MORE HOMOLOGY FLOWS

BY

APARNA SUNDAR

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Associate Professor Jeff Erickson

Abstract

Flows and cuts have been the topic of much study since Ford and Fulkerson's original paper. The problem we look at is the computation of flows on some generalizations of planar graphs. In particular, the input graph can be embedded on a surface of genus g, and has the source and sink on the same face. We show this problem can be reduced to a convex programming problem in dimension $2g$, and also show some interesting properties of the feasible polytope.

To Mother and Father.

Acknowledgments

This project was made possible by the support of many people - my adviser, Jeff Erickson, who read my numerous revisions and helped make some sense of the confusion, my friends Pavithra Prabhakar and Sankar Gurumurthy, for reading my revisions and offering useful criticism and my parents for all their support. Thanks also to the Computer Science Department at the University of Illinois for giving me a TA ship through the course of my education at the University of Illinois.

Table of Contents

Chapter 1

Introduction

The problem of finding the maximum flow in a network has been well studied for more than 50 years. In this problem, we are given a graph with capacities on the edges and two specific vertices s and t, and we are to find the maximum flow from s to t. There are many variants of the problem - some of these include the case when the graph is directed, or the case when it is undirected, or when the edge capacities are constrained to be integers, or when there is also a cost associated with each edge. Ford and Fulkerson's work in 1956 [15] was the first in a long line of results on this topic, some of which we will describe in the next section.

Flows in planar graphs are a natural topic of interest because many applications of flows, such as VLSI design or road transportation networks, involve planar graphs. Also, algorithms for planar graphs can often be extended to more general graph families such as graphs that can be embedded on surfaces of higher genus, or graphs with forbidden minors. Examples include shortest path algorithms [7], finding induced cycles [34], approximation algorithms for NP hard problems [5] and subgraph isomorphism [13].

In the case of the maximum flow problem, studying the special case of st-planar graphs inspired the development of algorithms for general planar graphs. st-planar graphs are graphs where s and t are on the same face; we redraw any st-planar graph such that the common face is the exterior face. st-planar graphs received much early attention, and fast $O(n \log n)$ algorithms for the maximum flow problem on st-planar graphs were discovered *much* before similar algorithms were designed for general planar graphs. Ford and Fulkerson [15] described the uppermost path algorithm to solve the st-planar maximum flow problem. As s and t are both on the exterior face, the uppermost path is just the uppermost path on the given drawing of the st-planar graph. The bottleneck edge of the uppermost path can only be saturated once, after which it can safely be removed from consideration. Because each iteration saturates at least one edge, the algorithm terminates after $O(n)$ iterations. Itai and Shiloach showed that each iteration can be implemented in $O(\log n)$ time [29], which implies an overall running time of $O(n \log n)$.

Weihe [44] described a $O(n \log n)$ algorithm for planar graphs that satisfy a certain connectivity requirement. The first $O(n \log n)$ time algorithm for max-

imum flows on arbitrary directed planar graphs was given by Borradaille and Klein [4]. They generalized uppermost paths in st-planar graphs to "leftmost paths". (For a more formal description of leftmost path, see Chapter 3.) They also show that each edge can be saturated at most three times, and that each pivot takes $O(\log n)$ time, thus resulting in an $O(n \log n)$ time algorithm.

Ford and Fulkerson [15] observed that for st-planar networks, the cuts found in the primal network corresponded to a shortest path being computed in the dual. In their own words, "chains of N joining source and sink correspond to cuts (relative to two particular vertices) of the dual of N and vice versa". Hassin [24] proved that maximum flows in st-planar graphs can be found by computing a shortest path in the dual network.

Hassin and Johnson [25] in turn generalized this approach and showed that given the value of a feasible flow in a network, as well as some minimum cut information, one shortest path computation of a derived network finds a flow of that specified value. Later Erickson [14] showed that the maximum flow problem in general planar graphs can be modeled as a parametric shortest path problem in the dual, and he presented an algorithm using dynamic tree data structures to keep track of the changes in the dual shortest path tree as the parameter changes, yielding the same algorithm as Borradaille and Klein [4].

As mentioned before, algorithms for planar graphs have often been extended to algorithms for graphs that can be embedded on higher genus surfaces. The maximum flow problem seems to be an exception in that though there has been a great deal of focus on st-planar and planar graphs, these algorithms had not been generalized for higher genus graphs. In fact, the best algorithms for higher genus graphs were the best algorithms for general graphs. One early algorithm by Imai and Iwano [28] could be adapted to give a running time of $O(n^{1.595} \log C)$ where C is the sum of edge capacities; however it can be outperformed by fast maximum flow algorithms for general graphs. Recently, Chambers, Erickson and Nayyeri [8] showed that the computation of flows in a graph embedded on a surface of genus g, can be reduced to a linear program in $2g + 1$ dimensions. Using the ellipsoid method, leads to a $O(g^7 n \log^2 n \log^2 C)$ algorithm for the maximum flow problem. The maximum flow as parametric shortest path in the dual approach also does not lead to an $O(n \log n)$-time algorithm for higher genus graphs because $\Omega(n^2)$ pivots may need to occur in the dual shortest path tree [14].

My thesis considers the maximum flow problem for surface embedded graphs, where the source and sink are on the same face. We show this problem can be reduced to a $2g$ dimensional convex programming problem using the same tools as Chambers, Erickson and Nayyeri [8]. We use a standard reduction of flows to circulations in order to look at the *feasible classes of homology circulations* instead of *feasible classes of homology flows*. The reduction is as follows. By adding an infinite-capacity edge $t \to s$ and finding a circulation ϕ, we can find an $s - t$ flow with value $\phi(t \to s)$ by simply dropping the edge $t \to s$.

We consider each feasible homology class of circulations, and show that one shortest path computation in a modified dual graph is sufficient to find the homologous feasible circulation carrying the largest flow across new edge $t \to s$. If there is no feasible circulation in the given homology class, the shortest path algorithm finds a negative cycle in the dual graph. We solve the maximum flow problem using the ellipsoid method, using this shortest path algorithm as an oracle. We present a general discussion of the method before applying it to the maximum flow problem for surface graphs. The resulting algorithm runs in $O(g^6 n \log^2 n \log^2(nC))$ time.

Chapter 2

Definitions

We first define the terms that we will be using. Much of this material is from Chambers, Erickson, and Nayyeri [8]. More detailed references to texts on algebraic topology include Hatcher [26] and Massey [36]; standard references in topological graph theory include Gross and Tucker [20] and Mohar and Thommasen [37]. Some of this material is also from lecture notes by Boyd [6] made available by Stanford as part of their Stanford Engineering Everywhere initiative. For more details on the ellipsoid method, see the textbook by Grötschel, Lovasz and Schrijver [22].

2.1 Graphs, planarity and duality

An *undirected graph* is an ordered pair $G(V, E)$ where V is a set of *vertices* and E is a set of undirected *edges*, where an undirected edge is an unordered pair of vertices. We use uv to denote the edge between vertex u and vertex v. Usually the size of V is denoted by n and the size of E by m. A *directed graph* is an ordered pair $\vec{G}(V, \vec{E})$ where V is defined as before and $\vec{E}$ is a set of *directed edges*. A *directed edge* is an ordered pair, which we denote $u \to v$. We can transform any undirected graph G into a directed graph $\vec{G}$ by replacing each edge uv with two oppositely directed edges $u \to v$ and $v \to u$. Both G and $\vec{G}$ can be represented by the same adjacency matrix.

A *planar graph* is a graph that can be drawn in the plane or on the sphere such that vertices are mapped to distinct points and edges to curves that intersect only at common endpoints. Any embedding of a planar graph divides the plane into regions called *faces*.

The *dual graph* of an embedded planar graph G is another graph G^*, which has a vertex for each face of G and an edge for each edge of G separating two neighboring faces. For any face f of G, let f^* denote the corresponding vertex in G^*. If $u \to v$ is an edge with face f_l to its left and face f_r to its right in G, we can denote this edge by $f_l \uparrow f_r$; we denote its dual edge as $f_l^* \to f_r^*$. Duality is an involution as $(G^*)^*$ is isomorphic to G.

Suppose G is fixed and H is a subgraph of G. We will use the notation H^* to denote the subgraph of G^* containing the edges dual to the edges of H. When the edges of H^* form a cycle in G^*, the subgraph H of G is called a *cocycle*.

2.2 Flows, circulations, and capacity

Let s and t be two vertices on G, called the source and target respectively. A *flow* is a function $f : \vec{E} \to R$ that satisfies the following equations: the conservation constraint $\sum_{v \in V} f(u \to v) = 0$ for every vertex $u \in V \setminus \{s, t\}$ and the skew symmetry constraint. $f(u \to v) = -f(v \to u)$ for every edge $u \to v$. We extend any flow to a function on E, by fixing an arbitrary direction $u \to v$ for each undirected edge uv, and then let $f(uv) = f(u \to v)$. The *value* of the flow is $\sum_{sr \in E} f(s \to r)$ which is the net outgoing flow at s. If the conservation constraints also apply at s and t, then f is called a *circulation*.

Let $c : E \to R^+$ be a capacity function such that $c(uv) \geq 0$. We extend the capacity function to $c : \vec{E} \to R$ by defining $c(u \to v) = c(v \to u) = c(uv)$. A flow f is *feasible* if $|f(uv)| \leq c(uv)$ for every edge uv in E. The *residual capacity* of an edge is $c_f(u \to v) = c(u \to v) - f(u \to v)$; the *residual graph* $G_f(V, \vec{E_f})$ is the subgraph of edges in $\vec{G}$ with nonzero residual capacity. A flow f is feasible if and only if every edge in G_f has nonnegative residual capacity. A *maximum flow* f is a feasible flow of maximum value. f is a maximum flow if and only if there are no directed paths in G_f from s to t.

As mentioned earlier, a circulation is a flow where the conservation constraint holds at every vertex. Every circulation is a point in R^{2m} satisfying n conservation constraints and the m skew symmetry constraints. This subspace is called the *circulation space* and is denoted by $Z(G)$. $Z(G)$ is a subspace of R^{2m} with dimension $2m - (m + n - 1) = m - n + 1$ because, of the n conservation equations, only $n - 1$ are linearly independent. Thus the circulation space $Z(G)$ is isomorphic to R^{m-n+1}. Similarly, a flow is a point R^{2m} satisfying $n-2$ conservation constraints (which are linearly independant and) and m skew symmetry constraints. Thus, the *flow space* $Z(G; s, t)$ is isomorphic to R^{m-n+2}.

2.3 Graphs on surfaces

A *surface* is a Hausdorff topological space in which every point has an open neighborhood homeomorphic to R^2. We consider only compact, connected, and orientable surfaces with no boundary. 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, where a *cycle* is defined as the (the image of) a continuous map $\gamma : S^1 \to \Sigma$.

An *embedding* of graph G on a surface Σ is a drawing of the graph on Σ, with the vertices mapped to points and edges to non-crossing curves. This definition generalizes embeddings of planar graphs on the sphere. A *face* is a maximal connected subset of Σ that does not intersect the image of any edge or vertex. An embedding is *cellular* if every face is an open topological disk. The *dual graph* of a graph embedded on a surface is defined exactly as for planar graphs.

Suppose G is a simple n vertex graph that has a cellular embedding on Σ.

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 each face of the embedding is a triangle. Thus for any graph that can be embedded on a surface of genus $g = o(n)$, the overall complexity of any embedding is $O(n)$. Our input will consist of a graph $G(V, E)$ embedded on Σ. F is the set of faces on Σ.

2.4 Topology of flows on surfaces

We now redefine flows and circulations in the language of homology.

A 0-*chain* is a function $w : V \to R$; a 1-*chain* is a function $\phi : E \to R$; and a 2-*chain* is a function $\alpha : F \to R$. Note that we are using real coefficients in our definition of chains; later in this section, when we define homology, we will also use real coefficients. Most standard texts use integral coefficients in both cases.

A 1-chain assigns both an orientation and a non-negative value to each undirected edge: for each undirected edge uv in E, choose one of the two corresponding directed edges, say $u \to v$, and set $\phi(u \to v) = \phi(uv)$ and $\phi(v \to u) = -\phi(uv)$.

The *boundary* of 1-chain ϕ is the 0-chain $\partial\phi : V \to R$ defined as

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

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 just the conservation constraint at vertex v. An *st-flow* is a 1-chain ϕ such that $\partial\phi(v) = 0$ for every vertex v except s and t . The *value* of the flow ϕ is $\partial\phi(t) = -\partial\phi(s)$; a circulation is simply a flow with value 0.

The boundary of 2-chain $\alpha : F \to R$ is the 1-chain $\partial\alpha : E \to R$ defined as $\partial\alpha(u \to v) = \alpha(f_r) - \alpha(f_l)$, where f_r is the face to the right of $u \to v$ and f_l the face to the left. The set of all incoming edges to any vertex z forms a cocycle λ. We have $\sum_{u\to v\in\lambda} \partial\alpha(u \to v) = \sum_{f_l\to f_r\in\lambda^*} \partial\alpha(f_l \to f_r) = 0$, because for every vertex v in λ^*, the term $\alpha(v)$ occurs once with a positive sign and once with a negative sign. Thus the net incoming flow to any vertex z is zero, and the conservation constraint at z is met and the boundary of α is a circulation.

A *boundary circulation* is the boundary of some 2-chain. For planar graphs, every circulation is a boundary circulation, but this is not the case for graphs on higher genus surfaces. The *boundary space* $B(G)$ is the vector space of all boundary circulations; this is a linear subspace of $Z(G)$. The space of 2-chains has dimension F, but two 2-chains, that differ by a constant, have the same boundary. Thus, $B(G)$ is isomorphic to $R^{|F|-1}$.

Two flows ϕ and ψ are *homologous* if $\phi - \psi$ is a boundary circulation; homologous flows are said to belong to the same *homology class*. In particular, two flows in a planar graph are homologous if and only if they have the same value.

The homology space $H(G)$ is the vector space of all homology classes of circulations in G, which is isomorphic to $Z(G)/B(G) \cong R^{|E|-|V|-|F|+2} = R^{2g}$ by Euler's formula. We say a homology class of circulations is *feasible* if it contains at least one feasible circulation. The set of all feasible homology classes can be described by the solution space of a set of linear inequalities. This solution space is called the *feasible homology polytope* Φ.

Similarly $H(G : s, t)$ is the space of homology classes of flows. As before, $H(G : s, t) = Z(G : s, t)/B(G)$ is isomorphic to $R^{|E|-|V|-|F|+1} = R^{2g+1}$ by Euler's formula. A homology class of flows is *feasible* if at least one of its elements is feasible.

The homology space $H(G)$ can be generated by $2g$ cycles in independent homology classes as it is isomorphic to R^{2g}. For any fixed basis $(\gamma_1, \ldots, \gamma_{2g})$, a *basic circulation* is a circulation ϕ that can be expressed as $\sum_{i=1}^{2g} \phi_i \cdot \gamma_i$ for some real coefficients $\phi_1 ... \phi_{2g}$. Every circulation in G is homologous to one basic circulation. Likewise $H(G; s, t)$ can be generated by homology classes of $2g + 1$ curves each of which is a (s, t) path or cycle. See *Lemma 3.4* of Chambers, Erickson and Nayyeri [8] for further details on the computation of one such basis.

2.5 Convex functions and subgradients

A *convex set* is a set of points in a vector space such that given any two elements of the set, the entire line segment joining the two points is also in the set. Let $A \subset R^n$ be a convex set. A function $f : A \to R$ is *convex* if for all $\theta \in [0, 1]$, and for all x and y in A, we have $f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$.

The *graph* of f is $Gr(f) = \{(x, y) \in A \times R | f(x) = y\}$. The *epigraph* of f is $Epi(f) = \{(x, y) \in A \times R | f(x) \leq y\}$ which is the set if points on or above the graph of f. The *hypograph* of f is $Hyp(f) = \{(x, y) \in A \times R | f(x) \geq y\}$ which is set of points on or below the graph of f.

A function f is convex if and only if $Epi(f)$ is a convex set. Similarly, a function f is *concave* if and only if $Hyp(f)$ is a convex set. Also f is convex if and only if $-f$ is concave.

A vector $g \in R^n$ is a *subgradient* of a convex function $f : A \to R$ at $x \in A$ if $f(z) \geq f(x) + g^T(z - x)$ for all $z \in A$. If f is differentiable, then its gradient $\nabla f(x)$ is its only subgradient, but nondifferentiable functions can also have subgradients. A function f is called *subdifferentiable* at x if it has a subgradient at x. The set of subgradients of f at the point x is called the *subdifferential* of f at x, and is denoted $\partial f(x)$. A function is called subdifferentiable if it is subdifferentiable at all points $x \in A$.

Grötschel, Lovasz and Schrijver [22] define the following *strong separation problem* for convex sets: Given $z \in R^n$, decide whether z is a member of the closed convex set A, and if not, find a hyperplane that separates z from A. To be more precise, we need to find a vector g such that $g \in R^n$ and $g^T z >$

$\max\{g^T y | y \in A\}$.

Chapter 3

History and related material

The maximum flow problem is the problem of finding the maximum feasible flow from source to target in a network where the edge capacities are given. Maximum flows and minimum cuts gained importance during the Cold War when they were used to study the railway and transportation networks. The seminal work in this area was by Ford and Fulkerson [15] in 1956, who credit Harris with first posing the maximum flow problem on a railway network. Later, in their book *Flows in Networks* [16], they give more details on the origin of the problem and cite a secret report by Harris and Ross [23] as their original source for the problem. Schrijver, in his review of the history of the transportation and maximum flow problem [39], mentions this secret report as evidence of early interest in computing a minimum cut. He also mentions the Soviet interest in cargo transportation planning [42] as early as the 1930's.

The classical maximum flow problem is defined as follows. We are given a graph $G(V, E)$, a specific source vertex s, and a sink vertex t. We are also given a capacity function. The problem is to push as much flow from s to t as possible, without violating any edge capacity. The maximum flow problem can be modeled by a $|V| = n$ dimensional linear program in $|E| = m$ variables. As such, the simplex method [10] to solve linear programs is the earliest known algorithm for this problem. In fact, Ford and Fulkerson [15] point out that the maximum flow problem "can be set up as a linear programming problem with as many equations as there are vertices and hence can be solved by the simplex method". They motivate the uppermost path algorithm by pointing out that for st-planar graphs, the uppermost flow algorithm is simpler and can be computed by hand a lot easier than solving a linear programming problem.

Since then algorithms that solve the maximum flow problem have involved looking at general graphs as well as (faster) algorithms for more specific families of graphs. For general graphs, Table 3.1 compares the running times of some of the better known algorithms.

The last two algorithms require the edge capacities to have integer capacities less than some integer U. For sparse graphs, the best of these algorithms, by Goldberg and Rao [18], runs in time $O(n^{3/2} \log n \log U)$.

Algorithms have also being developed to solve the maximum flow problem in

Algorithm	Author(s)	Running Time
Augmenting path	Ford and Fulkerson [15]	$O(mf^*)$
Edmonds-Karp	Edmonds and Karp [12]	$O(nm^2)$
Blocking flow	Dinic [11]	$O(n^2m)$
General push-relabel	Goldberg and Tarjan [19]	$O(n^2m)$
Push-relabel with dynamic trees	Sleator and Tarjan [41]	$O(nm\log(n^2/m))$
Binary blocking flow	Goldberg and Rao [18]	$O(\min(n^{2/3}, m^{1/2})$ $m\log(n^2/m)\log U)$
Generalised flow using interior points	Daitch and Spielman [9]	$O(m^{3/2}\log U\log^k m)$

Table 3.1: Some maximum flow algorithms and their running time

planar graphs, partly motivated by the fact that planar graphs are used to model real world situations and by the fact that these algorithms are comparatively faster and simpler than algorithms for general graphs. The first maximum flow algorithm was the uppermost path algorithm for st-planar networks. This approach has since been generalized to general planar graphs.

Ford and Fulkerson [15] show that in an st planar graph G, there is always a s to t path that meets each cut of G once. When the graph is drawn on the plane with s and t at the left and right extremes of the outer face, the uppermost path from s to t is one such path. They augment the flow along the this uppermost path, deleting any saturated edges and updating the residual capacities of the remaining edges. Note that unlike a general augmenting path algorithm, they can delete a saturated edge entirely. They repeat the procedure until there is no path from s to t in the residual graph.

Later in the same paper, Ford and Fulkerson point out the duality between the problem of finding a shortest path from s to t in G, and finding the minimum cut in the dual of G between two particular vertices. They propose the following algorithm to find the shortest path between s and t in an st planar graph. Add an edge $t \to s$ and let p and q be the regions separated by ts. Run the uppermost path algorithm in dual network, and a minimum cut found from the deleted edges is dual to a shortest path in the primal. Dijkstra's algorithm to find a single source shortest path in a given graph first appeared in 1959, a few years after the publication of Ford and Fulkerson's work [15]. Interestingly, if we keep track of the dual edges to the saturated edges, we can see that the uppermost path algorithm implements Dijkstra's shortest path algorithm in the dual graph.

Ford and Fulkerson's uppermost flow algorithm, is also referred to as Berge's algorithm [1], by whom it was simultaneously developed. Itai and Shiloach [29] showed how to implement the uppermost path algorithm in $O(n\log n)$ time. Hassin [24] demonstrated that the maximum st planar flow can itself be found by building a shortest path tree in the modified dual described by Ford and Fulkerson, showing that for st planar graphs, finding the maximum flow is equivalent

to just one shortest path computation in the dual. As in Ford and Fulkerson's method [15], add an edge of very large capacity from t to s. Let p and q be faces adjacent to the added edge in G. Let T^* be a shortest path tree in G^* rooted at p^* where the cost of each dual edge e^* is the capacity of the corresponding primal edge e Define $\alpha(f)$ to be the length of the shortest path from p^* to f^* in T^*. Let ϕ be the boundary of 2-chain α, ie $\phi(u \to v) = \alpha(f_r) - \alpha(f_l)$, where f_l is the face to the left of edge $u \to v$ and f_r the face to the right. Edges in G that are dual to edges on shortest path tree T^*, are saturated. For other edges, $\phi(u \to v) = \alpha(f_r) - \alpha(f_l) \leq c(u \to v)$. Thus ϕ corresponds to a feasible circulation. The shortest p^* to q^* path does not include the added edge and is dual to a minimum cut in the original primal graph G. Since ϕ(when restricted to the original graph) sends a flow of value equal to the minimum cut in the original graph, it must be a maximum flow in the original graph. When Frederickson [17] and Henzinger et al. [27] came up with faster shortest path algorithms for planar graphs, Hassin's result immediately gave a faster maximum flow algorithms for *st* planar graphs. Henzinger's algorithm runs in $O(n)$ time.

For undirected planar networks with s and t on different faces, Itai and Shiloach [29] described a $O(n^2 \log n)$ method to find a flow with a given value, or report that no such flow exists. Itai and Shiloach's algorithm was based on the correspondence between shortest paths and minimum cuts. Reif [38] developed a divide and conquer method to compute a minimum cut, and thus the maximum flow value, in $O(n \log^2 n)$ time. Reif's result was extended by Hassin and Johnson [25] to compute the actual maximum flow in $O(n \log n)$ additional time, so that the overall time was $O(n \log^2 n)$. Frederickson [17] later improved Reif's algorithm to $O(n \log n)$ time using a faster shortest path algorithm. Frederickson's result [17] and the shortest path algorithm of Henzinger et al. [27] can also be used with Hassin and Johnson's algorithm to compute the maximum flow in $O(n \log n)$ time for undirected graphs.

Johnson and Venkatesan [30] described a divide and conquer algorithm that finds a flow of value v in a *directed* planar graph in time $O(n\sqrt{n} \log n)$, using recursive separator decompositions. Venkatesan [43] also 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 [35]. Interestingly enough, Tolstoi [42], an early Russian researcher, had included many solution approaches, one of which was the observation that an optimal flow did not include any negative cycles in the residual graph, though he did not give any sufficient conditions for optimality. Weihe [44] gives a planar maximum-flow algorithm that runs in $O(n \log n)$ time, provided the input graph satisfies a certain connectivity condition. However, the fastest known algorithm to enforce this condition runs in $O(n^2)$ time [2]. Borradaile and Klein [4] describe an $O(n \log n)$-time algorithm to find maximum flows in arbitrary directed planar graphs. Inspired by the uppermost path algorithm, they use an augmenting

path algorithm, where in each iteration the leftmost path in the residual graph is chosen as the augmenting path. A leftmost path in the residual is defined to be a path that does not have clockwise residual cycles in its dual. They start with a "leftmost circulation" and augment the flow with a leftmost path each time, so that at any point in time their algorithm computes the leftmost flow of all flows of that same value. Thus, each leftmost flow is a canonical choice to represent the class of flows of equal value.

Earlier, Khuller, Naor and Klein [32] showed that circulations in planar graphs defined a finite distributive lattice, and the unique minimum of this lattice was called the leftmost circulation. The residual graph of a leftmost circulation did not have any clockwise residual cycles. It is also the boundary circulation of a 2-chain that corresponds to shortest path distances of a face from the external face of the given embedding. The last definition of leftmost circulation can be extended to a higher genus setting. In higher genus graphs that have s and t on the same face, we can set p^* the dual vertex corresponding to the face to the left of $t \to s$, and define a boundary circulation that corresponds to a 2-chain defined by distances given by a shortest path tree based at p^*. This boundary circulation is a canonical choice to represent the class of all homologous circulations.

Finally, Erickson [14] formulates the maximum flow problem in planar graphs as a parametric shortest path problem in the dual graph and gives an $O(n \log n)$ algorithm using dynamic tree data structures to maintain the shortest path tree. The resulting algorithm is in fact identical to the Borradaile and Klein algorithm [4]. He also observes that on a torus, a similarly structured algorithm can take $\Omega(n^2)$ time.

Chambers, Erickson, and Nayyeri [8] show that the maximum flow problem for graphs that can be drawn on a surface of genus g can be formulated as a linear programming problem in $2g + 1$ dimensions. Standard methods to solve linear programming problems are then be used to get an $O(g^7 n \log^2 n \log C)$ running time, where C is the sum of the edge capacities of the original graph.

3.1 Our results

Because both the uppermost path algorithm and Hassin's dual shortest path formulation (both of which work on *st* planar graphs) have inspired algorithms for maximum flow in planar graphs, we look at surface embedded graphs where both s and t are on the same face, to study any solution structure that could be exploited.

Formally, we are given a graph G, embedded on a surface of genus g and vertices s and t on the same face. We wish to find the maximum flow from s to t. Add an infinite-capacity edge $t \to s$ to G to obtain a new graph G'. A maximum flow in G is equivalent to a feasible circulation in G' maximizing the flow through the new edge $t \to s$.

Using the idea of feasible homology classes of circulations as introduced by Chambers, Erickson, and Nayyeri [8], we show that we can formulate this circulation problem as a convex programming problem in $2g$ dimensions. We first show that a single shortest path computation in the dual residual graph tells us whether a given homology class contains any feasible circulations. The separation oracle used to determine if an entire class of homologous circulations is feasible or not is the same as that used by Chambers, Erickson, and Nayyeri [8] to determine if an entire class of homologous flows is feasible or not, with the only difference being that the homologous circulations are in $H(G')$ which is isomorphic to R^{2g}, while the homologous flows are in $H(G; s, t)$ isomorphic to R^{2g+1}. If a class of homologous circulations is found to be feasible, we also find the optimal feasible circulation ψ in that class. This is the canonical representative of this class of homologous circulations and we can use this circulation to define a flow of value $\psi(t \to s)$. To find the maximum flow we need to maximize $\psi(t \to s)$ over all basic circulations. As $\psi(t \to s)$ can be computed by a convex function, we solve the problem using the ellipsoid method.

In Chapter 5, we describe the general ellipsoid method to solve a convex programming problem, and mention the properties that must be satisfied by the feasible region and the separation oracle. We also need a way to find a subgradient of the objective function at any given point. For more details about these results for the general ellipsoid method, please refer to the textbook by Grötschel, Lovasz, and Schrijver [22]. A similar analysis for using the ellipsoid method for linear programming was given by Chambers, Erickson, and Nayyeri [8]. Finally, we plug in the details for our specific problem and find the running time to be $O(g^6 n \log^2 n \log^2(Cn))$.

Chapter 4

Homology flows and circulations

Recall that we are given an embedding of a graph $G(V, E)$ on a surface of genus g, a capacity function $c : E \to R$, and two vertices s and t on the same face. We are to find the maximum flow from s to t. We add an edge $t \to s$ of infinite capacity and call the modified graph G'. The face to the left of $t \to s$ is p and the face to the right is q. F is the set of faces in G'.

The vector space $H(G')$ of homology classes of circulations in G' has dimension $2g$. We define a basis for $H(G')$ as follows.

Let p_0 be the path from s to t along the boundary of face p. Let T be a spanning tree of G' that includes the path p_0. Let C^* be a spanning tree of $(G' \setminus T)^*$ including $t \to s$. Euler's formula implies that there are $2g$ edges $\{e_1, \ldots, e_{2g}\}$ in $E \setminus (T \cup C)$. For each index i, let γ_i denote the unique cycle in $T \cup e_i$.

Lemma 1 *The cycles $\{\gamma_1, \ldots, \gamma_{2g}\}$ form a basis for $H(G')$ and $H(G)$. Along with p_0, they form a basis for $H(G'; s, t)$ and $H(G; s, t)$.*

Proof: That $(\gamma_1, \ldots, \gamma_{2g})$ forms a basis for $H(G')$ follows from *Lemma 3.4* of Chambers, Erickson, and Nayyeri [8]. The same cycles also form a basis for $H(G)$, because we select T and C^* such that no cycle γ_i contains the edge $t \to s$. The proof of the other claim is similar. ∎

Recall that a basic circulation is a circulation ϕ that can be expressed as $\sum_{i=1}^{2g} \phi_i \cdot \gamma_i$ for some coefficients $\phi_1 \ldots \phi_{2g}$. We can now also state that a circulation is basic if and only if $\phi(e) = 0$ for every cotree edge e in C.

We need to find the feasible circulation that maximizes the flow sent through $t \to s$. In Theorem 3 we show that given a basic circulation ϕ on G', we can find a circulation ψ homologous to ϕ such that $\psi(t \to s)$ is maximized using one shortest path calculation on G'^*_ϕ. Finally we show how to set up our problem as a convex programming problem.

4.1 Feasible homology classes of circulations

Theorem 2 *There is a feasible circulation in G homologous to a given circulation ϕ if and only if the dual residual network G^*_ϕ has no negative cost cycles.*

Proof: Suppose there is some feasible circulation homologous to the given circulation ϕ. Let λ^* be an arbitrary cycle in G^*_ϕ and let λ be the corresponding cocycle in G.

Let $c(\lambda) = \sum_{e\in\lambda} c(e)$ denote the total capacity of λ and let $\phi(\lambda) = \sum_{e\in\lambda} \phi(e)$, denote the total flow through λ. The residual capacity of λ is, $c_\phi(\lambda) = c(\lambda) - \phi(\lambda) = \sum_{e\in\lambda} c(e) - \sum_{e\in\lambda} \phi(e)$. If this residual capacity is negative, the total flow through λ is greater than its total capacity and so ϕ is not feasible.

For any 2-chain $\alpha : F \to R$ in G, we have

$$\partial\alpha(\lambda) := \sum_{e\in\lambda} \partial\alpha(e) = \sum_{f_l \uparrow f_r \in \lambda} \alpha(f_l) - \alpha(f_r) = \sum_{f_l^* \to f_r^* \in \lambda^*} \alpha(f_l^*) - \alpha(f_r^*) = 0.$$

The last step follows from the fact that λ^* is a cycle.

Any circulation ψ homologous to ϕ can be written as $\psi = \phi + \partial\alpha$ for some 2-chain α. We know $\partial\alpha(\lambda) = 0$, hence $\psi(\lambda) = \phi(\lambda)$, which in turn means that $c_\psi(\lambda^*) = c_\phi(\lambda^*)$. Thus, if G^*_ϕ has a negative cycle, no circulation homologous to ϕ is feasible.

Suppose G^*_ϕ has no negative cost cycles. For any face f of G, let $\alpha(f)$ denote the shortest path distance from p^* to f^* in G^*_ϕ.

Consider the circulation $\psi = \phi + \partial\alpha$, which is homologous to ϕ. The cost of an edge $f \uparrow g$ in G_ϕ is $c_\phi(f \uparrow g) = c_\phi(f^* \to g^*) \geq \alpha(g) - \alpha(f)$, since α is defined by shortest path distances.

Thus,

$$\psi(f \uparrow g) = \phi(f^* \to g^*) + (\alpha(g) - \alpha(f)) \leq \phi(f^* \to g^*) + c_\phi(f^* \to g^*) = c(f \uparrow g)$$

We conclude that ψ is feasible. ∎

4.2 Best circulation in a feasible homology class

Theorem 3 *Let ϕ be a basic circulation in G' whose homology class is feasible. For all faces f of G', let $\alpha(f)$ be the shortest path distance from p^* to f^* in G'^*_ϕ. Let $\psi = \phi + \partial\alpha$. For any circulation χ homologous with ϕ (or with ψ), we have $\chi(t \to s) \leq \psi(t \to s)$.*

Proof: Let λ^* be the shortest path in G'^*_ϕ, from p^* to q^*. The corresponding primal subgraph λ is actually a cocycle in G. The effect of removing the edge $t \to s$ from G'^*_ϕ is to fuse the path λ^* into a cycle in G^*. From *Lemma 1* we know that we can use the same homology basis for $H(G)$ and $H(G')$. Thus, ϕ is also a basic circulation in G.

We are given that $\psi = \phi + \partial\alpha$. Let $\chi = \phi + \partial\beta$ be a circulation homologous to ϕ in G', where β is some 2-chain on the faces of G'. We need to show that either

$\psi(t \to s) \geq \chi(t \to s)$ or χ is infeasible. Assume χ is feasible since otherwise we are done.

The edges of λ are saturated by ψ as they correspond to the edges of a shortest path tree in the dual. Thus, $c(\lambda) = \psi(\lambda)$.

Given the feasible circulation ψ in G', we can construct a feasible flow in G of value $\psi(t \to s)$ simply by dropping edge $t \to s$. We will denote this flow by ψ_G. Also, $\psi_G \cong \phi + \psi(t \to s) \cdot p_0$. In other words, ψ_G is a feasible flow in G and its representative in the flow homology space of G is $\langle \phi_1, \ldots, \phi_{2g}, \psi(t \to s) \rangle$.

The total flow through the edges of any cocycle is the same for any two homologous flows. Thus,

$$\psi_G(\lambda) = \phi(\lambda) + \psi(t \to s) \cdot p_0(\lambda).$$

Similarly, starting with the *feasible* circulation χ in G' we construct a *feasible flow* χ_G in G whose representative in the flow homology space of G is $\langle \phi_1, \ldots, \phi_{2g}, \chi(t \to s) \rangle$. As before, we have

$$\chi_G(\lambda) = \phi(\lambda) + \chi(t \to s) \cdot p_0(\lambda).$$

We also have $\psi(\lambda) = \psi_G(\lambda)$ because λ does not include the edge $t \to s$.

The residual capacity of cocycle λ for the flow χ_G is:

$$\begin{aligned} c_{\chi_G}(\lambda) &= c(\lambda) - \chi_G(\lambda) \\ &= \psi(\lambda) - \chi_G(\lambda) \\ &= \psi_G(\lambda) - \chi_G(\lambda) \\ &= (\psi(t \to s) - \chi(t \to s)) \cdot p_0(\lambda). \end{aligned}$$

The first edge on λ^* is dual to an edge on p_0. So $p_0(\lambda) = 1$.

Thus, $c_{\chi_G}(\lambda) = \psi(t \to s) - \chi(t \to s) \geq 0$, because χ is feasible. We conclude that $\psi(t \to s) \geq \chi(t \to s)$. ∎

Corollary 4 *Given a basic circulation ϕ, we can either compute a feasible circulation ψ homologous to ϕ maximizing $\psi(t \to s)$, or determine that no circulation homologous to ϕ is feasible, in $O(g^2 n \log^2 n)$ time.*

Proof Given a basic circulation ϕ, we can either find the shortest path tree based at p^* or a negative cycle in G'^*_ϕ, by a generalizing an algorithm by Klein, Moses, and Weimann [33]. ∎

4.3 Optimization

Let Φ be the feasible homology polytope. Let $F : \Phi \to R$ be a function such that for any basic circulation ϕ, $F(\phi)$ is the maximum of $\psi(t \to s)$ over all feasible circulations ψ homologous to ϕ.

Theorem 3 implies that $F(\phi) = \min_i P_i(\phi_1, \ldots, \phi_{2g})$, where $P_i(\phi_1, \ldots, \phi_{2g})$ is the length of parameterized path P_i, which is the i^{th} path from p^* to q^* in $G_\phi^{'*}$; $P_i : \Phi \rightarrow R$ is a linear function.

Theorem 5 *$F : \Phi \rightarrow R$ is a concave, piecewise linear function.*

Proof: The length of any path P_i in $G_\phi^{'*}$ depends linearly on $\phi_1, \ldots, \phi_{2g}$, so we can write $P_i = a_{i,1}\phi_1 + \ldots + a_{i,2g}\phi_{2g} + b_i$. Thus F is a piecewise linear function.

The hypograph of F is the intersection of the hypographs of P_i. The hypograph of each P_i is a linear halfspace. As the intersection of convex sets, the hypograph of F is also a convex set, and thus F is concave. ∎

We have reduced computing the maximum flow in graph G to finding a basic circulation whose dual residual graph has no negative cycle and that maximizes the concave function $F : \Phi \rightarrow R$. Thus, we have the following convex programming problem,

$$\text{maximize } F(\phi)$$

$$\text{subject to } \phi(\lambda) \leq c(\lambda), \text{for every directed cocycle} \lambda \text{ in } G^{'}.$$

Unfortunately, this convex program appears to be too complex to solve directly; there could be $n^{O(g)}$ non-redundant constraints. Instead we must solve it using implicit methods, such as the ellipsoid method [21] or multidimensional parametric search. In the next chapter we discuss how we can use the ellipsoid method to solve our problem.

Chapter 5

Ellipsoid method

The ellipsoid method was originally devised for nonlinear nondifferentiable optimization. Shor [40] first stated the ellipsoid method as known today. It was then modified by Khachiyen [31] to give the first polynomial time algorithm for linear programming. Later, it was further modified to solve linear programs implicitly. We now give a brief sketch of the central-cut ellipsoid method. Much of this material is from Chambers, Erickson and Nayyeri [8] and Grötschel, Lovasz and Schrijver [22] as well as from lecture notes made available by Boyd [6].

Grötschel, Lovasz, and Schrijver [22] offer the following example to explain the basic idea behind the ellipsoid method. Suppose we know that there is exactly one lion in the Sahara and that it can exist only in some specific region of the Sahara where environmental conditions are satisfactory. We want to catch the lion. We start by fencing in the entire Sahara. From our current location we have a method named Oracle, to separate the currently fenced in portion into two halves: one that does not contain the lion and therefore can be safely discarded and the other half which must contain our lion. Having discarded one half of the region, we need to refence the other half quickly. Thus, we always have the lion within our fenced in region, which is shrinking. After a finite number of steps we will catch our lion, as the fenced in region will become too small for the lion to hide.

A few words about the method Oracle that we used to refine the region we need to search. Given an arbitrary point x, the Oracle returns a halfspace h whose bounding hyperplane contains x, such that the lion lies in h.

5.1 Generic reduction from convex programming to linear programming

Here we describe a generic reduction from convex programs with piecewise linear objective functions to linear programming.

Let $F : \Phi \to R$ be a concave, piecewise linear function over the convex set Φ, we formulate this generic problem as follows:

$$\begin{array}{ll} \text{maximize} & f(\phi) \\ \text{such that} & \phi \in \Phi \end{array}$$

We rewrite the convex program above as follows:

$$\begin{array}{ll} \text{maximize} & z \\ \text{subject to} & (\phi, z) \in Hyp(f) \end{array}$$

which is a standard linear program if F is a piecewise linear function and Φ is a polytope.

Applying this tranformation to our convex program, we recover the LP formulation of Chambers, Erickson and Nayyeri [8].

5.2 Ellipsoid method for convex programs

Now we describe the ellipsoid method as it can be applied to solve a convex optimization problem,

$$\begin{array}{ll} \text{minimize} & F(\phi) \\ \text{subject to} & \phi \in \Phi \end{array}$$

Let ϕ_{opt} denote the optimum point of Φ. In general, the optimum vertex need not be unique, so we can apply standard perturbation techniques described by Grötschel, Lovasz and Schrijver [22] to perturb the objective function slightly such that the perturbed objective function has a unique optimum, which is also optimal for the original function. We assume that ϕ_{opt} is integral, but prove this fact for our problem. In general, we may need to scale the problem to ensure integrality.

For any $\epsilon > 0$, let $\Phi_\epsilon = \{\phi \in \Phi | F(\phi) \leq F(\phi_{opt}) + \epsilon\}$.

We need to choose ϵ such that Φ_ϵ lies in a ball of radius $1/4$ so that Φ_ϵ contains no integral points other than ϕ_{opt}.

Calculate a bounding ellipsoid E_0 that is guaranteed to contain Φ. The ellipsoid algorithm maintains an ellipsoid E_i that is guaranteed to contain ϕ_{opt}.

The separation oracle is an algorithm that decides correctly whether a given point belongs to Φ or not. If not, it returns a hyperplane separating the point from Φ.

The subgradient oracle is an algorithm that calculates a subgradient g of F at a given feasible point x. The subgradient defines a halfspace $\langle g, \phi \rangle \leq \langle g, x \rangle$ that must contain ϕ_{opt} because $F(\phi_{opt}) \geq F(x) + \langle g, \phi_{opt} \rangle - \langle g, x \rangle$.

We can use a single oracle, $Oracle_\phi$, to return a violated constraint if the given point is infeasible, or return the halfspace described by the subgradient if the point is feasible. In either case, the oracle returns a halfspace h containing ϕ_{opt}. The ellipsoid method then updates E_{i+1} to be the minimum volume ellipsoid containing $E_i \cap h$.

At each iteration, $Oracle_\phi$ is queried at the center of the current ellipsoid E_i. Thus the volume of the ellipsoid shrinks by a constant factor $e^{1/O(d)}$ each time, where d is the dimension of Φ.

Finally, when the volume of E_i is less than the volume of Φ_ϵ we stop and

round the last feasible query point, y, to the integer grid. Let $\Phi'_y = \{z|F(z) \leq F(y)\}$. We have $\Phi'_y \subset E_i$ and thus $\text{Vol}(\Phi_y) \leq \text{Vol}(E_i)$. When $\text{Vol}(E_i) < \text{Vol}(\Phi_\epsilon)$, then $\text{Vol}(\Phi'_y) < \text{Vol}(\Phi_\epsilon)$ which implies that $\Phi'_y \subset \Phi_\epsilon \subset B(1/4)$, and we can round y to get the optimum point.

We assume that the oracle can be modeled by a linear decision tree. Grötschel, Lovasz and Schrijver [22] show why this is necessary. Let T_s denote the number of arithmetic operations required for a single $Oracle_\phi$ query. The running time of this method depends on the volume of the initial ellipsoid and on a lower bound for the volume of Φ_ϵ, as well as the running time of the $Oracle_\phi$. The method requires $I = O(d \log \delta)$ iterations where d is the dimension and $\delta = (volE_0/vol\Phi_\epsilon)$ when Φ is represented by linear constraints Grötschel, Lovasz and Schrijver [22]. Each iteration requires T_s arithmetic operations by $Oracle_\phi$, plus $O(d^2)$ arithmetic operations to compute the new ellipsoid with upto $O(i)$ bits of precision being sufficient to preserve correctness in iteration i Grötschel, Lovasz and Schrijver [22]; the running time of the general method is thus $O(T_s d^2 \log^2 \delta + d^4 \log^2 \delta \log^2(d \log \delta))$ as also shown by Chambers, Erickson and Nayyeri [8].

5.3 Ellipsoid method for our convex program

Assume that the vertices of Φ have integral coordinates. This condition holds if Φ is the feasible region of a maximum flow, or more generally if the constraint matrix defining Φ is totally unimodular. In particular, this means that ϕ_{opt} is integral. In our case, Φ is the projection of the feasible homology flow polytope which has integral coordinates as shown by Chambers, Erickson, and Nayyeri [8]. The projection maps a feasible class of homologous flows to a feasible class of homologous circulations and is achieved by dropping the last coordinate. As such, the vertices of Φ are integral.

We express our problem as,

$$\text{maximize } F(\phi)$$

$$\text{subject to } \phi(\lambda) \leq c(\lambda), \text{for every directed cocycle} \lambda \text{ in } G^{'}.$$

We negate the concave function to get a convex function (and abuse notation by reusing the variable F) to get:

$$F(\phi_1, ..., \phi_{2g}) = \max_i F_i(\phi_1, ..., \phi_{2g})$$

where $F_i(\phi_1, ..., \phi_{2g}) = -P_i(\phi_1, ..., \phi_{2g})$. Thus, F is now a convex, piecewise linear function which we want to minimize over the feasible region.

The convex program to be solved can now be written in standard form as:

$$\begin{array}{ll} \min_{(\phi_1,\ldots,\phi_{2g})} & \max_i F_i(\phi_1,\ldots,\phi_{2g}) \\ \text{subject to} & \phi(\lambda) \leq c(\lambda), \qquad \text{for every cocycle} \quad \lambda \quad \text{in} \quad G' \end{array}$$

We use the ellipsoid method as described in the previous section to solve this problem. The convex function in this problem is $F(\phi) = \max_i F_i(\phi)$. The bounding ellipsoid E_0 is a ball of radius $C\sqrt{2g}$ centered at the origin. We set $\epsilon = 1/(4C\sqrt{2g})$ so that Φ_ϵ fits in a ball of radius $1/4$. Also $Vol(\Phi_\epsilon) \geq B(\epsilon/\sqrt{2g}n)$ which implies that the number of iterations is $O(g^2 \log(C))$. The running time of $Oracle_\phi$ is $O(g^2 n \log^2 n)$ and the overall running time of the algorithm is $O(g^6 n \log^2 n \log^2(C))$. In subsequent theorems we show why these statements are true. $Oracle_\phi$ for this problem uses the algorithm described earlier in *Corollary* 4 at the query point, ϕ. If we get a negative cycle, that is the violated constraint. If the point is feasible, the shortest path from p^* to q^* defines a function F_i whose subgradient we can calculate. We now describe how to find this subgradient.

Let k be an index at which $F_k(\phi) = F(\phi)$. Let $g = \nabla F_k(\phi)$.

We have $F_k(x) = F_k(\phi) + \langle g, x\rangle - \langle g, \phi\rangle$, at all $x \in \Phi$. Also $F(x) \geq F_k(x)$ as F is the pointwise maximum of the functions F_i. Putting these together we have, $F(x) \geq F_k(\phi) + \langle g, x\rangle - \langle g, \phi\rangle = F(\phi) + \langle g, x\rangle - \langle g, \phi\rangle$. Thus, $g \in \partial F(x)$.

Theorem 6 *The ellipsoid method finds the max flow in an st embedded graph in* $O(g^6 n \log^2 n \log^2(Cn))$.

Proof: The time to run *Oracle* once is T_s is $O(g^2 n \log^2 n)$. The number of iterations, I is $O(g \log(VolB(C\sqrt{2g})\ /VolB(\epsilon/\sqrt{2g}n)))$ which is $O(g \log(2Cgn/\epsilon)^{2g})$ which in turn is $O(g^2 \log(8\sqrt{2}C^2 n g^{3/2}))$. Assuming that $g = O(\sqrt{n})$, the number of iterations is $O(g^2 \log(C^2 n^2)) = O(g^2 \log(Cn))$.

As the Oracle takes time $O(g^2 n \log^2 n)$, and uses arithmetic operations which take time proportional to the number of bits, the total time from running the Oracle is $O(g^6 n \log^2 n \log^2(Cn))$. The total time spent maintaining the ellipsoid E_i is $O((gI \log I)^2)$ which is $O((g^3 \log(Cn) \log\log(Cn))^2)$ but this is dominated by the rest of the algorithm if $C = O(2^n)$. ∎

5.4 Required theorems

Lemma 7 *Φ lies in a ball of radius $C\sqrt{2g}$.*

Proof : The total capacity of any cycle is at most C, and so $-C \leq \phi_i \leq C$. Thus, the feasible region Φ lies in the hypercube $[-C, C]^{2g}$ which lies in a ball of radius $C\sqrt{2g}$ centered at the origin. ∎

Lemma 8 *For any $\epsilon \in [0, 1]$, Φ_ϵ is the result of scaling Φ_1 by a factor of ϵ about ϕ_{opt}.*

Proof: We shoot a ray from ϕ_{opt} into some direction and let x_1 be the point the ray meets Φ_1 and x_ϵ be the point where the ray meets Φ_ϵ.

Suppose that $F(x_\epsilon) = F_i(x_\epsilon)$ and $F(x_1) \neq F_i(x_1)$. Then there is a y in the interior of Φ_1 such that $F_i(y) = F_k(y) = F(y)$ for some $i \neq k$.

Let $R_{ik} = \{x|F(x) = F_i(x) = F_k(x)\}$. As y is strictly in the interior of Φ_1, $F(y) < F(\phi_{opt}) + 1$. Every vertex in R_{ik} is the projection of a vertex of the feasible homology flow polytope Φ' and is therefore integral.

Let r_{ik} be the optimal vertex of R_{ik} such that $F(r_{ik})$ is minimized. Since r_{ik} is not ϕ_{opt}, $F(\phi_{opt}) < F(r_{ik})$. $F(r_{ik})$ is integral, but that means $F(\phi_{opt}) < F(r_{ik}) < F(\phi_{opt}) + 1$, which is not possible. We conclude that $F_i(x_1) = F(x_1)$ and that as F_i is linear Φ_ϵ is a scaled copy of Φ_1 about ϕ_{opt}. ∎

Corollary 9 *If $\epsilon = 1/4C\sqrt{2g}$, Φ_ϵ lies in a ball of radius $1/4$.*

Proof: If $\epsilon = 1/4C\sqrt{2g}$, Φ_ϵ is the result of scaling Φ_1 by a factor of ϵ about ϕ_{opt}. Since Φ_1 lies in a ball of radius $C\sqrt{2g}$, this means that Φ_ϵ lies in a ball of radius $C\epsilon\sqrt{2g}$. Thus, Φ_ϵ lies in a ball of radius $1/4$. ∎

Lemma 10 *$Vol(\Phi_\epsilon) \geq B(\epsilon/\sqrt{2g}n)$.*

Proof: Recall that $\Phi_i = \{x|F(x) \leq F(phi_{opt}) + i\}$. Φ_ϵ is a convex set, bounded by constraints in $2g$ dimensions. Recall that $F(\phi) = \max_i F_i(\phi)$ where $F_i(\phi)$ is the negated length of some path from p^* to q^* in the dual residual graph. Each function F_i is affine; that is, $F_i(\phi) = \langle a_i, \phi\rangle + c_i$ for some vector $a_i \in R^{2g}$ with $||a_i|| \leq n\sqrt{2g}$ and some scalar $c_i \in R$.

Let $L_i = F_i^{-1}(F_i(\phi_{opt}) + \epsilon)$; this set is a hyperplane in R^{2g} with equation $\langle a_i, \phi\rangle + c_i = F_i(\phi_{opt}) + \epsilon$.

The boundary of Φ_ϵ is contained in the union of all hyperplanes L_i such that $F_i(\phi_{opt}) = F(\phi_{opt})$.

For each i, let r_i denote the minimum distance from ϕ_{opt} to L_i. If $F_i(\phi_{opt}) = F(\phi_{opt})$, we have $r_i = \epsilon/\,||a_i|| \geq \epsilon/n\sqrt{2g}$.

Thus, ϕ_ϵ contains a ball of radius $\epsilon/\sqrt{2g}n$ centered at the origin. ∎

Lemma 11 *The vertices of Φ have integral coordinates.*

Proof: Let Φ' be the feasible flow polytope in $H(G; s, t)$. Chambers, Erickson, and Nayyeri [8] show that the vertices of the feasible homology flow polytope Φ' have integral coordinates. Φ is a projection of Φ', with the projection map simply dropping the last coordinate. Thus every vertex of Φ is a projection of a vertex of Φ' and hence has integral coordinates.

∎

Lemma 12 *The objective function attains its maximum at an integral point.*

Proof: Chambers, Erickson, and Nayyeri [8] show that the maximum flow is attained at an integral vertex of Φ', say m. Consider the vertex of Φ to which

m is projected. Using Theorem 3 at this vertex, we find circulation maximizing $\psi(t \rightarrow s)$ and thus the maximum valued flow which is integral as the dual residual graph considered by theorem 3, has integral edge weights. ∎

Theorem 13 *When $g = 1$, the number of non-redundant constraints is $O(n^2)$.*

Proof: When a non-redundant constraint is satisfied, it just means that one of the constraints of the convex program P is satisfied with equality. ie. $c(\lambda) - \psi(\lambda) = 0$. ie. $\psi(\lambda) = c(\lambda) = d$ say.

Now $\psi = x\beta + y\gamma$ and $\psi(\lambda) = x(\beta(\lambda)) + y(\gamma(\lambda))$ where $\beta(\lambda)$ = Number of edges that λ shares with β and $\gamma(\lambda)$ = Number of edges that λ shares with γ

Now as λ corresponds to co-cycles that are dual to simple cycles, and as both β and γ which are a set of cycles forming the homology basis are simple, $\beta(\lambda) \leq n$ and $\gamma(\lambda) \leq n$. d is at most the sum of the capacities on G, some constant.

Therefore the number of non-redundant constraints is $O(n^2)$.

Also, the maximum number of vertices's in the convex polygon defined by these $O(n^2)$ non-redundant constraints is also $O(n^2)$ ∎

References

[1] C. Berge. Two problems in graph theory. *Proc. Nat. Acad. Sci.* 43:842–844, 1957.

[2] T. C. Biedl, B. Brejová, and T. Vinař. Simplifying flow networks. *Proc. 25th Symp. Math. Found. Comput. Sci.*, 192–201, 2000. Lecture Notes Comput. Sci. 1893, Springer-Verlag.

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

[4] 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`⟩.

[5] G. Borradaile, C. Kenyon-Mathieu, and P. N. Klein. A polynomial-time approximation scheme for Steiner tree in planar graphs. *Proc. 18th Ann. ACM-SIAM Symp. Discrete Algorithms*, 1285–1294, 2007.

[6] S. Boyd. Convex optimization ii, 2007. ⟨`http://see.stanford.edu/see/materials/lsocoee364b/handouts.aspx`⟩.

[7] S. Cabello and E. W. Chambers. Multiple source shortest paths in a genus g graph. *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 89–97, 2007. Society for Industrial and Applied Mathematics.

[8] E. W. Chambers, J. Erickson, and A. Nayyeri. Homology flows, cohomology cuts. *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, 273–282, 2009. ACM.

[9] S. I. Daitch and D. A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. *STOC*, 451–460, 2008. ACM.

[10] G. Dantzig. *Maximization of a linear function of variables subject to linear inequalities: Activity analysis of production and allocation(Cowles Commission).* Rand Corporation, 1951.

[11] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.* 11:1277–1280, 1970.

[12] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM* 19(2):248–264. ACM, 1972.

[13] D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica* 27:275–291, 2000.

[14] J. Erickson. Maximum flows and parametric shortest paths in planar graphs. *To appear in SODA '10: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, 2010. ACM.

[15] L. Ford and D. Fulkerson. Maximal flow through a network. *Canadian J. Math* 8:399–404, 1956.

[16] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[17] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.* 16(6):1004–1022. Society for Industrial and Applied Mathematics, 1987.

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

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

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

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

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

[23] 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 [39].

[24] R. Hassin. Maximum flow in (s, t) planar networks. *Inf. Process. Lett.* 13(3):107, 1981.

[25] 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.

[26] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2001. ⟨`http://www.math.cornell.edu/~hatcher/`⟩.

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

[28] 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.

[29] A. Itai and Y. Shiloach. Maximum flow in planar networks. *SIAM J. Comput.* 8(2), 1979.

[30] 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.

[31] 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.

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

[33] 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. *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, 236–245, 2009. Society for Industrial and Applied Mathematics.

[34] Y. Kobayashi and K.-i. Kawarabayashi. Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, 1146–1155, 2009. Society for Industrial and Applied Mathematics.

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

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

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

[38] 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.

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

[40] 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 [3, 22].

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

[42] A. N. Tolstoi. Methods of finding the minimal total kilometrage in cargo-transportation planning in space. *Transportation Planning, Volume I, TransPress of the National Commissariat of Transportation*, 1930.

[43] S. M. Venkatesan. *Algorithms for network flows.* Ph.D. thesis, The Pennsylvania State University, 1983. Cited in [30].

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