In order to efficiently perform pivots, our algorithm stores the edges of $D$ in a dual grove data structure with no designated anchor vertices. For each dart $f \uparrow g$ in the dual grove, the value $\text{val}(f \uparrow g) = \theta_\lambda(f \uparrow g)$ if $\theta_\lambda(f \uparrow g) > 0$. Otherwise, $\text{val}(f \uparrow g) = \infty$. Let $S$ be the connected component of $R$ containing $s$ (which our algorithm does not explicitly compute). Let $C$ be the set of saturated darts computed at the end of the augmentation step immediately preceding a pivot. Let $S'$ be the set of primal grove anchor vertices that lie in $S$ which our algorithm also computes in the augmentation step.

Our algorithm performs a $\text{GroveLink}(f \uparrow g, 0, \infty)$ operation for each
dart \ f \uparrow g \text{ in } C. \text{ Dart set } C \text{ contains } O(g) \text{ darts, so this process takes } O(g \log n) \text{ amortized time. After the darts are added to the dual grove, } D \text{ contains a set of edges (including all edges in } C) \text{ that separate } S \text{ from the rest of the primal vertices. In order to terminate or pivot, our algorithm must either determine that every dart leaving } S \text{ has non-negative coflow, or it must determine the minimum coflow value for any dart entering } S. \text{ The anchor paths } \pi_1, \pi_2, \ldots \text{ from the dual grove cut the surface into faces. One of these faces contains exactly the primal vertices in } S. \text{ Therefore, for each anchor path } \pi_i \text{ with anchor vertices } p_i \text{ and } q_i, \text{ one of the following must hold: (1) Every dart on the path from } p_i \text{ to } q_i \text{ has } S \text{ only on its left, (2) every dart on the path from } p_i \text{ to } q_i \text{ has } S \text{ only on its right, or (3) no dart on the path from } p_i \text{ to } q_i \text{ lies on the boundary of } S. \text{ Our algorithm determines the set of paths } \Pi = \pi_1, \pi_2, \ldots \text{ such that the dual darts of each } \pi_i \text{ lie on the boundary of } S, \text{ and then directs them so that } S \text{ lies to the right of the darts. The algorithm determines these paths in } O(g^2) \text{ time simply by picking a representative primal dart } x \rightarrow y \text{ from each of the } O(g) \text{ anchor paths in the dual grove and testing if exactly one of } x \text{ or } y \text{ shares a primal grove tree with a member of } S'. \text{ For each anchor path } \pi_i \text{ in } \Pi \text{ from anchor vertex } p_i \text{ to } q_i, \text{ our algorithm perform a } \text{MinPath}(q_i, p_i) \text{ operation to find a dual dart } g \uparrow f \text{ with } g^* \in S \text{ that minimizes } \theta_\lambda(g \uparrow f) \text{ while having } \theta_\lambda(g \uparrow f) > 0. \text{ These MinPath operations take } O(g \log n) \text{ amortized time total. Let } g \uparrow f \text{ be the dual dart returned by the operation, and let } w \text{ be the value returned by GetDartValue}(g \uparrow f). \text{ If } w = \infty, \text{ then there is no dual dart with with positive coflow and } S \text{ on its left (every primal dart leaving } S \text{ has } 0 \text{ or positive coflow). The algorithm may safely report that } f_\lambda \text{ is a maximum flow. Otherwise, it performs a pivot by performing AddPath}(w, p_i, q_i) \text{ for each anchor path } \pi_i \text{ in } \Pi \text{ from anchor vertex } p_i \text{ to } q_i. \text{ During this } O(g \log n) \text{ amortized time operation, every dart in } C \text{ receives positive coflow } w. \text{ The dual grove now stores a new coflow } \theta'_\lambda. \text{ Finally, the algorithm needs to remove darts from } D \text{ that contain no coflow in } \theta'_\lambda. \text{ Let } L \text{ be an initially empty set of edges. For each anchor path } \pi_i \text{ in } \Pi \text{ from anchor vertex } p_i \text{ to } q_i, \text{ our algorithm does the following. It performs a } \text{MinPath}(q_i, p_i) \text{ operation which returns a dart } g \uparrow f. \text{ If GetDartValue}(g \uparrow f) \text{ returns } 0, \text{ then the algorithm performs Cut}(f|g) \text{ and Link}(g \uparrow f, \infty, \infty) \text{ so that dart } f \uparrow g \text{ cannot be the result of } \text{MinPath}(q_i, p_i) \text{ again. Primal edge } (f|g)^* \text{ is then added to } L, \text{ and the loop}
over anchor paths continues on $\pi_i$ again in case any other darts of $\pi_i$ have 0 coflow. If instead $\text{GetDartValue}(g \uparrow f)$ returns anything greater than 0, the loop continues with the next member of $\Pi$. Each possible addition of an edge to $L$ takes $O(\log n)$ amortized time. Lemma 6.5.1 guarantees that each pivot sets at most $O(g)$ edges to have 0 coflow, and there are $O(g)$ anchor paths, so the above process takes $O(g \log n)$ amortized time.

Now that $L$ is populated with every edge carrying 0 coflow, the algorithm moves every member of $L$ from $D$ to $R$. For each edge $uv$ in $L$, the algorithm performs $\text{GroveCut}((uv)^*)$ in the dual grove and then $\text{GroveLink}(u \rightarrow v,c,\lambda(u \rightarrow v),c,\lambda(v \rightarrow u))$ in the primal grove. The pivot takes $O(g \log n + g^2)$ amortized time total.

6.7 Time Bounds

In Sections 6.5 and 6.6, we described an efficiently computable coflow $\theta_0$ that is minimum capacity for its cohomology class. We then explained how our algorithm performs augmentation steps and pivots efficiently during a parametric minimum capacity coflow procedure that starts with $\theta_0$. In this section, we give a quadratic time bound on the number of pivots performed by our algorithm before we can be sure it has already computed a maximum $s,t$-flow. This bound implies our algorithm runs in time $O(gn^2(g + \log n))$. We acknowledge that this running time does not meet our goal of $O(g^k n \log n)$ for some small $k$, but it does provide a “proof of concept” that our topology based algorithm does run in strongly polynomial time.

Let $\pi$ be any $s,t$-path in $G$, and let $\theta$ be any coflow in $G$. Let $f_\pi$ denote the $s,t$-flow that sends one unit of flow along every dart in $\pi$ so that $f(u \rightarrow v) = 1$ for any $u \rightarrow v$ in $\pi$ and $f(u \rightarrow v) = 0$ for any $u \rightarrow v$ not on $\pi$ or its reverse. We define the \textbf{winding number} of $\theta$ with respect to $\pi$ as $\langle f_\pi, \theta \rangle$.

The parametric minimum capacity coflow procedure maintains a coflow $\theta_\lambda$ that is minimum capacity in the residual graph for some $s,t$-flow $f_\lambda$. During a pivot, $\theta_\lambda$ is modified by pushing $w \geq 0$ units of coflow around an $s,t$-cut, creating the cohomologous coflow $\theta'_\lambda$. For any $s,t$-path $\pi$, pushing these $w$ units of coflow increases the winding number of $\theta_\lambda$ with respect to $\pi$ by $w$. In other words, $\langle \pi, \theta'_\lambda \rangle = \langle \pi, \theta_\lambda \rangle + w$. In fact, if the procedure pushes a total of $w$ units around several $s,t$-cuts, then the winding number still increases.
by \( w \). The following lemma is immediate.

**Lemma 6.7.1.** Let \( \theta \) be any coflow in \( G \), and let \( \theta' \) be the result of pushing a total of \( w \) units of coflow on \( \theta \) around one or more \( s,t \)-cuts. Let \( \pi \) be any simple \( s,t \)-path avoiding edges \( e \) with \( \theta'(e^*) \neq 0 \). Assuming \( \pi \) is well defined, we have

\[
\langle f_\pi, \theta \rangle = -w.
\]

Now, consider our algorithm as described in Sections 6.5 and 6.6. Note that \( \theta_0 \) is integral. Let \( S \) be the component of \( R \) containing \( s \) at the beginning of some pivot step. We may inductively assume that all edges have integral coflow, and so any dual dart with \( S \) on its left and positive coflow has at least one unit. Therefore, at least one unit of coflow is pushed around an \( s,t \)-cut every pivot, even if no primal flow is pushed during an augmentation step. Let \( \theta_\lambda \) be some coflow created during a pivot. Let \( \pi \) be a simple \( s,t \)-path avoiding edges \( e \) with \( \theta_\lambda(e^*) \neq 0 \). For each dual dart \( f \uparrow g \), we have \( \theta_0(f \uparrow g) > -|F| + 1 \). Also, exactly \( |F| - 1 \) dual darts have non-zero coflow in \( \theta_0 \). Therefore, \( \langle f_\pi, \theta_0 \rangle \geq -(|F| - 1)^2 \). Path \( \pi \) can only exist if coflow \( \theta_\lambda \) is created within \( (|F| - 1)^2 \) pivots. We can safely terminate our algorithm if the total number of pivots is higher than \( (|F| - 1)^2 \), because there will never be an \( s,t \)-path available to the augmenting step after that point. The flow must already be maximized.

**Theorem 6.7.2.** Our algorithm for computing a maximum \( s,t \)-flow in an \( n \)-vertex graph embedded on an orientable surface of genus \( g \) runs in \( O(gn^2(g + \log n)) \) time.

### 6.8 Conclusions and Future Directions

In this chapter, we described a new algorithm for computing maximum flows in surface embedded graphs. We feel our algorithm is in a sense the best shot we have right now for solving the maximum flow problem in \( O(g^kn \log n) \) time for some small constant \( k \). It very directly generalizes the planar maximum flow algorithm of Borradaile and Klein [9], and its efficient implementation is relatively straightforward.

The ideas in this chapter have applications to other problems besides solving maximum flows in surface embedded graphs. By modifying the para-
metric minimum capacity coflow procedure presented in Section 6.4, it is possible to give high level procedures for computing multiple-source multiple-sink maximum flows or even maximum cost circulations in planar and more general surface embedded graphs. However, it is not clear how to efficiently perform augmentations (increasing the quality of the primal flow/circulation) or pivots in these applications. If the algorithm presented in this chapter can be shown to have a fast implementation, then finding faster algorithms for these other problems would make a natural research direction.
References


97


