Homology Flows, Cohomology Cuts[*]

Erin W. Chambers[†] Jeff Erickson[‡] Amir Nayyeri[§]

Submitted to *SIAM Journal on Computing* (STOC 2009 special issue) — August 1, 2010
Revised and resubmitted September 2, 2011

Abstract

We describe the first algorithm to compute maximum flows in surface-embedded graphs in near-linear time. Specifically, given a graph embedded on a surface of genus g, with two specified vertices s and t and integer edge capacities that sum to C, our algorithm computes a maximum (s,t)-flow in $O(g^8 n \log^2 n \log^2 C)$ time. We also present a combinatorial algorithm that takes $g^{O(g)} n^{3/2}$ arithmetic operations. Except for the special case of planar graphs, for which an $O(n \log n)$-time algorithm has been known for 20 years, the best previous time bounds for maximum flows in surface-embedded graphs follow from algorithms for general sparse graphs. For graphs of any fixed genus, our algorithms improve these time bounds by roughly a factor of $\sqrt{n}$. Our key insight is to optimize the homology class of the flow, rather than directly optimizing the flow itself; two flows are in the same homology class if their difference is a weighted sum of directed facial cycles. A dual formulation of our algorithm computes the minimum-cost circulation in a given (real or integer) homology class.

Errors, like straws, upon the surface flow;
He who would search for pearls must dive below.
— John Dryden, *All for Love*, Prologue (1677)

[*]Research partially supported by NSF grant DMS-0528086. An extended abstract of this paper was presented at the 50th ACM Symposium on Theory of Computing [19]. Please see http://www.cs.uiuc.edu/~jeffe/pubs/surflow.html for the most recent version of this paper.

[†]Department of Computer Science and Mathematics, Saint Louis University. Portions of this work were done while this author was affiliated with the Department of Computer Science, University of Illinois, Urbana-Champaign.

[‡]Department of Computer Science, University of Illinois, Urbana-Champaign. Portions of this work were done while this author was visiting IST Austria.

[§]Department of Computer Science, University of Illinois, Urbana-Champaign. Portions of this work were done while this author was visiting the Toyota Institute of Technology, Chicago.

1 Introduction

Planar graphs are natural targets for study. In addition to modeling real-world scenarios ranging from road networks to VLSI layouts, they often admit much faster algorithms compared to more general graphs. Most algorithms for planar graphs have been generalized to larger families of graphs, such as graphs of higher genus, graphs with forbidden minors, or graphs with small separators. Examples include single-source and multiple-source shortest paths [16, 42, 63, 83, 84, 86, 110]; minimum spanning trees [87, 100]; graph and subgraph isomorphism [35, 36, 54, 66, 90]; and approximation algorithms for the traveling salesman problem, Steiner trees, and other NP-hard problems [8, 12, 14, 28, 36, 46].

A stark exception to this general pattern is the classical maximum flow problem and its dual, the minimum cut problem. Flow and cuts were originally developed as tools for studying railway and other transportation networks [59], which are naturally modeled as planar graphs; Ford and Fulkerson's seminal paper [43] includes an algorithm for planar networks where the source and target vertices lie on the same face. A long series of results has led to planar maximum-flow algorithms that run in $O(n \log n)$ time, first for undirected graphs [44, 61, 102] and more recently for directed graphs [11, 13, 38]. Despite more than half a century of attention on flows in planar graphs, surprisingly little is known about flows in these more general graph families. Even for graphs embedded on the torus, the fastest algorithms to compute maximum flows are no faster than for arbitrary sparse graphs.

This paper describes the first algorithm to find maximum flows in surface-embedded graphs in near-linear time when the genus is fixed. The input to our problem is a graph $G = (V, E)$ embedded on a surface of genus g, along with two vertices s and t and a capacity function $c \colon E \to \mathbb{R}^+$. For any fixed genus g and polynomially-bounded integer capacities, our algorithm runs in $O(n \operatorname{polylog} n)$ time (bit operations). We also describe a combinatorial algorithm that runs in $O(n^{3/2})$ time (arithmetic operations) for arbitrary real capacities, for graphs of any fixed genus. In related work [20, 40], we describe the first algorithms to compute minimum cuts in undirected surface-embedded graphs in $O(n \log n)$ time for any fixed genus, using different techniques.

Before describing our results in more detail, we review several previous related results. Table 1 summarizes the fastest known maximum flow algorithms for several related families of graphs, including our new results. For general background on maximum flow algorithms and related results, we refer the reader to monographs by Ahuja *et al.* [4] and Schrijver [105].

Sparse graphs	$O(n^2 \log n)$	[108, 49]
Sparse graphs with integer capacities	$O(n^{3/2} \log n \log U)$	[48]
Planar undirected graphs	$O(n \log \log n)$	[72]
Planar directed graphs	$O(n \log n)$	[11, 13]
Planar graphs plus k edges	$O(k^3 n \log n)$	[65]
Surface graphs	$g^{O(g)} n^{3/2}$	Theorem 3.17
Surface graphs with integer capacities	$O(g^8 n \log^2 n \log^2 C)$	Theorem 3.16

Table 1. Fastest known maximum flow algorithms for several families of graphs. Here, n is the number of vertices; g is the genus of the surface; U is the maximum edge capacity; and C is the sum of the edge capacities.

Flows in sparse graphs. Euler's formula implies that an n-vertex graph embedded on a surface of genus $O(n)$ has at most $O(n)$ edges. The fastest known combinatorial maximum-flow algorithms for sparse graphs, due to Sleator and Tarjan [108] and Goldberg and Tarjan [49], run in time $O(n^2 \log n)$. The *minimum-cost* maximum flow can be computed in $O(n^2 \log^2 n)$ time using an algorithm of Orlin [97]. (For graphs with small separators, the running time of Orlin's algorithm can be improved to $O(n^2 \log n)$ by replacing Dijkstra's algorithm with a linear-time shortest-path algorithm [63, 110].) The fastest algorithm known for integer capacities, due to Goldberg and Rao [48], runs in $O(n^{3/2} \log n \log U)$ time, where U is an upper bound on the edge capacities. A more recent algorithm of Diatch and Spielman [31] computes the minimum-cost maximum flow in $O(n^{3/2} \operatorname{polylog} n \log U)$ time.

Flows in planar graphs. Maximum flows in planar graphs have received considerable attention for more than 50 years. Weihe [116] and Borradaile and Klein [11, 13] describe the history of planar flow algorithms in detail; we describe only a few important highlights.

Itai and Shiloach exploited the connection between maximum flows in an undirected planar graph and shortest paths in its dual graph to obtain an $O(n \log n)$-time algorithm when the source and sink vertices lie on a common face [71]; see also Hassin [60].

Reif [102] developed a divide-and-conquer algorithm to compute a minimum cut, and thus the maximum flow *value*, in a planar undirected network in $O(n \log^2 n)$ time. Reif's algorithm was extended by Hassin and Johnson to compute the actual maximum flow in $O(n \log n)$ additional time, using a carefully structured dual shortest-path computation [61]. Frederickson subsequently improved Reif's algorithm to $O(n \log n)$ time [44]. Frederickson's improvement can also be obtained more directly using more recent planar shortest-path algorithms [16, 63, 83, 110]. Very recently, after almost 25 years without progress, Italiano *et al.* [72] described an improved algorithm that runs in $O(n \log \log n)$ time.

Maximum flows in *directed* planar graphs were first investigated by Johnson and Venkatesan [74], who described a divide-and-conquer algorithm, based on recursive separator decompositions, with running time $O(n^{3/2} \log n)$. Venkatesan [114] observed that a feasible flow with a given *value*, if such a flow exists, can be computed in $O(n^{3/2})$ time by computing a single-source shortest path tree in a dual graph with both positive and negative edge weights, using an algorithm of Lipton, Rose, and Tarjan [86]. (Venkatesan's reduction is described in greater detail in Section 3.1.) For graphs with integer capacities, binary search over the possible flow values immediately yields a max-flow algorithm that runs in $O(n^{3/2} \log C)$ time, where C is the sum of the capacities. This running time can be improved by more recent planar shortest path algorithms [63, 42, 84]; in particular, the recent algorithm of Mozes and Wulff-Nilsen [95] implies a running time of $O(n \log^2 n \log C / \log \log n)$. Miller and Naor [92] generalized Johnson and Venkatesan's algorithm to planar (single-commodity) flow networks with multiple sources and sinks. Returning to the classical augmenting path technique, Weihe [116, 115] described a planar maximum-flow algorithm that runs in $O(n \log n)$ time, provided the input graph satisfies a certain connectivity condition. Finally, Borradaile and Klein [11, 13] described the first $O(n \log n)$-time algorithm to find maximum flows in arbitrary directed planar graphs. Erickson [38] simplified the presentation and analysis of Borradaile and Klein's algorithm by reformulating it in terms of parametric shortest paths.

Generalizations of planar graphs. Surprisingly little is known about the complexity of flow algorithms for generalizations of planar graphs. A recent algorithm of Hochstein and Wiehe [65] computes a maximum flow in a planar graph with k additional edges in $O(k^3 n \log n)$ time, using a clever simulation of Goldberg and Tarjan's push-relabel algorithm [49]. Another related result is the algorithm of Hagerup *et al.* [58] to compute maximum flows in graphs of constant treewidth in $O(n)$ time.

To our knowledge, the only prior result that applies to graphs of positive genus, but not to arbitrary sparse graphs, is an algorithm of Imai and Iwano [70] that computes minimum-cost flows in graphs with small balanced separators, using a combination of nested dissection [86, 98], interior-point methods [113], and fast matrix multiplication. Their algorithm can be adapted to compute maximum flows in any graph of constant genus in time $O(n^{1.595} \log C)$, where C is the sum of the capacities. However, this is slower than more recent and more general algorithms [31, 48].

New results in this paper. Our key insight generalizes the relationship between flows and dual shortest paths in planar graphs first observed by Venkatesan [114] using a standard equivalence relation from algebraic topology called *homology*. We prove in Section 3.1 that given any flow f, one can find a feasible flow in the same *homology class* in near-linear time, by computing a single-source shortest path tree in the dual of the residual network G_f. Two flows are in the same homology class if their difference is the weighted sum of directed facial cycles. This observation allows us to optimize the homology class of the flow, rather than directly optimizing the flow itself. Instead of optimizing a vector of $O(n)$ flow values, our algorithm optimizes a vector of $2g + 1$ homology coefficients, subject to a much larger set of linear constraints; see Section 3.3.

We perform this optimization implicitly using two different techniques. In Section 3.4, we describe an adaptation of the central-cut ellipsoid method [56, 57] yields an algorithm that runs in $O(g^8 n \log^2 n \log^2 C)$ time for integer capacities that sum to C. The separation oracle for this algorithm is a new algorithm to compute shortest paths in surface-embedded graphs with positive and negative edge weights in $O(g^2 n \log^2 n)$ time, generalizing the recent planar shortest-path algorithm of Mozes and Wulff-Nilsen [95]. (The actual running times of these algorithms are somewhat more complicated than the bounds stated here; see the discussion just before Corollary 3.6. In particular, when the genus g is constant, our shortest-path algorithm runs in $O(n \log^2 n / \log\log n)$ time, and our maximum-flow algorithm runs in $O(n \log^2 n \log^2 C / \log\log n)$ time.) Alternatively, in Section 3.5, we use multidimensional parametric search [3, 26], together with a parallel shortest-path algorithm of Cohen [24], to obtain a combinatorial algorithm for graphs with arbitrary real capacities that runs in $g^{O(g)} n^{3/2}$ time. For any fixed genus g, both our algorithms improve the previous best time bounds by roughly a factor of $\sqrt{n}$.[1] We describe our algorithms first for undirected graphs embedded on orientable surfaces; some additional work is required to handle directed graphs (Section 3.6) and non-orientable surfaces (Section 3.7).

Following a strategy first suggested by Sullivan [109], we show that a dual formulation of our algorithm finds the minimum-cost circulation in the same homology class as a given circulation, in a graph with non-negative edge costs but no capacities, in roughly the same time as computing a maximum flow; see Section 4. If the given flow values and edge costs are integers, the resulting circulation is the minimum-cost *integer* circulation in the desired homology class. The minimum-cost circulation is always the weighted sum of at most $2g$ directed cycles. (In particular, the minimum-cost circulation in any planar graph is identically zero.) For more recent related results, see Dey, Hirani, and Krishnamoorthy [30] and Dunfield and Hirani [34].

We emphasize that all our algorithms require an explicit embedding as part of the input. Computing the minimum genus of an abstract graph is NP-hard [111]; moreover, no efficient algorithms are known that approximate the genus within a factor of $o(\sqrt{n})$ [22]. On the other hand, for any constant g, it is possible to compute either an embedding of a given graph on a surface of genus g, or an obstruction to such an embedding, in $O(n)$ time [77, 93].

New results on minimum cuts. In two related papers [20, 40], we describe the first algorithms to compute minimum (s, t)-cuts in near-linear time for any fixed genus, using different techniques than described in this paper. Both algorithms exploit the observation that finding a minimum-capacity (s, t)-cut in G is equivalent to finding the minimum-cost collection of cycles in the dual graph G^* (as defined in Section 2.6) in the same $\mathbb{Z}_2$-homology class as the boundary of s^*. (Two subgraphs are in the same $\mathbb{Z}_2$-homology class if their symmetric difference is the boundary of the union of a subset of the faces.) The first algorithm reduces the problem to several instances of the planar minimum-cut problem, each in a finite portion of the universal cover of the surface, using a technique introduced by Kutz [85] and generalized by Chambers *et al.* [18]. In combination with the recent planar minimum-cut algorithm of Italiano *et al.* [72], the running time of this algorithm is $g^{O(g)} n \log\log n$ time. The second algorithm solves the problem in $2^{O(n)} n \log n$ time by applying a multiple-source shortest path algorithm [16, 83]

[1]In the previous version of this paper [19], we incorrectly reported the running time of the first algorithm as $O(g^7 n \log^2 n \log^2 C)$ and the running time of the second algorithm as $n(g \log n)^{O(g)} = O(n \operatorname{polylog} n)$.

to a different covering space of the surface. (Except for a brief mention in Section 3.7, we do not use covering spaces in this paper; we refer the interested reader to our other papers for definitions and technical details [20, 40].) Essentially the same algorithms compute the minimum-weight subgraph in *every* $\mathbb{Z}_2$-homology class, in the same running time. Unlike the corresponding problem for circulations considered in this paper, computing the minimum-weight subgraph in an arbitrary $\mathbb{Z}_2$-homology class is NP-hard [20]. For related NP-hardness results, see Cabello *et al.* [17], Chambers *et al.* [18], Chen and Friedman [21], and Dunfield and Hirani [34].

2 Dramatis Personae

We begin by recalling several useful definitions from topological graph theory and algebraic topology. For more comprehensive background, we refer the interested reader to Gross and Tucker [55] or Mohar and Thommasen [94] for topological graph theory; and Hatcher [62] or Massey [88] for algebraic topology.

2.1 Surfaces

A ***surface*** (more formally, a *2-manifold*) is a Hausdorff topological space in which every point has an open neighborhood homeomorphic to $\mathbb{R}^2$. A ***cycle*** in a surface Σ is (the image of) a continuous map $\gamma\colon S^1 \to \Sigma$, where S^1 denotes the unit circle. A cycle is ***simple*** if this map is injective, and it is ***separating*** if its removal disconnects the underlying surface. The ***genus*** of a surface Σ is the maximum number of simple, disjoint, non-separating cycles $\gamma_1, \gamma_2, \ldots, \gamma_g$ in Σ; that is, $\gamma_i \cap \gamma_j = \varnothing$ for all i and j, and the space $\Sigma \setminus (\gamma_1 \cup \cdots \cup \gamma_g)$ is connected. A surface is ***non-orientable*** if it contains a subspace homeomorphic to the Möbius band (a one-sided surface with genus 1 and one boundary component), and ***orientable*** otherwise.

We consider only compact and connected surfaces in this paper; moreover, except in Section 3.7, we consider only orientable surfaces. Up to homeomorphism, for any non-negative integer g, there is exactly one orientable surface and exactly one non-orientable surface with genus g, constructed from the sphere by attaching g handles and g cross-caps, respectively. We also assume that $g = O(\sqrt{n})$, as our new algorithms improve existing results only when g is small.

2.2 Graphs and Embeddings

Let $G = (V, E)$ be an undirected graph. We define an associated directed graph $\vec{G} = (V, \vec{E})$ by replacing each undirected edge in E with an antisymmetric pair of directed edges. The graphs G and $\vec{G}$ are represented by the same adjacency matrix. Following Borradaile and Klein [11, 13], we refer to the directed edges in $\vec{E}$ as ***darts***. Each dart connects two (possibly equal) vertices, called its ***tail*** and its ***head***; we say that the dart ***leaves*** its tail and ***enters*** its head. Each dart $\vec{e}$ has a unique ***reversal***, denoted $\mathit{rev}(\vec{e})$ and defined by swapping its endpoints: $\mathit{head}(\mathit{rev}(\vec{e})) = \mathit{tail}(\vec{e})$ and $\mathit{tail}(\mathit{rev}(\vec{e})) = \mathit{head}(\vec{e})$. We will often write $\boldsymbol{u{\to}v}$ to denote a dart with tail u and head v; thus, $\mathit{rev}(u{\to}v) = v{\to}u$.

Informally, an ***embedding*** of a graph G on an orientable surface Σ is a drawing of the graph on Σ, such that vertices are mapped to distinct points and edges are mapped to non-crossing curves. A ***face*** of an embedding is a maximal connected subset of Σ that does not intersect the image of any edge or vertex. An embedding is ***cellular*** (or *2-cell* [94]) if every face is an open topological disk. Any cellular embedding can be represented combinatorially by a ***rotation system***, which is a permutation π of the darts of G, where $\pi(\vec{e})$ is the dart that appears immediately after $\vec{e}$ in the counterclockwise[2] ordering of darts leaving $tail(\vec{e})$.

Suppose $G = (V, E)$ is a simple n-vertex graph cellularly embedded on an orientable surface of genus g, and F is the set of faces of the embedding. Euler's formula $|V| - |E| + |F| = 2 - 2g$ implies that G has at most $3n - 6 + 6g$ edges and at most $2n - 4 + 4g$ faces, with equality if every face of the embedding is a triangle. Our assumption that $g = O(\sqrt{n})$ implies that the overall complexity of any embedding is $O(n)$.

Every dart in a cellularly embedded graph G separates two (possibly equal) faces, called the ***left shore*** and ***right shore***.[3] Reversing any dart swaps its shores: $left(rev(\vec{e})) = right(\vec{e})$ and $right(rev(\vec{e})) = left(\vec{e})$. We sometimes write $\boldsymbol{f\uparrow g}$ to denote a dart whose left shore is f and whose right shore is g; thus, $rev(f\uparrow g) = g\uparrow f$. See Figure 1.

2.3 Chains, Circulations, and Flows

Let $G = (V, E)$ be an undirected graph cellularly embedded on a surface Σ, and let F denote the set of faces of the embedding. A ***k-chain*** is a function that assigns a real weight to all cells of dimension k.[4] Thus, a ***0-chain*** is a function $\omega\colon V \to \mathbb{R}$; a ***1-chain*** is a function $\phi\colon E \to \mathbb{R}$; and a ***2-chain*** is a function $\alpha\colon F \to \mathbb{R}$.

It is useful to think of each 1-chain as assigning both an orientation and a *non-negative* value to each edge in G. We implicitly extend any 1-chain to a function on the darts of G; for each edge uv, we arbitrarily choose one of its darts $u{\to}v$ and define $\phi(u{\to}v) = \phi(uv)$ and $\phi(v{\to}u) = -\phi(uv)$.

The ***boundary*** of a 1-chain ϕ is the 0-chain $\boldsymbol{\partial\phi}\colon V \to \mathbb{R}$ defined as follows:

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

A ***circulation*** is a 1-chain ϕ such that $\partial\phi(v) = 0$ for every vertex $v \in V$; the equation $\partial\phi(v) = 0$ is often called the ***conservation constraint*** at v; intuitively, the total flow into v equals the total flow out of v. For any two vertices s and t, an ***(s, t)-flow*** (or just a *flow* if s and t are fixed) is a 1-chain ϕ such that $\partial\phi(v) = 0$ for every vertex $v \in V \setminus \{s, t\}$. The ***value*** $\boldsymbol{|\phi|}$ of a flow ϕ is $\partial\phi(t) = -\partial\phi(s)$; a circulation is simply a flow with value 0.

The (first) ***chain space*** $\boldsymbol{C(G)}$ is the vector space of all 1-chains in G, which is isomorphic to $\mathbb{R}^{|E|}$; this is sometimes also called the *edge space*. The ***cycle space*** $\boldsymbol{Z(G)}$ is the vector space of all circulations in G, which $Z(G)$ is isomorphic to $\mathbb{R}^{|E|-|V|+1}$. The ***flow space*** $\boldsymbol{Z(G; s\,t)}$ is the vector space of all (s, t)-flows in G, which is isomorphic to $\mathbb{R}^{|E|-|V|+2}$. The cycle space is (redundantly) generated by the indicator functions of all simple directed cycles in $\vec{G}$, and the flow space is (redundantly) generated by the indicator functions of all directed walks from s to t in $\vec{G}$.

For notational convenience, we define $\phi(\lambda) = \sum_{\vec{e}\in\lambda} \phi(\vec{e})$ for any 1-chain ϕ and any set λ of darts.

2.4 Boundary Circulations and Homology

The boundary of a 2-chain $\alpha\colon F \to \mathbb{R}$ is the 1-chain $\partial\alpha\colon E \to \mathbb{R}$ defined by setting $\partial\alpha(\vec{e}) := \alpha(right(\vec{e})) - \alpha(left(\vec{e}))$. One can easily verify that the boundary of any 2-chain is a circulation. A ***boundary circulation*** is the boundary of some 2-chain. In planar graphs, every circulation is a boundary circulation, but this is

[2]We can define either orientation of the surface to be 'counterclockwise', but we must make the same choice at every vertex; a consistent choice is possible if and only if Σ is orientable.

[3]Again, the distinction between left and right shores is well-defined if and only if the underlying surface is orientable.

[4]Classically, k-chains are defined as formal weighted sums of oriented k-cells; we do not distinguish between this formal sum and the function assigning a coefficient to each k-cell.

not true for higher-genus embeddings. The ***boundary space*** $\boldsymbol{B(G)}$ is the vector space of all boundary circulations; $B(G)$ is a linear subspace of $Z(G)$, isomorphic to $\mathbb{R}^{|F|-1}$.

We say that two flows or circulations ϕ and ψ are ***homologous***, or in the same ***homology class***, if their difference $\phi-\psi$ is a boundary circulation. In particular, two flows in a planar graph are homologous if and only if they have the same value. The ***homology space*** $\boldsymbol{H(G)}$ is the vector space of all homology classes of circulations in G, which is isomorphic to $Z(G)/B(G) \cong \mathbb{R}^{|E|-|V|-|F|+2} = \mathbb{R}^{2g}$ by Euler's formula. Similarly, the ***(s,t)-flow homology space***, which we denote by $\boldsymbol{H(G;st)}$, is the vector space of all homology classes of (s,t)-flows in G, which is isomorphic to $Z(G;st)/B(G) \cong \mathbb{R}^{|E|-|V|-|F|+1} = \mathbb{R}^{2g+1}$.

2.5 Capacities and Residual Networks

Now fix a positive ***capacity*** function $c\colon E \to \mathbb{R}^+$. (In Section 3.6, we consider *directed* graphs, where the capacity function has the form $c\colon \vec{E} \to \mathbb{R}^+$.) A flow or circulation ϕ is ***feasible*** (with respect to c) if and only if $|\phi(e)| \le c(e)$ for every undirected edge $e \in E$. We emphasize that the flow value assigned to an edge may be negative, even if the flow is feasible; the sign of the flow on each edge indicates its direction. We emphasize that flow values may be negative.[5] The ***residual capacity*** function $c_\phi : \vec{E} \to \mathbb{R}$ is defined by setting $c_\phi(u{\to}v) = c(uv) - \phi(u{\to}v)$. The ***residual network*** $\boldsymbol{G_\phi}$ is just the graph $\vec{G}$ with darts weighted by the residual capacity function c_ϕ. Clearly, ϕ is feasible if and only if every dart in G_ϕ has non-negative residual capacity. Moreover, ϕ is a *maximum* flow if and only if there is no directed path in G_ϕ from s to t in which every dart has positive residual capacity. We emphasize that the functions c and c_ϕ are *not* 1-chains, because they are not skew-symmetric on the darts of G.

Again, for notational convenience, we extend the functions c and c_ϕ to any subset λ of darts by summation: $c(\lambda) = \sum_{\vec{e}\in\lambda} c(e)$ and $c_\phi(\lambda) = c(\lambda) - \phi(\lambda) = \sum_{\vec{e}\in\lambda} c_\phi(\vec{e})$.

2.6 Dual Graphs, Cocycles, and Cohomology

The ***dual graph*** $\boldsymbol{G^*}$ of an embedded graph G is the (multi-)graph whose vertices are the faces of G, where two faces are joined by a (dual) edge if and only if they are separated by an edge of G. Thus, every edge e in G has a corresponding dual edge in G^*, denoted $\boldsymbol{e^*}$.

For any face f of G, we let f^* denote the corresponding vertex of G^*. The dual graph G^* has a natural cellular embedding on Σ whose faces correspond exactly to vertices of G. For any vertex v of G, we let v^* denote the corresponding face of G^*. We orient the darts of G^* by defining $(u{\to}v)^* := u^*{\uparrow}v^*$ and $(f{\uparrow}g)^* := f^*{\to}g^*$. Duality is an involution—the dual of G^* is isomorphic to the original graph G. However, G and G^* use opposite orientations of the underlying surface Σ to distinguish left from right.

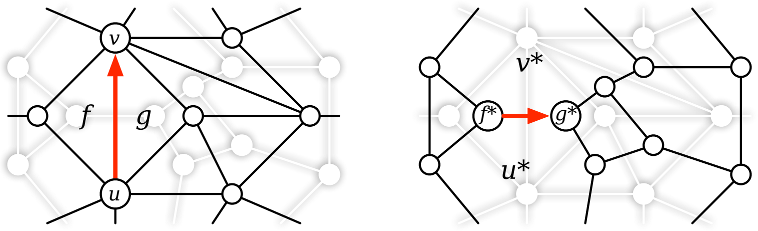

Figure 1. Graph duality. One dart $u{\to}v = f{\uparrow}g$ and its dual $f^*{\to}g^* = u^*{\uparrow}v^*$ are emphasized.

[5]Flows are often defined as functions from the darts to the *non-negative* reals, where without loss of generality, a directed edge and its reversal cannot both carry positive flow. While this classical formulation is more convenient for many algorithms, the equivalent skew-symmetric formulation is more suitable for our techniques.

When the graph G is fixed, we abuse notation by writing H^* to denote the subgraph of G^* containing the edges dual to the edges of a subgraph H of G. If the subgraph H is a cycle, we call H^* a ***cocycle***. The bijection between the edges of G and the edges of G^* extends to a bijection between 1-chains in G and in G^*. A ***cocirculation*** is a 1-chain whose dual is a circulation in G^*; a ***coboundary*** is a 1-chain whose dual is a boundary circulation in G^*. Two cocirculations are ***cohomologous***, or in the same ***cohomology class***, if their difference is a coboundary, or equivalently, if their dual circulations are homologous.

Many of the terms defined in this section are known by other names in other research communities. Algebraic topologists [62, 88] will immediately recognize circulations as 1-*cycles*, boundary circulations as 1-*boundaries*, $H(G)$ as the *first homology group* $H_1(\Sigma)$, and $H(G;st)$ as the *relative homology* group $H_1(\Sigma,\{s,t\})$, all with real coefficients. Discrete differential geometers [10, 29, 64] will recognize 1-chains as *discrete 1-forms*, the boundary operator as the adjoint of the *discrete exterior derivative*, circulations as duals of *closed* 1-forms, boundary circulations as duals of *exact* 1-forms, $H(G)$ as both the *first cohomology group* $H^1(\Sigma)$ and the space of *harmonic* 1-forms, $H(G;st)$ as the cohomology group $H^1(\Sigma\setminus\{s,t\})$, and combinatorial duality as a variant of the *Hodge star* operator.

3 Homology Flows

Throughout this section, we fix an undirected graph $G=(V,E)$, a cellular embedding of G on an orientable surface Σ of genus g, a capacity function $c\colon E\to\mathbb{R}^+$, and two vertices s and t. (We extend our results to directed graphs and non-orientable surfaces in Sections 3.6 and 3.7.)

3.1 Homologous Feasible Flows

More than 25 years ago, Venkatesan [114] observed that for any planar graph G, a feasible (s,t)-flow with a given value can be computed, if such a flow exists, by solving a single-source shortest path problem in a dual planar graph G^* with both positive and negative edge lengths. Similar approaches were proposed by Johnson and Venkatesan [74], Hassin and Johnson [61], Khuller *et al.* [80], and Miller and Naor [92]. The following lemma directly generalizes Venkatesan's observation to flow networks of higher genus.

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

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

Lemma 3.1. *There is a feasible (s,t)-flow in G homologous to a given (s,t)-flow ϕ if and only if the dual residual network G^*_ϕ contains no negative-cost cycles.*

Proof: Let λ^* be an arbitrary directed cycle in G^*_ϕ, and let λ denote the corresponding directed cocycle in $\vec{G}$. The total cost of λ^* is the difference between the total capacity of λ and the total flow through λ:

$$c_\phi(\lambda^*) \;=\; c(\lambda)-\phi(\lambda) \;=\; \sum_{\vec{e}\in\lambda}c(e)-\sum_{\vec{e}\in\lambda}\phi(\vec{e}).$$

For any 2-chain $\alpha\colon F\to\mathbb{R}$, we have

$$\sum_{\vec{e}\in\lambda}\partial\alpha(\vec{e}) \;=\sum_{f\uparrow g\in\lambda}(\alpha(g)-\alpha(f)) = \sum_{f^*\to g^*\in\lambda^*}(\alpha(g)-\alpha(f))=0.$$

(The last equality follows from the fact that λ^* is a cycle.) Thus, for any flow ψ homologous to ϕ, we have $\psi(\lambda) = \phi(\lambda)$, which immediately implies that $c_\psi(\lambda^*) = c_\phi(\lambda^*)$.

If the cycle λ^* has negative cost, then for any flow ψ homologous to ϕ, we have $c_\psi(\lambda^*) = c_\phi(\lambda^*) < 0$. It follows immediately that $c_\psi(\vec{e}) < 0$ for at least one dart $\vec{e}$ in λ; in other words, ψ is infeasible.

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

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

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

3.2 Shortest Paths with Negative Edges

Lemma 3.1 and its proof immediately imply an algorithm to find a feasible flow in a given homology class, if one exists, by directly applying any single-source shortest-path algorithm for embedded directed graphs with both positive- and negative-weight edges. The next two theorems describe the best parallel and serial algorithms known at present.

Theorem 3.2. *After $O(n)$ preprocessing time, given an (s,t)-flow ϕ in G, we can either find a feasible (s,t)-flow homologous with ϕ, or determine correctly that no homologous feasible flow exists, in $O(\log^3 n)$ time and $O(g^{3/2}n^{3/2})$ work on a EREW PRAM.*

Proof: The theorem follows immediately from the parallel shortest-path algorithm of Cohen [24]. The running time follows from the observation that any n-vertex graph of genus g can be separated into planar subgraphs, each with at most $2n/3$ vertices, by removing $O(\sqrt{gn})$ edges [32, 47, 69, 68]. Moreover, such a separator can be computed in $O(n)$ time [5, 37], after which the recursive separator decomposition of the resulting planar subgraphs can be constructed in $O(n)$ time [50]. □

The serial setting is not so straightforward. In the following theorem, we describe an algorithm that generalizes algorithms for planar graphs by Fakcharoenphol and Rao [42]; Klein, Mozes, and Weimann [84]; and Mozes and Wulff-Nilsen [95]. These algorithms rely on Miller's observation that any n-vertex planar graph contains a *simple cycle* separator of length $O(\sqrt{n})$, which can be computed in $O(n)$ time [91]. Because these algorithms require additional structure in the separator decomposition, we cannot directly substitute separator results for genus-g graphs.[6]

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

Theorem 3.3. *We can compute either a single-source shortest-path tree or a negative cycle in $\vec{G}$, with respect to any given edge weights $w\colon \vec{E} \to \mathbb{R}$, in $O(g^2 n \log^2 n)$ arithmetic operations.*

Proof: Our algorithm begins by cutting the undirected graph G along short non-separating cycles until the graph becomes planar. (Recall that a cycle on a surface is separating if its deletion disconnects the surface.) Cutting any cycle duplicates its vertices and edges and reduces the genus of the surface by 1.

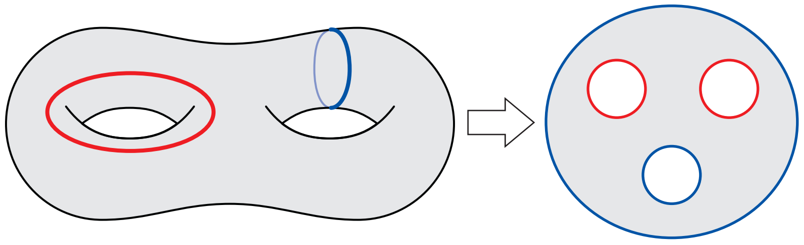

Figure 2. Cutting a surface of genus 2 along two non-separating cycles yields a planar surface with four boundary cycles.

We refer to the two copies of each cut cycle as *boundary cycles* and its vertices as *boundary vertices*; in the natural embedding of the resulting graph, every boundary cycle is a face. See Figure 2.

A theorem of Hutchinson [67] implies that every graph embedded on a surface of genus g contains a non-separating cycle with at most $O((\sqrt{n/g})\log g)$ vertices. For notational convenience, let $k = (\sqrt{n/g})\lg g$ in the rest of the analysis. The fastest algorithms known for computing *shortest* non-separating cycles [16] are too slow for our purposes.[7] Instead, we apply an algorithm of Erickson and Har-Peled [39] that computes a non-contractible cycle with at most twice the minimum number of edges, in $O(gn\log n)$ time. Thus, we obtain a planar graph P with exactly $2g$ boundary cycles, each with at most $O(k)$ vertices, in $\boldsymbol{O(g^2 n\log n)}$ time. The total number of vertices in P is at most $n + gk = n + O(\sqrt{gn}\log g) = O(n)$.

The remainder of our algorithm closely follows the recent planar shortest-path algorithm of Mozes and Wulff-Nilsen [95]. For the moment, we assume that there are no negative-weight cycles in $\vec{G}$; we describe the necessary modifications to detect negative cycles at the end of the proof.

Let $\vec{P}$ denote the symmetric directed planar graph corresponding to P, with edge weights inherited from $\vec{G}$, and let $dist_{\vec{P}}(u,v)$ denote the shortest-path distance in $\vec{P}$ from vertex u to vertex v. Fix an arbitrary boundary vertex r. We compute $dist_{\vec{P}}(r,v)$ for every vertex v in $\boldsymbol{O(n\log^2 n/\log\log n)}$ time, using the planar shortest-path algorithm of Mozes and Wulff-Nilsen [95]. We then define a *reduced weight* $\tilde{w}(u{\to}v) = dist_{\vec{P}}(r,u) + w(u{\to}v) - dist_{\vec{P}}(r,v)$ for every edge $u{\to}v$ of $\vec{P}$. It is well-known (and easy to prove) that the reduced weights are non-negative and define the same shortest paths as the original edge weights [73].

Next, we compute the distances between every pair of boundary vertices using Klein's multiple-source shortest path algorithm [83]. Klein's algorithm requires non-negative edge weights, so we call the algorithm using the reduced weights $\tilde{w}$. For each boundary cycle δ, Klein's algorithm constructs a data structure, in $O(n\log n)$ time, from which the shortest-path distance from any vertex of δ to any other vertex of $\vec{P}$ can be retrieved in $O(\log n)$ time. Given the shortest-path distance between any two vertices with respect to the reduced edge weights, we can compute the shortest-path distance with respect to the original edge weights in $O(1)$ time. Altogether, there are $O(g^2k^2)$ pairs of boundary vertices, so the total time for all $O(g)$ invocations of Klein's algorithm is $O(gn\log n + g^2k^2\log n) = \boldsymbol{O((g\log^2 g)(n\log n))}$.

The most subtle part of the algorithm computes shortest-path distances *in the original graph* $\vec{G}$ between every pair of boundary vertices. Let $\vec{H}$ be the complete directed graph over the boundary vertices of $\vec{P}$, where the weight any edge $u{\to}v$ is the shortest-path distance from u to v, with additional zero-weight edges between any pair of vertices arising from the same vertex of G. We compute all shortest-path distances in $\vec{H}$ using a modification of the Bellman-Ford algorithm described by Mozes and Wulff-Nilsen [95, Figure 4]. Instead of relaxing the edge in $\vec{H}$ individually in each Bellman-Ford iteration,

[6]...despite our claim to the contrary in the previous version of this paper [19].

[7]On the other hand, because this portion of the algorithm is independent of the edge weights, we could perform it only once in a preprocessing stage, using the exact $O(g^3 n\log n)$-time algorithm of Cabello and Chambers [16], without increasing the running time of our maximum-flow algorithm.

we consider each set of at most two boundary cycles separately. Klein *et al.* [84, Lemma 4] describe a method to relax all edges with both endpoints on the same boundary cycle in $O(k\alpha(k))$ time, where $\alpha(\cdot)$ is the inverse-Ackerman function, using an algorithm of Klawe and Kleitman [82] to find row-minima in totally-monotone triangular matrices. The algorithm of Klein *et al.* requires $O(k \log k)$ preprocessing time for each cycle. Mozes and Wulff-Nilsen [95, Lemma 4] describe an algorithm to relax all edges joining any two different boundary cycles in $O(k)$ time; this algorithm requires $O(k \log k)$ preprocessing time for each pair of cycles. We also directly relax each of the $O(gk)$ zero-weight edges in each iteration. Altogether, the modified Bellman-Ford algorithm requires $O(g^2 k \log k)$ preprocessing time, followed by $O(gk)$ Bellman-Ford iterations, each running in $O(g^2 k + gk\alpha(k))$ time. Thus, the overall running time of this phase of the algorithm $O(g^3 k^2 + g^2 k^2 \alpha(k)) = \boldsymbol{O((g^2 \log^2 g)n + (g \log^2 g)(n\alpha(n)))}$.

Let $dist_{\vec{G}}(u,v)$ denote the shortest-path distance in $\vec{G}$ from vertex u to vertex v. Following Klein *et al.* [84], let $\vec{G}'$ denote the graph obtained from $\vec{G}$ by removing all edges entering r and adding an edge $r{\to}u$ with weight $dist_{\vec{G}}(r,u)$ for every boundary vertex u; these distances were computed in the previous phase of the algorithm. Shortest-path distances from r are equal in $\vec{G}$ and $\vec{G}'$ [84, Lemma 6]. For any boundary node u, let $\Delta(u) = dist_{\vec{G}}(r,u) - dist_{\vec{P}}(r,u)$, and let $\Delta = \max_u \Delta(u)$. (Note that $\Delta < 0$.) We define a second set of reduced weights $w'(u{\to}v) = \phi(u) + w(u{\to}v) - \phi(v)$ for each edge $u{\to}v$ of $\vec{G}'$, where $\phi(r) = \Delta$ and $\phi(v) = dist_{\vec{P}}(r,v)$ for any vertex $v \neq r$. The reduced weights w' are non-negative [84, Lemma 7]. Thus, we can compute a shortest-path tree in $\vec{G}'$ rooted at r, using Dijkstra's algorithm, in $\boldsymbol{O(n \log n)}$ time. (We could also use the linear-time shortest-path algorithm of Henzinger *et al.* [63], but this would not improve our overall running time.)

To finish the algorithm, we compute the shortest-path tree rooted at r in the original graph $\vec{G}$ in $\boldsymbol{O(n \log n)}$ time using Dijkstra's algorithm with yet another reduced weight function $dist_{\vec{G}}(r,u) + w(u{\to}v) - dist_{\vec{G}}(r,v)$. (In fact, every edge in the shortest-path tree has reduced weight 0, so it can be constructed in $O(n)$ time by a simple breadth-first search.)

Finally, we consider the modifications required to return negative cycles when they exist. Negative cycles that contain at least one boundary vertex can be detected and returned by a standard extension of the Bellman-Ford algorithm. On the other hand, any negative cycle in $\vec{G}$ that avoids every boundary vertex appears as a negative cycle in $\vec{P}$ and thus can be detected and returned by the planar shortest-path algorithm of Mozes and Wulff-Nilsen [95]. Their algorithm uses a similar modified Bellman-Ford algorithm to combine the results of recursive subproblems; so in fact, any negative cycle is reported by their modified Bellman-Ford algorithm at some level of recursion. The necessary modifications increase the running time of the algorithm by only a constant factor. □

In the interest of readability, Theorem 3.3 gives a conservative upper bound on the running time of our shortest-path algorithm. The running time is more accurately bounded as

$$O(n \log^2 n / \log\log n + g^2 n \log n + g^2 n \log^2 g).$$

This bound is dominated by its first term whenever $g = O\big(\sqrt{\log n / \log\log n}\big)$ and dominated by its third term whenever $g = 2^{\Omega(\sqrt{\log n})}$. In particular, for any constant genus, the running time of our algorithm is dominated by the time to compute shortest paths in the planar graph $\vec{P}$; faster planar shortest-path algorithms would immediately improve our algorithm as well. Sharper results on the complexity of planarizing cycles would improve the second and third terms in our running time, but only slightly. For example, if one could find a planarizing set of $O(g)$ cycles with total complexity $O(\sqrt{gn})$ in $O(n)$ time, then the running time of our algorithm would drop to $O(n \log^2 n / \log\log n + gn \log n + g^2 n)$.

Corollary 3.4. *Given an (s,t)-flow ϕ in G, we can either find a feasible (s,t)-flow in G that is homologous with ϕ, or find a negative cycle in G^*_ϕ if no homologous feasible flow exists, using $O(g^2 n \log^2 n)$ arithmetic operations.*

3.3 Basic Flows and Optimization

Every (s,t)-flow can be expressed as a weighted sum of simple directed cycles and simple directed paths from s to t. Consequently, every homology class of (s,t)-flows is a weighted sum of homology classes of (s,t)-paths and cycles. It follows immediately that the flow homology space $Z(G,st) \cong \mathbb{R}^{2g+1}$ can be generated by the homology classes of $2g+1$ curves, each of which is a (s,t)-path or a cycle. We call such a collection of curves a ***flow homology basis***. Any flow homology basis includes at least one (s,t)-path; we call a flow homology basis *canonical* if it contains *exactly* one (s,t)-path and $2g$ cycles; these $2g$ cycles necessarily define a basis for the space $H(G)$ of homology classes of circulations.[8]

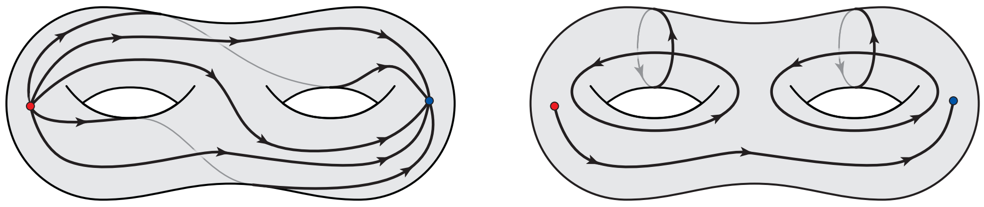

Figure 3. Two flow homology bases for a surface of genus 2; only the second basis is canonical.

Lemma 3.5. *We can compute a canonical flow homology basis for G in $O(gn)$ time.*

Proof: We begin by computing a tree-cotree decomposition [37]. Let T be an arbitrary spanning tree of G; let K^* be an arbitrary spanning tree of $(G \setminus T)^* = G^* \setminus T^*$; and finally, let $X = G \setminus (T \cup K)$. We refer to K as a *spanning cotree* of G, because the corresponding dual subgraph K^* is also a spanning tree of the dual graph G^*. Euler's formula implies that X contains exactly $2g$ edges; call them $e_1, e_2, \ldots, e_{2g}$.

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

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

Suppose to the contrary that for some index j, the cycle γ_j is homologous to $\sum_{i<j} a_i \gamma_i$ for some real coefficients a_i. Then the difference $\varphi := \gamma_j - \sum_{i<j} a_i \gamma_i$ is a boundary circulation, which has nonzero value only on edges of $T \cup X$. Let $\alpha \colon F \to \mathbb{R}$ be a 2-chain such that $\varphi = \partial\alpha$. For any cotree edge $e \in K$, we have $\varphi(e) = 0$. Thus, if a cotree edge separates two faces f and f', we must have $\alpha(f) = \alpha(f')$. Because K^* is a spanning tree for G^*, it follows that α assigns the same value to *every* face of G, which implies that φ is identically zero. On the other hand, γ_j contains an edge e_j that does not lie on any other cycle γ_i, so $\varphi(e_j) = \gamma_j(e_j) = \pm 1$. We have a contradiction.

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

Fix a canonical flow homology basis $p_0, \gamma_1, \ldots, \gamma_{2g}$ for G. A ***basic flow*** is any flow ϕ of the form $\phi_0 \cdot p_0 + \sum_{i=1}^{2g} \phi_i \cdot \gamma_i$ for some coefficients $\phi_0, \phi_1, \ldots, \phi_{2g}$. Specifically, we have $\phi_0 = |\phi|$ and $\phi_i = \phi(e_i)$ for each index i. Equivalently, a flow ϕ is basic if and only if $\phi(e) = 0$ for every cotree edge $e \in K$. (Given

[8]In the previous version of this paper [19], we defined a flow homology basis to be a set of $2g+1$ *walks* from s to t. The current formulation simplifies our analysis slightly, in part because we can assume that the generating curves are simple.

a flow ϕ that avoids every edge in K, subtracting the basic flow with coefficients $|\phi|, \phi(e_1), \dots, \phi(e_{2g})$ leaves a circulation that avoids every edge in $K \cup X = G \setminus T$ and is therefore identically zero.) Every flow in G is homologous to exactly one basic flow.

Corollary 3.6. *Given the coefficients $\phi_0, \phi_1, \dots, \phi_{2g}$ of a basic flow ϕ, we can either find a feasible (s,t)-flow in G that is homologous with ϕ, or find a negative cycle in G^*_ϕ if no homologous feasible flow exists, using $O(g^2 n \log^2 n)$ arithmetic operations.*

Corollary 3.7. *After $O(n)$ preprocessing time, given the coefficients $\phi_0, \phi_1, \dots, \phi_{2g}$ of a basic flow ϕ, we can either find a feasible (s,t)-flow homologous with ϕ, or determine correctly that no homologous feasible flow exists, in $O(\log^3 n)$ time and $O(g^{3/2}n^{3/2})$ work on a EREW PRAM.*

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

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

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

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

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

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

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

3.4 The Ellipsoid Method

We now transform our decision procedure into an optimization algorithm using the classical *central-cut ellipsoid method*. The ellipsoid method was first proposed independently by Yudin and Nemirovsky [118, 119] and Shor [107] as a convex programming algorithm. Khachiyan [78, 79] adapted the ellipsoid method to give the first polynomial-time algorithm for linear programming; see also [9, 45]. Khachiyan's algorithm was further adapted to solve *implicit* linear programming problems by Grötschel, Lovász, and Schrijver [56, 57]. Here we give only a brief sketch of the method, with just enough details to complete the analysis; we refer the interested reader to Grötschel *et al.* [57] for further details.

3.4.1 A Brief Sketch

In its simplest form, the ellipsoid method can be used to solve any linear programming problem whose constraints are represented implicitly by an ellipsoid $\mathcal{E}_0$ containing the feasible region Φ, a known lower bound on the volume of Φ, and a black-box subroutine called a *(strong) separation oracle*. Let d denote the number of variables in the linear program. Given a point $x \in \mathbb{R}^d$, the separation oracle either returns a constraint that is violated by x or asserts correctly that x is feasible. For purposes of analysis, we assume that the separation oracle is *fully combinatorial*; that is, every branch in the algorithm is based on the sign of an affine combination of the coordinates of x. Equivalently, we assume that the separation oracle can be modeled by a family of *linear decision trees* [33, 117]. Let T_s denote the number of arithmetic operations (additions, subtractions, scalar multiplications, and comparisons) executed by a single call to the separation oracle.

Let ϕ_{OPT} denote the optimum vertex of Φ; if necessary we perturb the objective function $\tilde{z}$ so that ϕ_{OPT} is unique. The ellipsoid algorithm maintains an ellipsoid $\mathcal{E}$ guaranteed to contain ϕ_{OPT}, starting with the bounding ellipsoid $\mathcal{E}_0$. At each iteration, we invoke the separation oracle to determine whether the centroid x of $\mathcal{E}$ is feasible. If x is infeasible, the oracle returns a violated constraint in the linear program. If x is feasible, we add the artificial constraint $\langle \tilde{z}, \phi \rangle \geq \langle \tilde{z}, x \rangle$. In either case, we obtain a halfspace h that contains ϕ_{OPT}. We then compute (a close approximation of) the smallest ellipsoid containing $\mathcal{E} \cap h$, and let this be the new ellipsoid $\mathcal{E}$. Each iteration reduces the volume of $\mathcal{E}$ by a factor of $e^{1/O(d)}$.

Suppose all the vertices of Φ have integer coordinates; we show in Lemma 3.12 that this assumption indeed holds for our problem. Let $\Phi_\varepsilon := \Phi \cap h_\varepsilon$, where h_ε is the halfspace $\langle \tilde{z}, \phi \rangle \geq \langle \tilde{z}, \phi_{OPT} \rangle - \varepsilon$ and ε is chosen so that Φ_ε lies inside a ball of radius $1/4$. (The specific value of ε depends on the volume lower bound for Φ.) The iterations stop when the volume of $\mathcal{E}$ is smaller than the volume of Φ_ε. We find the optimal vertex ϕ_{OPT} by rounding the last feasible point found, which is guaranteed to lie inside Φ_ε, to the integer grid.

The running time of the ellipsoid algorithm depends on the volumes of the initial and final ellipsoids and the running time of the separation oracle as follows. The method requires $N := O(d \log \Delta)$ iterations, where $\Delta := (\operatorname{vol} \mathcal{E}_0 / \operatorname{vol} \Phi_\varepsilon)$. Each iteration requires T_s arithmetic operations by the separation oracle, plus $O(d^2)$ arithmetic operations (which include multiplications, divisions, and square roots) to compute the new ellipsoid. Grötschel *et al.* prove that to maintain sufficient precision in the kth iteration, it suffices to round all numbers to $O(k)$ bits [56, 57]. Thus, the overall running time of our algorithm is (crudely [6, 103]) at most $O(N(T_s N + d^2 N \log^2 N)) = \boldsymbol{O(T_s d^2 \log^2 \Delta + d^4 \log^2 \Delta \log^2(d \log \Delta))}$.

3.4.2 The Flow Homology Polytope

Let Φ denote the polytope of feasible flow homology classes, that is, the feasible region of the linear program (LP). We now establish several facts about this polytope that are necessary to analyze of our application of the ellipsoid method; some of these properties will also be useful later in the paper. Recall that $G = (V, E)$ is an undirected graph embedded on an orientable surface Σ of genus g, with an associated positive integer capacity function $c \colon E \to \mathbb{Z}^+$. Let C denote the sum of the edge capacities $\sum_{e \in E} c(e)$. We assume that $C = O(2^n)$, since otherwise our new algorithms do not improve on existing results.

Lemma 3.8. *Φ lies inside the hypercube $[-C, C]^{2g+1}$.*

Proof: The value of any basic flow ϕ is the coefficient ϕ_0, so any feasible basic flow trivially satisfies the inequality $|\phi_0| \leq C$.

Recall from the proof of Lemma 3.5 that the flow homology basis is defined in terms of a tree-cotree decomposition (T, K, X), where T is a spanning tree, K is a spanning cotree, and $X = G \setminus (T \cup K)$. Fix an index i, let $G_i = K \cup \{e_i\}$, and let Σ_i denote the surface obtained by deleting the subgraph $T \cup X \setminus \{e_i\} = G \setminus G_i$ from Σ. The dual graph G_i^* is connected and contains exactly one cycle, which implies that the surface Σ_i is homeomorphic to an annulus (the sphere minus two disks).

For each index i, let λ_i be the cocycle in G dual to the unique cycle in G_i^*. The cocycle λ_i shares exactly one edge e_i with the corresponding cycle γ_i in the flow homology basis, which implies that $\gamma_i(\lambda_i) = \pm 1$. On the other hand, for any index $j \neq i$, the cocycle λ_i does not share any edges with the basis cycle γ_j, which implies that $\gamma_j(\lambda_i) = 0$. Similarly, λ_i avoids every edge in T, which implies that $p_0(\lambda_i) = 0$. Thus, for any basic flow ϕ, we have $\phi_i = \pm\phi(\lambda_i)$. (In topological terms, the cocycles $\lambda_1, \dots, \lambda_{2g}$ constitute a *cohomology basis*.) We conclude that every *feasible* basic flow ϕ satisfies the inequality $|\phi_i| = |\phi(\lambda_i)| \leq c(\lambda_i) \leq C$. □

Corollary 3.9. *Φ lies inside a ball of radius $C\sqrt{2g+1}$ centered at the origin.*

Corollary 3.9 immediately implies that we can use the ball of radius $C\sqrt{2g+1}$ centered at the origin as the initial ellipsoid $\mathcal{E}_0$ in our optimization algorithm.

Lemma 3.10. *Φ contains a ball of radius $1/\sqrt{2g+1}$ centered at the origin.*

Proof: The graph G is undirected, and every edge has capacity at least 1, so each basis path p_i can carry at least one unit of flow in either direction. Thus, every unit basis vector $(0, \dots, \pm 1, \dots, 0)$ is feasible. The convex hull of these $4g + 2$ unit vectors is contained in Φ. This cross-polytope contains the ball of radius $1/\sqrt{2g+1}$ centered at the origin. □

Lemma 3.11. *Every integer flow in G is homologous to an integer basic flow.*

Proof: Let ϕ be an integer flow in G, and let (T, K, X) be the tree-cotree decomposition of G used to define the flow homology basis. Choose an arbitrary dual vertex r^* and orient every edge of the dual spanning tree K^* away from r^*. For each face f of G, let $\alpha(f)$ denote the shortest path distance from r^* to f^* in K^*, where the cost of each dual dart $\vec{e}^*$ is its corresponding flow value $\phi(\vec{e})$. The flow $\psi = \phi - \partial\alpha$ is an integer basic flow homologous to ϕ. □

Lemma 3.12. *The vertices of Φ have integer coordinates.*

Proof: Let ψ and ψ' be flows in G, and ϕ and ϕ' be the basic flows homologous to ψ and ψ', respectively. Then for any real scalar a, the flows $a\psi + \psi'$ and $a\phi + \phi'$ are also homologous. It follows that the function that maps any flow in G to its unique homologous basic flow is a *linear* map from the chain space $C(G) \cong \mathbb{R}^{|E|}$ to the flow homology space $H(G; st) \cong \mathbb{R}^{2g+1}$.

Now let $\mathcal{F} \subset \mathbb{R}^E$ denote the polytope of feasible flows in G. The polytope Φ is the image of $\mathcal{F}$ under the projection from flows to homologous basic flows. Recall that a matrix is *totally unimodular* if every square submatrix has determinant 0, 1, or -1. It is well known[9] that the constraint matrix of any network-flow linear program is totally unimodular [105, Theorem 13.9], which implies by Cramer's rule that the vertices of $\mathcal{F}$ have integer coordinates [104, Theorem 19.1]. Lemma 3.11 implies that every vertex of $\mathcal{F}$ projects to an integer point in $\mathbb{R}^{2g+1}$. The polytope Φ is the convex hull of these projected integer points. □

We emphasize here that the constraint matrix that defines Φ contains entries with absolute value larger than 1 and thus is *not* totally unimodular.

[9]Schrijver [105] observes that this result follows directly from a theorem of Poincaré [101, Section 6]. Poincaré described a condition on the boundary matrices of simplicial complexes that implies total unimodularity, which in turn implies that the corresponding homology groups are torsion-free. Incidence matrices of directed graphs satisfy Poincaré's condition.

3.4.3 Finding the Optimal Flow Homology Class

We now apply the ellipsoid method to our implicit linear program (LP). We clearly have $d = 2g + 1$, and Corollary 3.6 gives us $T_s = O(g^2 n \log^2 n)$. To complete our analysis, we need to fill in a few details of the algorithm and derive an upper bound on the volume ratio Δ. Our presentation closely follows Grötschel, Lovász, and Schrijver [56, 57].

Let $\boldsymbol{z} := (\mathbf{1}, \mathbf{0}, \ldots, \mathbf{0})$ denote the objective function for the linear program (LP). Our algorithm uses a perturbed objective function $\tilde{\boldsymbol{z}} := (\boldsymbol{Q}^{2g}, \boldsymbol{Q}^{2g-1}, \ldots, \boldsymbol{Q}, \mathbf{1})$, where $\boldsymbol{Q} := \mathbf{2}\boldsymbol{C} + \mathbf{2}$. We also set $\boldsymbol{\varepsilon} := \mathbf{1}/(\mathbf{8}\boldsymbol{C}\sqrt{\mathbf{2}\boldsymbol{g} + \mathbf{1}})$. Recall that h_ε denotes the halfspace $\langle \tilde{c}, \phi \rangle \geq \langle \tilde{z}, \phi_{OPT} \rangle - \varepsilon$ and $\Phi_\varepsilon := \Phi \cap h_\varepsilon$.

Lemma 3.13. *Φ has a unique optimal vertex ϕ_{OPT} with respect to $\tilde{z}$, and this vertex is also optimal with respect to z.*

Proof: Let ϕ_{OPT} be any vertex of Φ such that $\langle \tilde{z}, \phi_{OPT} \rangle$ is maximized. Let φ be any other vertex of Φ, and let $v = \phi_{OPT} - \varphi$. Lemmas 3.8 and 3.12 imply that v is a non-zero integer vector, each of whose components has absolute value at most $2C$. At least one coordinate of v must be non-zero; let v_i be the non-zero coordinate with minimum index. The definition of $\tilde{z}$ implies that $\langle \tilde{z}, v \rangle = \sum_{j=0}^{2g} Q^{2g-j} v_j$. The ith term of this sum has absolute value at least Q^{2g-i}, and the sum of the other terms has absolute value at most $2C\,Q^{2g-i}/(Q-1) < Q^{2g-i}$. It follows that $\langle \tilde{z}, v \rangle \neq 0$. We conclude that that ϕ_{OPT} is the *unique* optimal vertex for objective $\tilde{z}$.

The optimality of ϕ_{OPT} implies that $\langle \tilde{z}, v \rangle > 0$. The sign of the sum $\sum_{j=0}^{2g} Q^{2g-j} v_j$ is determined by its most significant non-zero term, so its first term $Q^{2g} v_0$ must be non-negative. Thus, $\langle z, v \rangle = v_0 \geq 0$. We conclude that ϕ_{OPT} is also an optimal vertex for the original objective vector z. □

Lemma 3.14. *Φ_ε lies inside a ball of radius $1/4$.*

Proof: Because the vertices of Φ are integral (Lemma 3.12) and $\varepsilon \leq 1$, every vertex of Φ_ε except ϕ_{OPT} lies on the boundary of the halfspace h_ε. Let h_1 denote the halfspace $\langle \tilde{z}, \phi \rangle \geq \langle \tilde{z}, \phi_{OPT} \rangle - 1$ and let $\Phi_1 := \Phi \cap h_1$. Again, every vertex of Φ_1 except ϕ_{OPT} lies on the boundary of h_1. Thus, Φ_ε can be obtained by scaling Φ_1 by a factor of ε around ϕ_{OPT}. Corollary 3.9 implies that $\Phi_1 \subseteq \Phi$ lies inside a ball of radius $2C\sqrt{2g+1}$. Thus, Φ_ε lies inside a ball of radius $2\varepsilon C\sqrt{2g+1} = 1/4$, as required. □

Lemma 3.15. *Φ_ε has volume $C^{-O(g^2)}$.*

Proof: Consider the cone X whose apex is ϕ_{OPT} and whose base is the $2g$-dimensional ball of radius $1/\sqrt{2g+1}$ centered at the origin on the hyperplane $\langle \tilde{z}, \phi \rangle = 0$. Lemma 3.10 implies that $X \subseteq \Phi$. The volume of X is

$$\frac{\pi^g}{g!} \cdot \left(\frac{1}{\sqrt{2g+1}} \right)^{2g+1} \cdot \frac{\langle \tilde{z}, \phi_{OPT} \rangle}{\|\tilde{z}\|} = \frac{1}{g^{O(g)}} \cdot \frac{\langle \tilde{z}, \phi_{OPT} \rangle}{\|\tilde{z}\|}.$$

(Recall that $\pi^g/g!$ is the volume of the $2g$-dimensional unit ball.) The volume of $X \cap h_\varepsilon \subseteq \Phi_\varepsilon$ is therefore

$$\frac{1}{g^{O(g)}} \cdot \frac{\langle \tilde{z}, \phi_{OPT} \rangle}{\|\tilde{z}\|} \cdot \left(\frac{\varepsilon}{\langle \tilde{z}, \phi_{OPT} \rangle} \right)^{2g+1} = \frac{1}{g^{O(g)}} \cdot \frac{1}{\|\tilde{z}\| \, \langle \tilde{z}, \phi_{OPT} \rangle^{2g} C^{2g+1}},$$

by the definition of ε. Corollary 3.9 implies that $\|\phi_{OPT}\| \leq C\sqrt{2g+1}$, and the definition of $\tilde{z}$ gives us $\|\tilde{z}\| = O(g\,C^{2g})$. It follows that $\langle \tilde{z}, \phi_{OPT} \rangle \leq \|\tilde{z}\| \cdot \|\phi_{OPT}\| = O(g^{3/2} C^{2g+1})$. The lemma now follows by straightforward algebra. □

Recall that the generic ellipsoid algorithm runs in $O(T_s d^2 \log^2 \Delta + d^4 \log^2 \Delta \log^2(d \log \Delta))$ time. In our application of the method, we have $d = 2g+1$; Corollary 3.9 and Lemma 3.15 imply that $\Delta = \operatorname{vol}\mathcal{E}_0 / \operatorname{vol}\Phi_e = C^{O(g)}/C^{-O(g^2)} = C^{O(g^2)}$; and Corollary 3.6 implies that $T_s = O(g^2 n \log^2 n)$. We conclude that our maximum-flow algorithm runs in $O(g^8 n \log^2 n \log^2 C)$ time. (The $O(g^8 \log^2 C \log^2 \log C)$ term in the running time is dominated, because $C = O(2^n)$.)

Theorem 3.16. *Given an undirected graph $G = (V,E)$ embedded on an orientable surface of genus g, a positive integer capacity function $c\colon E \to \mathbb{Z}^+$, and two vertices $s,t \in V$, a maximum (s,t)-flow in G can be computed in time $O(g^8 n \log^2 n \log^2 C)$, where C is the sum of the edge capacities.*

Other variants of this algorithm, with different dependencies on g and $\log C$, are also possible. For example, a binary search over the possible flow values reduces the maximum-flow problem to $O(\log C)$ instances of finding a feasible flow with a given value. Finding a flow with any particular value can be reduced to a $2g$-dimensional LP-feasibility problem using our techniques, which we can solve in $O(g^6 n \log^2 n \log^2 C)$ time. This algorithm uses fewer ellipsoid steps in each iterations, because we can stop as soon as the volume of $\mathcal{E}$ falls below $1/(2g)!$. Finding the maximum flow value is now relatively straightforward, but there are some additional technical difficulties in finding a flow with that value, because the resulting homology polytope is not full-dimensional. We can resolve these difficulties using a perturbation technique similar to Section 3.6 below. The resulting algorithm runs in $O(g^6 n \log^2 n \log^3 C)$ time. We omit further details.

3.5 Multidimensional Parametric Search

We now describe a fully combinatorial algorithm that works for arbitrary capacities, but whose running time is worse, even for fixed genus. Our algorithm uses the *multidimensional parametric search* paradigm, a generalization of Megiddo's parametric search technique [89] developed independently by Cohen and Megiddo [23, 25, 26], Norton, Plotkin, and Tardos [96], and Aneja and Kabadi [7], and extended by several other authors [1, 2, 3, 27, 75, 76, 112]. Specifically, we use the version of the technique described by Agarwal, Sharir, and Toledo [3].

The multidimensional parametric search technique requires two black-box algorithms for the decision problem, one serial and the other parallel. The parallel algorithm must be fully combinatorial, meaning (as in the previous section) that every branch is based on the sign of an affine combination of the input parameters. Unlike the ellipsoid method described in the previous section, multidimensional parametric search requires only a membership oracle; thus our parallel algorithm (Corollary 3.7) need not actually return a negative-length cycle if one exists.

Let T_s denote the running time of the serial decision algorithm, T_p the running time of the parallel decision algorithm, P the number of processors used by the parallel decision algorithm, and d the number of parameters to be resolved. The running time of the resulting optimization algorithm is $d^{O(d)}\big(T_p P + (T_p \log P)^d T_s\big)$.[10] In our application of the method, we have $d = 2g+1$; Corollary 3.6 gives us $T_s = O(g^2 n \log^2 n)$; and Corollary 3.7 implies $T_p = O(\log^3 n)$ and $P = O(n^{3/2}/\log^3 n)$. Thus, the overall running time of our algorithm is $g^{O(g)} n^{3/2}$.

Theorem 3.17. *Given a graph $G = (V,E)$ embedded on an orientable surface of genus g, a positive capacity function $c\colon E \to \mathbb{R}^+$, and two vertices $s,t \in V$, a maximum (s,t)-flow in G can be computed in $g^{O(g)} n^{3/2}$ time.*

[10] Agarwal *et al.* [3] assume only the existence of a parallel decision algorithm, and thus report the running time as $d^{O(d)}(T_p \log P)^d T_p P$; the running time we report here follows by using a separate serial algorithm at the lowest level of recursion. Agarwal and Sharir [2] report the running time as $d^{O(d)} T_s (T_p \log P)^d$; however, this time bound is incorrect if T_s is significantly smaller than $T_p P$.

3.6 Directed Graphs

So far we have restricted our attention to flows in undirected graphs; however, our results can be extended to directed graphs with only a little more effort. We assume without loss of generality that the input graph is the symmetric directed graph $\vec{G}$ associated with some undirected graph G and that the embedding of $\vec{G}$ is induced by a cellular embedding of G. However, the capacity function is not necessarily symmetric; in particular, some edges in $\vec{G}$ may have zero capacity, even though their reversals do not.

Because the residual graph G_ϕ is already a directed graph, all the results in Sections 3.1 and 3.3 apply without modification to directed graphs. The multidimensional parametric search algorithm described by Theorem 3.17 also applies immediately to the directed setting. However, our proof of Theorem 3.16 does not generalize to directed graphs, because the flow homology polytope Φ may have zero volume. Specifically, Lemmas 3.10 and 3.15 no longer hold; all the other lemmas generalize immediately. Fortunately, we can work around this problem by adapting a standard perturbation technique proposed by Khachiyan [78, 79] for arbitrary linear programs. Our description closely follows Papadimitriou and Steiglitz [99, Lemma 8.7].

Let $c\colon \vec{E} \to \mathbb{N}$ be the input capacity function, and C denote the sum of the edge capacities, as before. Let $m := |E|$, and observe that $m = O(C)$. We define a new capacity function $c'\colon \vec{E} \to \mathbb{Z}^+$ by setting $\boldsymbol{c'(\vec{e}) := c(\vec{e}) + 1/m^3}$ for every directed edge $\vec{e}$. After scaling up by a factor of m^3, this capacity function satisfies Lemma 3.10; the rest of the proof of Theorem 3.16 goes through unchanged, except with $C' = 2m^3C + m$ in place of C. Thus, we can compute a maximum flow ϕ'_{OPT} for the new capacity function in $O(g^8 n \log^2 n \log^2 C') = O(g^8 n \log^2 n \log^2 C)$ time using our earlier algorithm. Finally, we round ϕ'_{OPT} to the nearest integer flow and return the result as the solution to our original flow problem. The overall running time of the algorithm is still $O(g^8 n \log^2 n \log^2 C)$.

It remains to prove that this rounding algorithm actually works. Let $\mathcal{F} \subset \mathbb{R}^m$ denote the polytope of feasible flows with respect to the original capacity function c, and let $\mathcal{F}'$ denote the corresponding polytope for c'. The polytope $\mathcal{F}$ is the intersection of $2m + n$ halfspaces, one for each linear constraint in the maximum-flow linear program; let $\mathcal{H}$ denote the boundary hyperplanes of those halfspaces. Every vertex of $\mathcal{F}$ is the intersection of m linearly-independent hyperplanes in $\mathcal{H}$. Similarly, let $\mathcal{H}'$ denote the corresponding hyperplanes for $\mathcal{F}'$.

Lemma 3.18. *Let v be the intersection point of m linearly independent hyperplanes in $\mathcal{H}$, and let v' be the intersection point of the corresponding hyperplanes in $\mathcal{H}'$. Then $\|v - v'\| \le 1/m^{3/2}$. In particular, v is the closest integer point to v'.*

Proof: The point v is the solution to an $m \times m$ linear system $Av = b$. Similarly, v' is the solution to a linear system $Av = b'$ with the same matrix A, where every coordinate in the vector $b' - b$ is either 0 or $1/m^3$. Let $[A \mid b]_i$ denote the matrix formed by replacing the ith column of A with the vector b. For each index i, Cramer's rule implies that

$$v'_i = \frac{\det[A \mid b']_i}{\det A} = \frac{\det[A \mid b]_i + \det[A \mid (b' - b)]_i}{\det A} = v_i + \frac{1}{m^3} \cdot \frac{\det[A \mid u]_i}{\det A}$$

for some $(0,1)$-vector u. The matrix A is a minor of a totally unimodular matrix (namely, the constraint matrix of a maximum-flow linear program), so $|\det A| = 1$. Similarly, cofactor expansion around the ith column implies that $|\det[A \mid u]_i| \le m$. Thus, $|v'_i - v_i| \le 1/m^2$ for all i, which implies $\|v - v'\| \le 1/m^{3/2}$. The total unimodularity of A also implies that v is an integer point. □

Lemma 3.19. *Every point in $\mathcal{F}'$ has distance at most $2/m^{3/2}$ from $\mathcal{F}$.*

Proof: Let φ' be an arbitrary point in $\mathcal{F}'$; then φ' is a feasible flow with respect to c'. Lower the capacities of the edges one at a time, replacing $c'(\vec{e})$ with $c(\vec{e})$, and simultaneously modify the flow to maintain feasibility. Each time we subtract $1/m^3$ from the capacity of one edge, we must modify the flow value of *every* edge by at most $1/m^3$. Thus, after all $2m$ edge capacities are changed, we have a new feasible flow $\varphi \in \mathcal{F}$ such that $|\varphi(\vec{e}) - \varphi'(\vec{e})| \le 2m/m^3 = 2/m^2$ for every edge $\vec{e}$. We conclude that $\|\varphi - \varphi'\| \le 2/m^{3/2}$. □

Lemma 3.20. *Every integer point that is not in $\mathcal{F}$ has distance at least $1/\sqrt{m}$ from $\mathcal{F}$.*

Proof: Fix a point $y \in \mathbb{Z}^m \setminus \mathcal{F}$. This point must violate some constraint $\langle a, x\rangle \le b$ in the constraint system that defines $\mathcal{F}$—either the capacity constraint of some edge, or the conservation constraint of some vertex. In either case, every coefficient of a is either 0, 1, or -1, which implies that $\|a\| \le \sqrt{m}$, and b is either 0 or the capacity of an edge. Let y' be the closest point on the hyperplane $\langle a, x\rangle = b$ to y; the distance from y to $\mathcal{F}$ is at least $\|y - y'\|$. Because a and y are integral, we have $\langle a, y\rangle \ge b + 1 = \langle a, y'\rangle + 1$, so $\langle a, y - y'\rangle \ge 1$. On the other hand, we also have $\langle a, y - y'\rangle \le \|a\| \cdot \|y - y'\| \le \sqrt{m} \cdot \|y - y'\|$. We conclude that $\|y - y'\| \ge 1/\sqrt{m}$. □

Let φ' be an arbitrary vertex of $\mathcal{F}'$, and let φ be the closest integer point to φ'. Lemmas 3.18 and 3.19 imply that φ has distance at most $3/m^{3/2}$ from $\mathcal{F}$. Thus, Lemma 3.20 implies that $\varphi \in \mathcal{F}$. On the other hand, Lemma 3.18 implies that φ is the intersection of m linearly-independent constraint hyperplanes, each supporting a facet of $\mathcal{F}$. It follows that φ is actually a vertex of $\mathcal{F}$.

Let $z \in \mathbb{R}^m$ denote the objective vector for the flow linear program, so that $z \cdot \varphi = |\varphi|$; because every coefficient of z lies in the set $\{-1, 0, 1\}$, we have $\|z\| \le \sqrt{m}$. Thus, Lemma 3.18 implies that rounding φ' to φ changes the flow value by at most $\|z\|\,\|\varphi - \varphi'\| \le 1/m$. Finally, we observe that the flow ϕ'_{OPT} computed by our ellipsoid-based algorithm is actually a vertex of $\mathcal{F}'$. We conclude that the closest integer point to ϕ'_{OPT} is an optimal vertex in $\mathcal{F}$, and so our reduction is correct.

Theorem 3.21. *Given an directed graph $G = (V, E)$ embedded on an orientable surface of genus g, a positive integer capacity function $c \colon E \to \mathbb{Z}^+$, and two vertices $s, t \in V$, a maximum (s,t)-flow in G can be computed in time $O(g^8 n \log^2 n \log^2 C)$, where C is the sum of the edge capacities.*

3.7 Non-orientable Surfaces

Finally, we consider the case where the underlying surface Σ is non-orientable. There is no globally consistent way to dualize a directed graph embedded on a non-orientable surface, so Lemma 3.1 cannot be directly generalized to this setting. Instead, we show that we can compute a maximum flow by considering a larger graph embedded on an orientable surface called the *orientable double cover* of Σ, invoking our earlier maximum-flow algorithm a constant number of times, and projecting the resulting flow back to the original graph.

We recall some standard definitions from topology. A continuous function $p \colon \hat{\Sigma} \to \Sigma$ is called a *covering map* if every point $x \in \Sigma$ has an open neighborhood U such that $p^{-1}(U)$ is the union of disjoint open sets $\bigcup_i \hat{U}_i$, and the restriction of p to each open set $\hat{U}_i$ is a homeomorphism. If such a map exists, then $\hat{\Sigma}$ is called a *covering space* of Σ. A point $\hat{x} \in \hat{\Sigma}$ is called a *lift* of its image $p(\hat{x})$. The *orientable double cover* of Σ is the unique orientable covering space $\hat{\Sigma}$ whose covering map is 2-to-1; that is, every point in Σ has exactly two lifts in $\hat{\Sigma}$. For example, the sphere is the orientable double cover of the projective plane, and the torus is the orientable double cover of the Klein bottle. If Σ is non-orientable and has genus g, Euler's formula implies that its orientable double cover $\hat{\Sigma}$ has genus $g - 1$. If Σ is orientable, its orientable double cover consists of two disjoint copies of Σ.

For any undirected graph G embedded on Σ, there is a corresponding lifted graph $\hat{G}$ embedded on the double-cover $\hat{\Sigma}$. We can compute both $\hat{G}$ and its embedding using the following standard voltage construction [55, Chapter 4]. Recall that a *rotation system* is a permutation π of the edges specifying the 'counterclockwise' ordering of the darts leaving each vertex. The embedding of G can be represented by a rotation system π, together with a *signature* function $\iota\colon E \to \{0,1\}$ specifying whether the local 'counterclockwise' orientations at the endpoints of each edge are consistent (0) or inconsistent (1). The lifted graph $\hat{G} = (\hat{V}, \hat{E})$ is now defined by setting $\hat{V} := \{v_0, v_1 \mid v \in V\}$ and $\hat{E} := \{u_0 v_{\iota(uv)},\ u_1 v_{1-\iota(uv)} \mid uv \in E\}$. The embedding of $\hat{G}$ onto $\hat{\Sigma}$ is defined by the following rotation system: for each vertex $v \in V$, darts leave v_0 in the same counterclockwise order as their lifts leave v, and darts leave v_1 in the opposite order.

Now let s and t be vertices of G, and let $c\colon E \to \mathbb{R}$ be a capacity function. We define a lifted capacity function $\hat{c}\colon \hat{E} \to \mathbb{R}$ by setting $\hat{c}(u_i v_j) = c(uv)/2$. A *multi-terminal* (single-commodity) flow in $\hat{G}$ is a function $\hat{\phi}\colon \hat{E} \to \mathbb{R}$ that satisfies the conservation constraint at every vertex except (possibly) at s_0, s_1, t_0, t_1. The *value* of a multi-terminal flow is the total net flow out of s_0 and s_1 (or into t_0 and t_1).

Now we augment $\hat{G}$ by adding new source and target vertices $\hat{s}$ and $\hat{t}$ and infinite-capacity edges $\hat{s}s_0$, $\hat{s}s_1$, $t_0\hat{t}$, and $t_1\hat{t}$; let $\hat{G}'$ denote the augmented graph. Adding these edges to the embedding of $\hat{G}$ requires increasing the genus of $\hat{\Sigma}$ by at most two. We can clearly extract a maximum multi-terminal flow in $\hat{G}$ from a maximum $(\hat{s}, \hat{t})$-flow in $\hat{G}'$.

Alternatively, we can compute the maximum multi-terminal flow using the following decomposition approach proposed by Miller and Naor [92]. Let f be a maximum (s_0, t_0)-flow in $\hat{G}$; let f' be a maximum (s_0, t_1)-flow in the residual graph $\hat{G}_f$; let f'' be a maximum (s_1, t_0)-flow in the residual graph $\hat{G}_{f'}$; and let f''' be a maximum (s_1, t_1)-flow in the residual graph $\hat{G}_{f''}$. Miller and Noar's results imply that $\hat{\phi} = f + f' + f'' + f'''$ is maximum multi-terminal flow in $\hat{G}$. Even if the original input graph G is undirected, the residual graphs $\hat{G}_f$, $\hat{G}_{f'}$, and $\hat{G}_{f''}$ are directed, so we need the results of the previous section to use this approach.

For each edge $uv \in G$, let $\phi(uv) := \hat{\phi}(u_0 v_{\iota(uv)}) + \hat{\phi}(u_1 v_{1-\iota(uv)})$. We mechanically prove that ϕ is a valid and feasible (s,t)-flow with the same value as $\hat{\phi}$ as follows. For any vertex v, except possibly s and t, we have $\partial\phi(v) = \partial\hat{\phi}(v_0) + \partial\hat{\phi}(v_1) = 0 + 0 = 0$, so ϕ is a valid (s,t)-flow. For any edge uv, we have

$$|\phi(uv)| \;\le\; |\hat{\phi}(u_0 v_{\iota(uv)})| + |\hat{\phi}(u_1 v_{1-\iota(uv)})| \;\le\; \hat{c}(u_0 v_{\iota(uv)}) + \hat{c}(u_1 v_{1-\iota(uv)}) \;=\; c(uv),$$

so ϕ is feasible. Finally, the definition of flow value implies that $|\phi| = \partial\phi(s) = \partial\hat{\phi}(s_0) + \partial\hat{\phi}(s_1) = |\hat{\phi}|$.

On the other hand, any (s,t)-flow φ in G can be lifted to a multi-terminal flow in $\hat{G}$ with the same value, by setting $\hat{\varphi}(u_i v_j) = \varphi(uv)/2$. Thus, ϕ is a maximum flow in G.

The following results are now immediate:

Theorem 3.22. *Given a graph $G = (V,E)$ embedded on a non-orientable surface of genus g, a positive integer capacity function $c\colon E \to \mathbb{Z}^+$, and two vertices $s, t \in V$, a maximum (s,t)-flow in G can be computed in time $O(g^8 n \log^2 n \log^2 C)$, where C is the sum of the edge capacities.*

Theorem 3.23. *Given a graph $G = (V,E)$ embedded on a non-orientable surface of genus g, a positive capacity function $c\colon E \to \mathbb{R}^+$, and two vertices $s, t \in V$, a maximum (s,t)-flow in G can be computed in $g^{O(g)} n^{3/2}$ time.*

4 Cohomology Cuts

Suppose we are given an undirected graph G (with no source or sink), a positive capacity function $c\colon E \to \mathbb{R}^+$, and a ***value*** function $\theta\colon \vec{E} \to \mathbb{R}$. The value of a circulation ϕ is the inner product

$\langle \phi, \theta \rangle = \sum_{\vec{e} \in \vec{E}} \phi(\vec{e}) \cdot \theta(\vec{e})$. Like the capacity function c, the value function θ is not (in general) a 1-chain; the values of a dart and its reversal need not have any relationship. In particular, some darts may have negative value. The goal of the *maximum-value circulation* problem is to compute a feasible circulation ϕ whose value $\langle \phi, \theta \rangle$ is as large as possible. The standard maximum-flow problem can be reduced to this problem by adding an edge $t \to s$ with infinite capacity and value 1 to the flow network, and assigning every other edge value 0. The maximum-value circulation problem is equivalently—and much more commonly—formulated as finding a feasible circulation of minimum *cost*, where the cost of an edge is just the negation of its value.

Our methods do not improve the fastest algorithms for the general maximum-value/minimum-cost circulation problem in surface-embedded graphs; even for planar graphs, the fastest algorithms known are those for arbitrary sparse graphs [31, 97]. However, a minor modification of our maximum-flow algorithm allows us to solve two interesting special cases in roughly the same running time. In the first special case, described in Section 4.1, we require that the value function is *homology invariant*; that is, any two homologous circulations must have the same value. In the second special case, described in Section 4.2, we find the minimum-cost circulation *in a given homology class*; this special case requires each edge to have a non-negative cost and infinite capacity. These two special cases are related by a combination of combinatorial (Poincaré) duality and linear programming duality.

4.1 Homology-Invariant Values

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

Theorem 4.1. *Given a graph $G = (V, E)$ embedded on a surface of genus g, a capacity function $c : E \to \mathbb{R}^+$, and a homology-invariant value function $\theta : \vec{E} \to \mathbb{R}$, we can compute a maximum-value circulation in $g^{O(g)} n^{3/2}$ time, or in $O(g^8 n \log^2 n \log^2 C)$ time if capacities are integers that sum to C.*

Proof: The homology space $H(G) \cong \mathbb{R}^{2g}$ can be generated by (the homology classes of) $2g$ directed cycles $\gamma_1, \gamma_2, \ldots, \gamma_{2g}$ in independent homology classes. The proof of Lemma 3.5 implies that we can construct such a *homology basis* in $O(gn)$ time [39, 41]. (See also the proof of Lemma 3.8.)

Corollary 3.6 implies that it suffices to find the homology class of the maximum-value circulation. Specifically, we need to find a feasible homology vector $(\phi_1, \ldots, \phi_{2g})$ such that the cost function

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

is maximized. Corollary 3.6 gives us both strong membership and strong separation oracles for this linear optimization problem, so we can apply either the central-cut ellipsoid method or multidimensional parametric search, exactly as we did for the standard maximum-flow problem. The perturbation scheme described in Section 3.4.3 requires only minor modifications to support more general objective functions; the standard details are described by Grötschel *et al.* [57]. Otherwise, the optimization algorithm is identical. □

The following lemma exactly characterizes homology-invariant value functions. Recall that a 1-chain $\theta : E \to \mathbb{R}$ is a ***cocirculation*** if its dual 1-chain $\theta^* : E^* \to \mathbb{R}$, defined by setting $\theta^*(\vec{e}^*) = \theta(\vec{e})$, is a circulation in G^*.

Lemma 4.2. *A value function $\theta\colon \vec{E} \to \mathbb{R}$ is homology invariant if and only if θ is a cocirculation.*

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

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

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

On the other hand, suppose θ is an arbitrary cocirculation and $\alpha\colon F \to \mathbb{R}$ is an arbitrary 2-chain. We mechanically verify that $\langle \partial\alpha, \theta \rangle = 0$ as follows:

$$\begin{aligned}
\langle \partial\alpha, \theta \rangle &= \sum_{\vec{e}\in\vec{E}} \partial\alpha(\vec{e}) \cdot \theta(\vec{e}) \\
&= \sum_{\vec{e}\in\vec{E}} \big(\alpha(\mathit{left}(\vec{e})) - \alpha(\mathit{right}(\vec{e}))\big) \cdot \theta(\vec{e}) \\
&= \sum_{f\in F} \Bigg(\sum_{\vec{e}\colon \mathit{left}(\vec{e})=f} \alpha(f)\cdot\theta(\vec{e}) \;-\; \sum_{\vec{e}\colon \mathit{right}(\vec{e})=f} \alpha(f)\cdot\theta(\vec{e}) \Bigg) \\
&= \sum_{f\in F} \Bigg(\alpha(f)\cdot \Bigg(\sum_{\vec{e}\colon \mathit{left}(\vec{e})=f} \theta(\vec{e}) \;-\; \sum_{\vec{e}\colon \mathit{right}(\vec{e})=f} \theta(\vec{e}) \Bigg)\Bigg) \\
&= \sum_{f\in F} \big(\alpha(f)\cdot(0-0)\big) \;=\; 0.
\end{aligned}$$

We conclude that θ is homology-invariant. □

4.2 Minimum-Cost Homologous Circulation

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

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

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

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

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

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

Let $\alpha_{OPT}(f)$ and $x_{OPT}(f\uparrow g)$ denote the variables in the optimum solution to this dual-dual linear program. We view the vector α_{OPT} of face variables as a 2-chain. We define a 1-chain $\vartheta\colon E^* \to \mathbb{R}$ by setting $\vartheta_{OPT}(f\uparrow g) := x_{OPT}(f\uparrow g) - x_{OPT}(g\uparrow f)$ for every dart $f\uparrow g$. Because every primal capacity $c(u\to v)$ is non-negative, each dart variable $x_{OPT}(f\uparrow g)$ is individually as small as possible without violating any constraint; that is,

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

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

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

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

Theorem 4.3 (Homological Maxflow/Mincut). *Let $G = (V,E)$ be an undirected graph embedded on a surface of genus g, let $c\colon E \to \mathbb{R}^+$ be a capacity function, and let $\theta : E \to \mathbb{R}$ be a cocirculation in G. The maximum value $\langle\phi,\theta\rangle$ of any feasible circulation ϕ in G is equal to the minimum capacity of any cocirculation cohomologous with θ.*

The previous theorem is a special case of a more general result of Sullivan [109], relating optimal homologous $(d-1)$-chains ("surfaces") in any orientable d-manifold cell complex to minimum-cost flows in the 1-skeleton of the dual cell complex. Essentially the same result was rediscovered by Buehler *et al.* [15, 51, 81]; see also recent results of Grady [52, 53].

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

Theorem 4.4. *Given a graph $G = (V,E)$ embedded on a surface of genus g, a cost function $c\colon E \to \mathbb{R}^+$, and a circulation $\theta : E \to \mathbb{R}$, we can compute a minimum-cost circulation homologous with θ in $g^{O(g)}n^{3/2}$ time, or in time $O(g^8 n\log^2 n\log^2 C)$ if all capacities are integers that sum to C.*

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

$$\begin{array}{lll} \max & \displaystyle\sum_{i=1}^{2g}\phi^*_i\cdot\langle\theta,\lambda^*_i\rangle & \\ \text{s.t.} & \displaystyle\sum_{i=1}^{2g}\phi^*_i\cdot\lambda^*_i(\gamma)\ \le\ c(\gamma) & \text{for every cycle } \gamma \text{ in } \vec{G} \end{array} \tag{4.2}$$

Here, $\lambda^*_1,\lambda^*_2,\ldots,\lambda^*_{2g}$ are cycles in G^* that generate the homology space of G^*. As described in the proof of Lemma 3.5, we can construct a suitable set of $2g$ cycles in $O(gn)$ time.

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

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

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

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

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

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

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

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

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

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

Finally, consider the special case where the input circulation θ is a single directed cycle. The minimum-cost circulation ϑ_{OPT} homologous to θ is not necessarily a single cycle, but the proof of the previous theorem implies that it is a weighted sum of at most $2g$ directed cycles. Moreover, ϑ_{OPT} is defined by a linear program (4.1) whose constraint matrix is totally unimodular, which implies that ϑ_{OPT} is an *integer* circulation. For more general applications of total unimodularity to optimization within an integer homology class, we refer the reader to Dey *et al.* [30] and Dunfield and Hirani [34].

5 Conclusions and Open Problems

We have described the first algorithm to compute maximum flows in graphs embedded on any fixed surface, with polynomially-bounded integer capacities, in near-linear time. In two related papers [20, 40], we describe the first algorithms to compute minimum cuts in undirected surface-embedded graphs in $O(n \log n)$ time for any fixed genus; this running time was recently improved to $O(n \log\log n)$ by Italiano *et al.* [72]. Both in this paper and in related work, we also consider the problem of finding the minimum-cost representative in a given homology class; the results depend on the coefficient ring used to define homology. For real or integer coefficients, the problem is a special case of minimum-cost circulation [109] and thus can be solved using our maximum flow algorithm. For $\mathbb{Z}_2$-coefficients, on the other hand, the problem is NP-hard, by a reduction from MaxCut [20], and thus can be solved efficiently only when the genus is small.

We regard the results in this paper largely as a 'proof of concept' that topology can helpful in solving flow problems. Our algorithms are in no way practical; we expect that for all but the simplest examples, existing algorithms for general graphs will significantly outperform all of our algorithms in practice. We optimistically conjecture that maximum flows and minimum cuts in embedded graphs can be computed in $O(g^k n \log n)$ time for some small constant k, perhaps by decomposing the graph into planar components, or using a generalization of the planar network-simplex algorithm of Borradaile and Klein [11, 13, 38]. Even the special case of undirected graphs with unit capacities on the torus is open.

Acknowledgments. Thanks to Cora Borradaile, Chandra Chekuri, Sariel Har-Peled, Shay Mozes, Aparna Sundar, and Kim Whittlesey for many helpful discussions. In particular, we would like to thank Sariel Har-Peled for suggesting the use of the ellipsoid method, and Shay Mozes for pointing out that Theorem 3.3 was not as simple as we claimed.

References

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

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

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

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

[5] L. Aleksandrov and H. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM J. Discrete Math.* 9(1):129–150, 1996.

[6] H. Alt. Functions equivalent to integer multiplication. *Proc. 7th Int. Colloq. Automata Lang. Prog.*, 30–37, 1980. Lecture Notes Comput. Sci. 85, Springer.

[7] Y. P. Aneja and S. N. Kabadi. Polynomial algorithms for Lagrangean relaxations in combinatorial problems. Faculty of Business Working Paper Series W91-03, University of Windsor, 1991. Cited in [76].

[8] B. S. Baker. Approximation algorithms for NP-complete problesm on planar graphs. *J. Assoc. Comput. Mach.* 41:153–180, 1994.

[9] R. G. Bland, D. Goldfarb, and M. J. Todd. The ellipsoid method: A survey. *Oper. Res.* 29(6):1039–1091, 1981.

[10] A. I. Bobenko, P. Schröder, J. M. Sullivan, and G. M. Ziegler. *Discrete Differential Geometry*. Olberwolfach Seminars 38. Birkhäuser Verlag, 2008.

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

[12] G. Borradaile, E. D. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Proc. 26th Int. Symp. Theoretical Aspects Comput. Sci.*, 171–182, 2009. Dagstuhl Seminar Proceedings. ⟨http://drops.dagstuhl.de/opus/volltexte/2009/1835/⟩.

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

[14] G. Borradaile, P. Klein, and C. Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Trans. Algorithms* 5(3):article 31, 2009.

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

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

[17] S. Cabello, É. Colin de Verdière, and F. Lazarus. Finding cycles with topological properties in embedded graphs. Preprint, October 2010. ⟨http://www.di.ens.fr/~colin/textes/09truecycle.pdf⟩. To appear in *SIAM J. Discrete Math*.

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

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

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

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

[22] J. Chen, S. P. Kanchi, and A. Kanevsky. A note on approximating graph genus. *Inform. Proc. Lett.* 61(6):317–322, 1997.

[23] E. Cohen. *Combinatorial Algorithms for Optimization Problems*. Ph.D. thesis, Dept. Comput. Sci., Stanford Univ., June 1991. ⟨http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA254553⟩. Tech. Report STAN-CS-91-1366.

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

[25] E. Cohen and N. Megiddo. Maximizing concave functions in fixed dimension. *Complexity in Numerical Optimization*, 74–87, 1993. World Scientific.

[26] E. Cohen and N. Megiddo. Strongly polynomial-time and NC algorithms for detecting cycles in periodic graphs. *J. Assoc. Comput. Mach.* 40(4):791–830, 1993.

[27] E. Cohen and N. Megiddo. Algorithms and complexity analysis for some flow problems. *Algorithmica* 11(3):320–340, 1994.

[28] E. D. Demaine, M. Hajiaghayi, and B. Mohar. Approximation algorithms via contraction decomposition. *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, 278–287, 2007.

[29] M. Desbrun, E. Kanso, and Y. Tong. Discrete differential forms for computational modeling. *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*, 39–54, 2006.

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

[31] S. I. Diatch and D. A. Spielman. Faster lossy generalized flow via interior point algorithms. *Proc. 40th Ann. ACM Symp. Theory Comput.*, 451–460, 2008. ArXiv:0803.0988.

[32] H. N. Djidjev and S. M. Venkatesan. Planarization of graphs embedded on surfaces. *Proc. 21st Workshop Graph-Theoretic Concepts Comput. Sci.*, 62–72, 1995. Lecture Notes Comput. Sci. 1017, Springer-Verlag.

[33] D. P. Dobkin and R. J. Lipton. On the complexity of computations under varying sets of primitives. *J. Comput. Syst. Sci.* 18:86–91, 1979.

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

[35] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.* 3(3):1–27, 1999.

[36] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica* 27:275–291, 2000. ArXiv:math.CO/9907126.

[37] D. Eppstein. Dynamic generators of topologically embedded graphs. *Proc. 14th Ann. ACM-SIAM Symp. Discrete Algorithms*, 599–608, 2003. ArXiv:cs.DS/0207082.

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

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

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

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

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

[43] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian J. Math.* 8(399–404), 1956. First published as Research Memorandum RM-1400, The RAND Corporation, Santa Monica, California, November 19, 1954.

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

[45] P. Gács and L. Lovász. Khachiyan's algorithm for linear programming. *Math. Program. Stud.* 14:61–68, 1981.

[46] S. O. Gharan and A. Saberi. The asymmetric traveling salesman problem on graphs with bounded genus. *Proc. 22nd Ann. ACM-SIAM Symp. Discrete Algorithms*, 967–975, 2011.

[47] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms* 5(3):391–407, 1984.

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

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

[50] M. T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.* 51(3):374–389, 1995.

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

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

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

[54] M. Grohe. Isomorphism testing for embeddable graphs through definability. *Proc. 32nd ACM Symp. Theory Comput.*, 63–72, 2000.

[55] J. L. Gross and T. W. Tucker. *Topological graph theory*. Dover Publications, 2001.

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

[57] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, 2nd edition. Algorithms and Combinatorics 2. Springer-Verlag, 1993.

[58] T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *J. Comput. Syst. Sci.* 57(3):366–375, 1998.

[59] T. E. Harris and F. S. Ross. Fundamentals of a method for evaluating rail net capacities. Memorandum RM-1573, The RAND Corporation, Santa Monica, California, October 24, 1955. Cited in [106].

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

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

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

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

[64] A. N. Hirani. *Discrete Exterior Calculus*. Ph.D. thesis, California Institute of Technology, 2003. ⟨http://resolver.caltech.edu/CaltechETD:etd-05202003-095403⟩.

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

[66] J. E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). *Proc. 6th Ann. ACM Symp. Theory Comput.*, 172–184, 1974.

[67] J. P. Hutchinson. On short noncontractible cycles in embedded graphs. *SIAM J. Discrete Math.* 1(2):185–192, 1988.

[68] J. P. Hutchinson. On genus-reducing and planarizing algorithms for embedded graphs. *Graphs and Algorithms, Proc. AMS-IMS-SIAM Joint Summer Res. Conf.*, 19–26, 1989. Contemporary Mathematics 89, American Mathematical Society.

[69] J. P. Hutchinson and G. L. Miller. Deleting vertices to make graphs of positive genus planar. *Discrete Algorithms and Complexity Theory, Proceedings of the Japan-US Joint Seminar, Kyoto, Japan*, 81–98, 1987. Academic Press.

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

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

[72] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Improved minimum cuts and maximum flows in undirected planar graphs. *Proc. 43rd Ann. ACM Symp. Theory Comput.*, to appear, 2011.

[73] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. Assoc. Comput. Mach.* 24(1):1–13, 1977.

[74] D. B. Johnson and S. M. Venkatesan. Partition of planar flow networks (preliminary version). *Proc. 24th IEEE Symp. Found. Comput. Sci.*, 259–264, 1983. IEEE Computer Society.

[75] S. N. Kabadi and Y. P. Aneja. ϵ-approximation minimization of convex functions in fixed dimension. *Oper. Res. Lett.* 18:171–176, 1996.

[76] S. N. Kabadi and Y. P. Aneja. Equivalence of ϵ-approximate separation and optimization in fixed dimensions. *Algorithmica* 29:582–594, 2001.

[77] K. Kawarabayashi, B. Mohar, and B. Reed. A simpler linear time algorithm for embedding graphs into an arbitrary surface and the genus of graphs of bounded tree-width. *Proc. 49th IEEE Symp. Found. Comput. Sci.*, 771–780, 2008.

[78] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Math. Dokl.* 20(1):191–194, 1979. Translated from *Doklady Akademiia Nauk SSSR* 244:1093–1996, 1979.

[79] L. G. Khachiyan. Polynomial algorithms in linear programming. *USSR Comp. Math. and Math. Phys.* 20:53–72, 1980. Translated from *Zhurnal Vychisditel'noi Matematiki i Matematicheskoi Fiziki* 20:51–68, 1980.

[80] S. Khuller, J. Naor, and P. Klein. The lattice structure of flow in planar graphs. *SIAM J. Discrete Math.* 477–490, 1993.

[81] D. Kirsanov. *Minimal discrete curves and surfaces*. Ph.D. thesis, Div. Engin. Appl. Sci., Harvard Univ., September 2004. ⟨http://www.eecs.harvard.edu/~sjg/papers/danilthesis.pdf⟩.

[82] M. M. Klawe and D. J. Kleitman. An almost linear time algorithm for generalized matrix searching. *SIAM J. Discrete Math.* 3(1):81–97, 1990.

[83] P. Klein. Multiple-source shortest paths in planar graphs. *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms*, 146–155, 2005.

[84] P. Klein, S. Mozes, and O. Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *ACM Trans. Algorithms* 6(2):article 30, 2010.

[85] M. Kutz. Computing shortest non-trivial cycles on orientable surfaces of bounded genus in almost linear time. *Proc. 22nd Ann. Symp. Comput. Geom.*, 430–438, 2006.

[86] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.* 16:346–358, 1979.

[87] M. Mareš. Two linear time algorithms for MST on minor closed graph classes. *Archivum Mathematicum* 40(3):315–320, 2004.

[88] W. S. Massey. *A basic course in algebraic topology*. Springer-Verlag, 1991.

[89] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. Assoc. Comput. Mach.* 30(4):852–865, 1983.

[90] G. L. Miller. Isomorphism testing for graphs of bounded genus. *Proc. 12th Ann. ACM Symp. Theory Comput.*, 225–235, 1980.

[91] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. System Sci.* 32(3):265–279, 1986.

[92] G. L. Miller and J. Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.* 24(5):1002–1017, 1995.

[93] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM J. Discrete Math.* 12(1):6–26, 1999.

[94] B. Mohar and C. Thomassen. *Graphs on Surfaces*. Johns Hopkins Univ. Press, 2001.

[95] S. Mozes and C. Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. *Proc. 18th Ann. Europ. Symp. Algorithms*, 206–217, 2010. Lecture Notes Comput. Sci. 6347, Springer-Verlag.

[96] C. H. Norton, S. A. Plotkin, and É. Tardos. Using separation algorithms in fixed dimension. *J. Algorithms* 13(1):79–98, 1992.

[97] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Oper. Res.* 41(2):338–350, 1993.

[98] V. Y. Pan and J. H. Reif. Fast and efficient parallel solution of sparse linear systems. *SIAM J. Comput.* 22(6):1227–1250, 1993.

[99] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[100] D. Pe'er. On minimum spanning trees. Master's thesis, Hebrew University, 1998. ⟨http://www.math.ias.edu/~avi/STUDENTS/dpthesis.pdf⟩.

[101] H. Poincaré. Second complément à l'Analysis Situs. *Proc. London Math. Soc.* 32:277–308, 1900.

[102] J. Reif. Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time. *SIAM J. Comput.* 12:71–81, 1983.

[103] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing* 7:281–292, 1971.

[104] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, 1986.

[105] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics 24. Springer-Verlag, 2003.

[106] A. Schrijver. On the history of combinatorial optimization (till 1960). *Handbook of Discrete Optimization*, 1–68, 2005. Elsevier.

[107] N. Z. Shor. Cut-off method wth space extension in convex programming problems. *Cybernetics* 13(1):94–96, 1977. Translated from *Kibernetika* (1):94–95, 1977. Cited in [9, 57].

[108] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.* 26(3):362–391, 1983.

[109] J. M. Sullivan. *A Crystalline Approximation Theorem for Hypersurfaces*. Ph.D. thesis, Princeton Univ., October 1990. ⟨http://torus.math.uiuc.edu/jms/Papers/thesis/thesis.pdf⟩.

[110] S. Tazari and M. Müller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to Steiner tree approximation. *Discrete Appl. Math.* 157:673–684, 2009.

[111] C. Thomassen. The graph genus problem is NP-complete. *J. Algorithms* 10(4):568–576, 1989.

[112] S. Toledo. Maximizing non-linear concave functions in fixed dimension. *Complexity in Numerical Optimization*, 429–447, 1993. World Scientific.

[113] P. M. Vaidya. Speeding-up linear programming using fast matrix multiplication. *Proc. 30th IEEE Symp. Found. Comput. Sci.*, 332–337, 1989.

[114] S. M. Venkatesan. *Algorithms for network flows*. Ph.D. thesis, The Pennsylvania State University, 1983. Cited in [74].

[115] K. Weihe. Edge-disjoint (s,t)-paths in undirected planar graphs in linear time. *J. Algorithms* 23(1):121–138, 1997.

[116] K. Weihe. Maximum (s,t)-flows in planar networks in $O(|V| \log |V|)$-time. *J. Comput. Syst. Sci.* 55(3):454–476, 1997.

[117] A. C.-C. Yao. On the complexity of comparison problems using linear functions (preliminary report). *Proc. 16th IEEE Ann. Symp. Foundations Comput. Sci.*, 85–89, 1975.

[118] D. B. Yudin and A. S. Nemirovskiĭ. Evaluation of the informational complexity of mathematical programming problems. *Matekon* 13(2):3–25, 1976–7. Translated from *Èkonomika i Matematicheskie Metody* 12:357–369, 1976. Cited by [9, 57].

[119] D. B. Yudin and A. S. Nemirovskiĭ. Informational complexity and effective methods of solution for convex extremal problems. *Matekon* 13:25–45, 1977. Translated from *Ekonomika i Matematicheskie Metody* 12:357–369, 1976. Cited by [9, 57].