Maximum Flows and Parametric Shortest Paths in Planar Graphs*

Jeff Erickson

Department of Computer Science
University of Illinois at Urbana-Champaign
jeffe@cs.uiuc.edu

Abstract

We observe that the classical maximum flow problem in any directed planar graph G can be reformulated as a parametric shortest path problem in the oriented dual graph G^*. This reformulation immediately suggests an algorithm to compute maximum flows, which runs in $O(n \log n)$ time. As we continuously increase the parameter, each change in the shortest path tree can be effected in $O(\log n)$ time using standard dynamic tree data structures, and the special structure of the parametrization implies that each directed edge enters the evolving shortest path tree at most once. The resulting maximum-flow algorithm is identical to the recent algorithm of Borradaile and Klein [*J. ACM* 2009], but our new formulation allows a simpler presentation and analysis. On the other hand, we demonstrate that for a similarly structured parametric shortest path problem on the torus, the shortest path tree can change $\Omega(n^2)$ times in the worst case, suggesting that a different method may be required to efficiently compute maximum flows in higher-genus graphs.

1 Introduction

The maximum flow and minimum cut problems have been staples of algorithms research for more than half a century. From the problem's inception in classified studies of the Soviet rail network [24], particular attention has been paid to computing flows and cuts in planar graphs, both because they arise naturally in many application settings, and because planar graphs often admit simpler and faster algorithms than general graphs. The seminal paper of Ford and Fulkerson [19], which introduced the maxflow-mincut theorem and the augmenting-path technique, also contained a polynomial-time augmenting-path algorithm for planar networks where the source and target are incident to a common face.

Maximum flows and minimum cuts are intimately related to shortest paths. It has been known since the early 1980s that computing a flow with a particular *value* in a planar graph G is equivalent to computing a single-source shortest-path tree in a carefully weighted dual graph G^* [25, 28, 30, 37, 45]. The fastest algorithms for computing minimum cuts in planar graphs [39, 20, 29, 27] and maximum flows in *undirected* planar graphs [26] are all directly formulated in terms of dual shortest paths; all these algorithms run in $O(n \log n)$ time. We describe these results in slightly more detail in Section 1.2.

Recently, Borradaile and Klein [5, 6] described an algorithm to compute maximum flows in arbitrary *directed* planar graphs in $O(n \log n)$ time, generalizing an earlier result of Weihe [47] for planar graphs satisfying a certain connectivity condition. Unlike the algorithms already mentioned, these two algorithms are *not* formulated in terms of shortest paths.

In this paper, we offer an alternative formulation of Borradaile and Klein's algorithm in terms of *parametric* shortest paths. In a parametric shortest path problem, the lengths of the edges are (in our case) linear functions of a real parameter λ; a typical goal is to compute shortest path trees for *all* values of λ for which the graph has no negative cycles [31, 49]. Parametric shortest paths have been previously used to find *minimum-cost* flows [21], but to our knowledge, not for the standard maximum-flow problem.

We set up the parametric shortest path problem in the dual graph G^*, so that for any value of λ, the current shortest path distances define a flow with value λ in the original network. In Section 2, we describe our parametric shortest-path in detail and show that it runs in $O(n \log n)$ time. As λ increases, each change in the shortest path tree can be effected in $O(\log n)$ time using a standard dynamic tree data structure [43], and the special structure of our parametrization implies that each directed edge enters the evolving shortest path tree at most once. The resulting algorithm is *identical* to the algorithm of Borradaile and Klein; however, in light of earlier results, our reformulation is arguably more natural, and our running time analysis is simpler.

We also consider a natural generalization of our parametric shortest-path formulation to higher genus graphs. Chambers *et al.* [10, 11] recently described the first algorithms to compute minimum cuts and

*Research reported in this paper was partially supported by NSF grant DMS-0528086. See http://www.cs.uiuc.edu/~jeffe/pubs/parshort.html for the most recent version of this paper.

maximum flows in graphs of fixed genus in near-linear time. In those papers, we conjectured that Borradaile and Klein's planar maximum-flow algorithm can be generalized to compute flows in any fixed-genus graph in $O(n \log n)$ time. As a first step in this direction, we consider a generalization of our planar parametric shortest path formulation to graphs on higher-genus surfaces. Unfortunately, our results are negative; in Section 3, we describe an infinite family of parametrized graphs on the torus, in which a shortest-path tree changes $\Omega(n^2)$ times.

1.1 Background

Planar graphs. Let $G = (V, E)$ be an directed plane graph; that is, a directed planar graph with a fixed planar embedding. Each edge $e \in E$ connects two vertices ***tail*(*e*)** and ***head*(*e*)**, and separates two faces ***left*(*e*)** and ***right*(*e*)**. We write $u \to v$ to denote the edge with tail u and head v, and $f \uparrow g$ to denote the edge with left shore f and right shore g. The ***reversal*** of edge e, denoted ***rev*(*e*)**, is defined by swapping its endpoints: $rev(u \to v) := v \to u$. Without loss of generality, we assume that the reversal of every edge in G is another edge in G. We also assume that the planar embedding maps every edge e and its reversal $rev(e)$ to the same curve in the plane (but parametrized in opposite directions). For any subgraph H of G, let ***rev*(*H*)** denote the subgraph obtained by reversing every edge of H.

Duality. The ***dual*** of a plane graph G is another plane graph G^* whose vertices correspond to faces of G and vice versa. Two vertices in G^* are joined by an edge if and only if the corresponding faces of G are separated by an edge of G. Thus, every edge e in G has a corresponding dual edge e^* in G^*. For any face f of G, let f^* denote the corresponding vertex of G^*; for any vertex v of G, let v^* denote the corresponding face of G^*. Each dual edge e^* is embedded so that it crosses the corresponding primal edge e (and its reversal) exactly once and intersects no other primal edge. Dual edges are oriented 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 plane to distinguish left from right. See Figure 1. For any subgraph H of G, let H^* denote the corresponding subgraph of G^*.

Flows and cuts. Let s and t be fixed vertices in G. An ***(s, t)*-cut** (or simply *cut* if there is no confusion) is a subset C of edges of G such that every path from s to t contains at least one edge in C.

An ***(s, t)*-flow** (or simply *flow* if there is no confusion) is an function $\phi : E \to \mathbb{R}$ that satisfies two conditions. First, ϕ must be *antisymmetric*; that is, $\phi(e) = -\phi(rev(e))$ for every edge e. Second, ϕ must satisfy the following *conservation constraint*: $\sum_w \phi(v \to w) = 0$ for every vertex v except s and t. The *value* of the flow is $\sum_w \phi(s \to w)$. (To simplify notation, we *define* $\phi(v \to w)$ to be 0 when $v \to w$ is not an edge in G.)

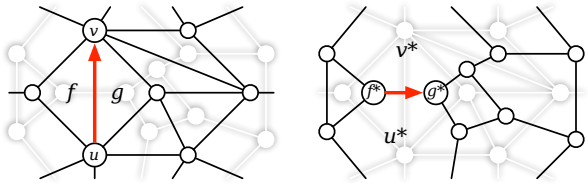

Figure 1. Graph duality. One edge and its dual are emphasized.

Now fix a function $c : E \to \mathbb{R}$ that assigns a non-negative real *capacity* to every edge in G. An flow ϕ is *feasible* if $\phi(e) \le c(e)$ for every edge e. A flow ϕ *saturates* an edge e if $\phi(e) = c(e)$. The capacity of a cut C is the sum of its edge capacities: $c(C) := \sum_{e \in C} c(E)$. The maxflow-mincut theorem [13, 15, 19] states that the maximum feasible (s, t)-flow value is equal to the minimum (s, t)-cut capacity. In particular, if a flow ϕ saturates every edge in a cut C, then ϕ must be a maximum flow, and C must be a minimum cut.

1.2 Prior Results

Maximum Flows in Planar Graphs. Borradaile and Klein [5, 6], Khuller and Naor [32], and Weihe [47] describe the history of planar flow algorithms in detail. Here we describe only a few key developments, emphasizing the deep connections between flows, cuts, and dual shortest paths.

More than 25 years ago, Venkatesan [45] observed that for any planar graph G, a feasible (s, t)-flow with a given value λ 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. We describe this reduction in detail in Section 2; similar approaches were also proposed by Itai and Shiloach [28], Hassin [25], Johnson and Venkatesan [30], Hassin and Johnson [26], Khuller *et al.* [33], and Miller and Naor [37]. Venkatesan applied the planar shortest-path algorithm of Lipton, Rose, and Tarjan [36] to compute a feasible flow with a given value in $O(n^{3/2})$ time. This running time can be improved by more recent planar shortest-path algorithms [27, 17]; the fastest algorithm to date, due to Klein, Mozes, and Weimann [35], runs in $O(n \log^2 n)$ time.

Itai and Shiloach [28] observed that the minimum cut in a planar graph G is dual to the minimum-cost cycle that separates faces s^* and t^* in the dual graph G^*.

Reif [39] described a divide-and-conquer algorithm to find this cycle in $O(n \log^2 n)$ time when the graph is undirected; Reif's algorithm was later generalized to directed graphs by Janiga and Koubek [29]. Combining these minimum-cut algorithms with the shortest-path algorithm of Klein *et al.* [35] yields an algorithm to compute maximum flows in directed planar graphs in $O(n \log^2 n)$ time. Frederickson [20] improved the running time of Reif's algorithm to $O(n \log n)$; the same improvement can also be obtained using more recent planar shortest path algorithms [27, 34, 7, 44]. The running time of Janiga and Koubek's algorithm can be improved to $O(n \log n)$ using the linear-time planar shortest-path algorithm of Henzinger *et al.* [27].

For undirected planar graphs, Hassin and Johnson [26] extended Reif's minimum-cut algorithm to compute a maximum flow in only $O(n \log n)$ additional time; together with Frederickson's improvement of Reif's algorithm [20], this implies an undirected planar maxflow algorithm that runs in $O(n \log n)$ time. Hassin and Johnson's algorithm also computes a single-source shortest path tree in a dual graph with positive and negative weights; however, the additional structure computed by Reif's algorithm allows these shortest paths to be computed using a modification of Dijkstra's algorithm [14], as through all the edge weights were positive.

Weihe [47, 46] and Borradaile and Klein [5, 6] described algorithms to compute maximum flows in directed planar graphs in $O(n \log n)$ time. Weihe's algorithm requires the input graph to satisfy a certain connectivity condition.[1] Borradaile and Klein's algorithm works for any directed planar graph. Both algorithms compute a single-source shortest path tree in the dual of the input graph in a preprocessing phase, in order to remove all clockwise cycles from the graph [33]; otherwise, they do not (explicitly) use dual shortest paths. Instead, both algorithms are instances of Ford and Fulkerson's classical augmenting path technique [19]. We describe Borradaile and Klein's algorithm in more detail later in the paper.

Maximum Flows in Higher-Genus Graphs. Chambers *et al.* [10] recently described the first algorithms to compute maximum flows in graphs of fixed genus in near-linear time. Specifically, given a graph with integer capacities embedded on a surface of genus g, they describe an algorithm to compute a maximum flow in $O(g^7 n \log^2 n \log^2 C)$ time. They also describe a combinatorial maximum-flow algorithm that runs in $g^{O(g)} n^{3/2}$ arithmetic operations.[2] Both algorithms reduce the maximum-flow problem to a $(2g+1)$-dimensional linear programming problem, which is then solved implicitly using a shortest-path algorithm as an oracle. These algorithms are considerably more complex than Borradaile and Klein's algorithm for planar graphs [5, 6]. Our results in Section 3 reflect an attempt to generalize Borradaile and Klein's simpler approach to the higher genus setting.

Parametric Shortest Paths. Karp and Orlin [31] formulated the parametric shortest path problem as follows. The input is a directed graph $G = (V, E)$, a subset of edges $E' \subseteq E$, and a cost function $c : E \to \mathbb{R}$. We add a second real parameter λ to the cost function by defining $c(\lambda, e) = c(e) - \lambda$ for each edge $e \in E'$, and $c(\lambda, e) = c(e)$ otherwise. The goal of the problem is to compute the largest value of λ such that the resulting weighted directed graph contains no negative cycles. Karp and Orlin describe an algorithm to solve this problem in $O(nm \log n)$ time; the running time was improved to $O(nm + n^2 \log n)$ by Young, Tarjan, and Orlin [49].

Both of these algorithms maintain a single-source shortest path tree rooted at some node s, with respect to the cost function $c(\lambda, \cdot)$, while increasing λ continuously from $-\infty$. At certain critical values of λ, the shortest-path tree changes by a single edge. For any vertex v, let $dist(\lambda, v)$ denote the shortest-path distance from s to v for a particular value of λ; these distances satisfy the inequality $dist(\lambda, v) \le dist(\lambda, u) + c(\lambda, u \to v)$ for every edge $u \to v$, with equality if $u \to v$ is in the shortest path tree. A critical value of λ occurs when $dist(\lambda, v) = dist(\lambda, u) + c(\lambda, u \to v)$ for some edge $u \to v$ that is *not* in the shortest path tree. At that critical value, $u \to v$ enters the shortest path tree, replacing some edge $u' \to v$; we call this change a ***pivot***. The algorithm ends when a pivot introduces a cycle (which must have cost 0) into the shortest-path tree.

Each pivot increases the number of edges in E' on the shortest path from s to some other vertex v. It follows immediately that the shortest path to any vertex changes at most n times, which implies that the shortest-path tree undergoes at most $O(n^2)$ pivots.

The parametric shortest path problem can be generalized by allowing the costs $c(\lambda, e)$ to be arbitrary linear functions of the parameter λ. The algorithms of Karp and Orlin [31] and Young *et al.* [49] can be adapted to this setting with only trivial modifications. If the cost functions have only a constant number of different λ-coefficients, the number of pivots is still at most $O(n^2)$,

[1] Specifically, every edge $u \to v$ must lie on a simple directed path from s to v and on a simple directed path from u to t. Any edge that does not satisfy this condition can be safely removed, because there is a maximum flow that does not use them; unfortunately, the fastest algorithm to find all such edges requires $O(n^2)$ time [4].

[2] The conference version of their paper [10] incorrectly claimed a running time of $g^{O(g)} n \operatorname{polylog} n$; see the full version for details.

and the running times of the algorithms changes by only a constant factor. However, for the most general case where every edge weight has a different λ-coefficient, Carstensen [9] proved that the worst-case number of pivots is $n^{\Omega(\log n)}$; see also Mulmuley and Shah [38]. Carstensen attributes a matching $n^{O(\log n)}$ upper bound to Gusfield [23].

2 Maximum Flows in Planar Graphs

Throughout this section, let $G = (V, E)$ be a directed plane graph, let $c : E \to \mathbb{R}$ be a nonnegative capacity function, let s and t be vertices of G. Our goal is to compute a maximum (s, t)-flow in G. We assume without loss of generality that the reversal of any directed edge in G is also an edge in G; otherwise, we can add the missing reversed edges with capacity 0. This assumption implies that both G and its dual G^* are strongly connected. To simplify our presentation, we assume that the capacity function is *generic*; we will point out the consequences of this assumption as they arise.[3]

Finally, in the interest of readability, we will always use the letters s, t, u, v, w to denote vertices of the primal graph G, and the letters o, p, q, r to denote vertices of the dual graph G^*. Thus, o^* is a face of G, and s^* is a face of G^*.

2.1 Venkatesan's Reduction

We now describe Venkatesan's algorithm [45] to compute a feasible (s, t)-flow with fixed value λ, or correctly report that no such flow exists, by reduction to a single-source shortest path problem in an appropriately weighted dual graph G^*. Our presentation differs significantly from Venkatesan's, so that we can introduce some useful notation.

Fix an arbitrary directed path P from s to t, and let $\pi : E \to \mathbb{R}$ denote the unit flow through P:

$$\pi(e) := \begin{cases} 1 & \text{if } e \in P, \\ -1 & \text{if } \mathit{rev}(e) \in P, \\ 0 & \text{otherwise} \end{cases}$$

For any subset $E' \subseteq E$, let $\pi(E') = \sum_{e \in E'} \pi(e)$. A subgraph C of G is called a ***cocycle*** if the corresponding dual subgraph C^* is a simple directed cycle in G^*. For any cycle C^* in G^*, we call $\pi(C)$ the ***crossing number*** of C^*; this is the number of times P crosses the cycle C^* from left to right, minus the number of times P crosses C^* from right to left. Whitney [48] observed that any cocycle in a planar graph is also a cut; the next lemma refines this observation.

Lemma 2.1. *$\pi(C) \in \{-1, 0, 1\}$ for any cocycle C. Moreover, $\pi(C) = 1$ if and only if C is an (s, t)-cut.*

Proof: Every edge in the cocycle C crosses its dual cycle C^* from left to right; every edge in $\mathit{rev}(C)$ crosses C^* from right to left. The Jordan Curve Theorem implies that C^* partitions the plane into exactly two connected regions; call these $\mathit{left}(C^*)$ and $\mathit{right}(C^*)$. There are only three cases to consider.

Suppose s and t lie on the same side of C^*. Then P must cross C^* the same number of times in each direction. Equivalently, P must contain the same number of edges in C and $\mathit{rev}(C)$. Thus, $\pi(C) = 0$.

Suppose $s \in \mathit{left}(C^*)$ and $t \in \mathit{right}(C^*)$. Then P must cross C from left to right once more than it crosses C from right to left. Equivalently, P must contain one more edge in C than in $\mathit{rev}(C)$, so $\pi(C) = 1$. Moreover, *any* path from s to t must contain at least once edge in C; in other words, C is an (s, t)-cut. See Figure 2.

A symmetric argument implies that if $s \in \mathit{right}(C^*)$ and $t \in \mathit{left}(C^*)$, then $\pi(C) = -1$.

Finally, if C is an (s, t)-cut, then every path from path from s to t must contain at least once edge in C, and therefore crosses C^* from right to left. It follows that $\pi(C) = 1$. □

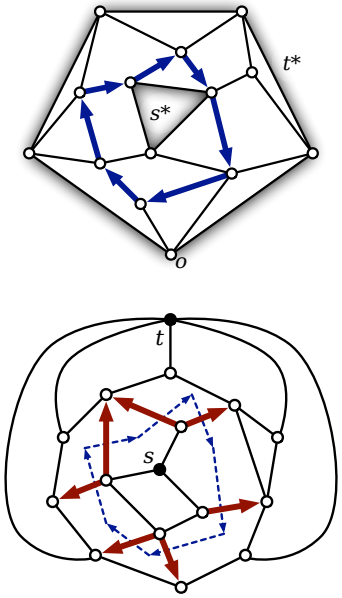

Figure 2. A directed cycle in G^* and the corresponding directed cocycle in G, which is also an (s, t)-cut.

Now consider the flow $\lambda \cdot \pi$, which assigns value λ to every directed edge in path P, value $-\lambda$ to every

[3] Our genericity assumption can be enforced by standard perturbation techniques, but in fact, ties that arise during our algorithm can be broken arbitrarily. Details will appear in the full version of the paper.

edge in $rev(P)$, and value 0 to every other edge. Let $\boldsymbol{G_\lambda} := G_{\lambda\cdot\pi}$ denote the residual network of this flow; this is just the graph G with the residual capacity function $c(\lambda, e) := c(e) - \lambda \cdot \pi(e)$. The flow $\lambda \cdot \pi$ is feasible if and only if $c(\lambda, e) \geq 0$ for every edge e of G. Finally, let $\boldsymbol{G^*_\lambda}$ denote the *dual residual network*, which is just the directed dual graph G^* where every dual edge e^* has a ***cost*** $c(\lambda, e^*)$ equal to the residual capacity of the corresponding primal edge: $c(\lambda, e^*) = c(\lambda, e)$.

Lemma 2.2. *There is a feasible (s,t)-flow in G with value λ if and only if the dual residual network G^*_λ does not contain a negative cycle.*

Proof: Because G^*_λ is strongly connected, there are only two cases to consider.

First, suppose G^*_λ contains a negative cycle C^*. We can decompose the cost of this cycle as follows:

$$c(\lambda, C^*) = \sum_{e \in C} c(\lambda, e) = \sum_{e \in C} c(e) - \lambda \cdot \pi(C) < 0.$$

Because $c(e) \geq 0$ for every edge e, we must have $\pi(C) > 0$. Thus, Lemma 2.1 implies that the cocycle C is an (s,t)-cut whose capacity is less than λ. We conclude that there is no feasible (s,t)-flow in G with value λ.

On the other hand, suppose shortest paths in G^*_λ are well-defined. Fix an arbitrary dual vertex o (called the *origin*) in G^*_λ. For any dual vertex p, let $\boldsymbol{dist(\lambda, p)}$ denote the shortest path distance in G^*_λ from o to p. Now consider the function

$$\phi(\lambda, e) := dist(\lambda, head(e^*)) - dist(\lambda, tail(e^*)) + \lambda \cdot \pi(e).$$

For every vertex v, we have $\sum_w \phi(\lambda, v{\to}w) = \sum_w \lambda \cdot \pi(v{\to}w)$; the duals of the edges leaving v define a directed cycle in G^*, so all the $dist(\lambda, \cdot)$ terms in the sum cancel out. It follows that $\phi(\lambda, \cdot)$ is a valid (s,t)-flow with value λ. Now define the *slack* of each dual edge e^* as follows:

$$\boldsymbol{slack(\lambda, e^*)} := dist(\lambda, tail(e^*)) + c(\lambda, e) - dist(\lambda, head(e^*)).$$

We easily observe that $slack(\lambda, e^*) = c(e) - \phi(\lambda, e)$. Ford's classical formulation of shortest paths [18] implies that every dual edge has non-negative slack. We conclude that the flow $\phi(\lambda, \cdot)$ is feasible. □

The proof of Lemma 2.2 immediately implies an $O(n)$-time algorithm to compute the flow values $\phi(\lambda, \cdot)$ from the dual shortest path distances $dist(\lambda, \cdot)$. We emphasize here that the final flow $\phi(\lambda, \cdot)$ depends on the value of λ and the choice of the origin vertex o, but does *not* depend on the path P. In fact, we can replace π with *any* flow of value 1 without changing the output flow.

2.2 Parametric Shortest Paths

We now extend the previous reduction into an algorithm to compute the maximum flow, by treating the value λ as a continuously varying parameter. Our parametric shortest path problem has almost the same form as the problem studied by Karp and Orlin [31] and Young, Tarjan, and Orlin [49]. The only differences are that the parametric cost function $c(\lambda, \cdot)$ has three different λ-coefficients (-1, 0, and 1), and that both the graph G^* and the set P^* of parametrized edges have a special structure.

Let $\lambda_{\max}$ denote the largest value of λ for which shortest paths in G^*_λ are well-defined; Lemma 2.2 implies that $\lambda_{\max}$ is also the value of the maximum flow. For any particular value of λ, let $\boldsymbol{T_\lambda}$ denote the single-source shortest path tree in G^*_λ rooted at o. Because our input capacities are non-negative, shortest-paths are well-defined in G^*_0, and therefore T_0 is well-defined. *Our genericity assumption implies that T_λ is uniquely defined for all λ between 0 and $\lambda_{\max}$, except for a finite set of* ***critical values****.*

At a very high level, our algorithm can be described as follows:

```
PlanarMaxFlow(G, c, s, t):
  Compute T_0.
  Maintain T_λ as λ increases continuously from 0 to λ_max.
  Compute φ(λ_max, ·) from T_{λ_max}.
```

The edges in T_λ are directed away from o. Thus, every dual vertex $p \neq o$ has exactly one incoming edge in T_λ, from its parent vertex, which we denote $\boldsymbol{pred(\lambda, p)}$. We call a dual edge e^* ***tense*** at λ if $slack(\lambda, e^*) = 0$; except at critical values of λ, a dual edge is tense at λ if and only if it lies in T_λ. At each critical value of λ, some non-tree dual edge $p{\to}q$ becomes tense and enters T_λ, replacing the previous edge $pred(\lambda, q){\to}q$ (unless $q = o$). We call this event a ***pivot***; see Figure 3. *Our genericity assumption implies that exactly one non-tree edge becomes tense at each critical value of λ.*

We can precompute $\lambda_{\max}$ in $O(n \log n)$ time by computing a minimum (s,t)-cut [29, 27]. However, no such precomputation is necessary; we can detect $\lambda_{\max}$ when it occurs. For any dual vertex p, let $\boldsymbol{path(\lambda, p)}$ denote the unique path from o to p in T_λ. For any dual edge $p{\to}q$, let $\boldsymbol{cycle(\lambda, p{\to}q)}$ denote the cycle obtained by concatenating the shortest path $path(\lambda, p)$, the dual edge $p{\to}q$, and the reversed shortest path $rev(path(\lambda, q))$.

Lemma 2.3. *$\lambda_{\max}$ is the first critical value of λ whose pivot introduces a directed cycle into T_λ.*

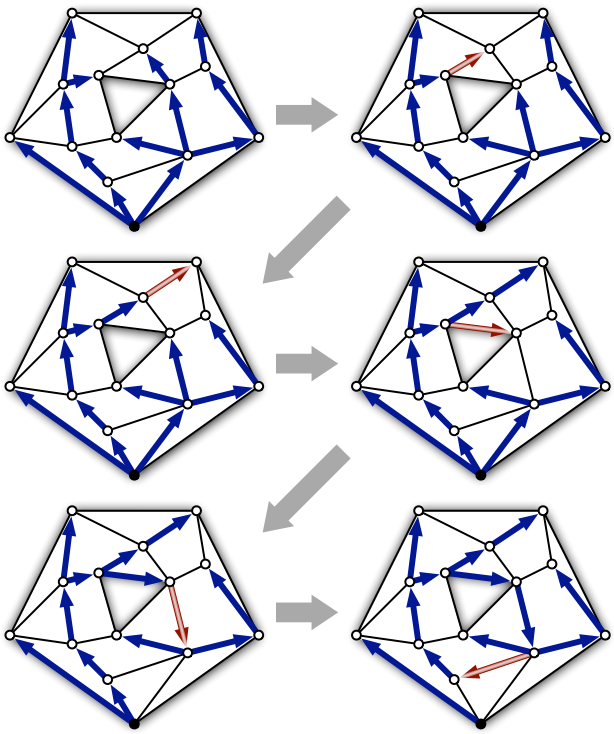

Figure 3. A possible sequence of shortest-path-tree pivots.

Proof: Let $\lambda^\circlearrowleft$ be the first critical value of λ whose pivot introduces a directed cycle into T_λ, and let C^* be that directed cycle. We easily observe that $C^* = \mathit{cycle}(\lambda^\circlearrowleft, p{\to}q)$, where $p{\to}q$ is the pivot edge introduced at $\lambda^\circlearrowleft$, so Lemma 2.1 implies that the cocycle C is an (s,t)-cut. Every dual edge in C^* is tense at $\lambda^\circlearrowleft$, which implies that $\phi(\lambda^\circlearrowleft, e) = c(e)$. It follows that the flow $\phi(\lambda^\circlearrowleft, \cdot)$ saturates the cut C and is therefore a maximum flow. (Moreover, C is a minimum cut.) We conclude that $\lambda^\circlearrowleft = \lambda_{\max}$. □

We can also characterize $\lambda_{\max}$ by considering a related structure in the primal graph G. Call a primal edge e ***loose*** at λ if neither its dual e^* nor its reversed dual $\mathit{rev}(e^*)$ is tense at λ, and let $\boldsymbol{L_\lambda}$ be the subgraph of all loose edges. Except at critical values of λ, the subgraph L_λ is a spanning tree of G; together, L_λ and T_λ define a *tree-cotree decomposition* of G [16, 42].

Lemma 2.4. *$\lambda_{\max}$ is the smallest critical value of λ whose pivot disconnects L_λ.*

A dual edge is ***active*** at λ if its slack at λ is decreasing. Because all slacks are always non-negative, only active edges can become tense. The primal spanning tree L_λ contains a unique directed path from s to t; call this loose path $\boldsymbol{LP_\lambda}$.

```
PlanarMaxFlow(G, c, s, t):
  Initialize the spanning tree L, predecessors, and slacks

  while s and t are in the same component of L
      LP ← the path in L from s to t
      p→q ← the edge in LP* with minimum slack
      Δ ← slack(p→q)
      for every edge e in LP
          slack(e*) ← slack(e*) − Δ
          slack(rev(e*)) ← slack(rev(e*)) + Δ
      delete (p→q)* from L
      if q ≠ o ⟨⟨that is, if pred(q) ≠ ∅⟩⟩
          insert (pred(q)→q)* into L
      pred(q) ← p

  for each edge e
      φ(e) ← c(e) − slack(e*)
  return φ
```

Figure 4. Our planar maximum flow algorithm.

Lemma 2.5. *A dual edge e^* is active at λ if and only if e is an edge of LP_λ.*

Proof: We can decompose the slack function for any edge into two components, one constant and one varying with λ, by defining

$$\mathit{slack}(\lambda, e^*) := \mathit{slack}_0(\lambda, e^*) - \lambda \cdot \mathit{slack}'(\lambda, e^*).$$

Thus, an edge e^* is active at λ if and only if $\mathit{slack}'(\lambda, e^*) > 0$. Straightforward definition-chasing implies that $\mathit{slack}'(\lambda, e^*)$ is the crossing number of $\mathit{cycle}(\lambda, p{\to}q)$. Thus, by Lemma 2.1, e^* is active at λ if and only if the cocycle $C(e) = \mathit{cycle}(\lambda, e^*)^*$ is an (s,t)-cut. If $C(e)$ is an (s,t)-cut, then LP_λ must contain at least one edge in $C(e)$, but the only loose edge in C is e. On the other hand, removing any edge e from L disconnects L, so $C(e)$ must be an (s,t)-cut. □

2.3 Implementation Details

Our algorithm is described in detail in Figure 4. Our algorithm explicitly maintains three structures: The spanning tree L_λ of loose edges, the predecessor $\mathit{pred}(\lambda, q)$ of every dual vertex q, and the slack value $\mathit{slack}(\lambda, e^*)$ of every dual edge e^*. We maintain the spanning tree L_λ and the dual slacks in a self-adjusting top tree [43] or some equivalent dynamic tree structure [1, 3, 41]. This data structure stores an n-vertex forest with weighted edges supports the following operations in $O(\log n)$ amortized time: determine whether two nodes are in the same component, *expose* a path between two specified nodes, find the edge on the exposed path with minimum value, add some amount to all values on the exposed path, remove an edge, and insert an edge. We maintain the predecessors as simple pointers. Surprisingly, we do

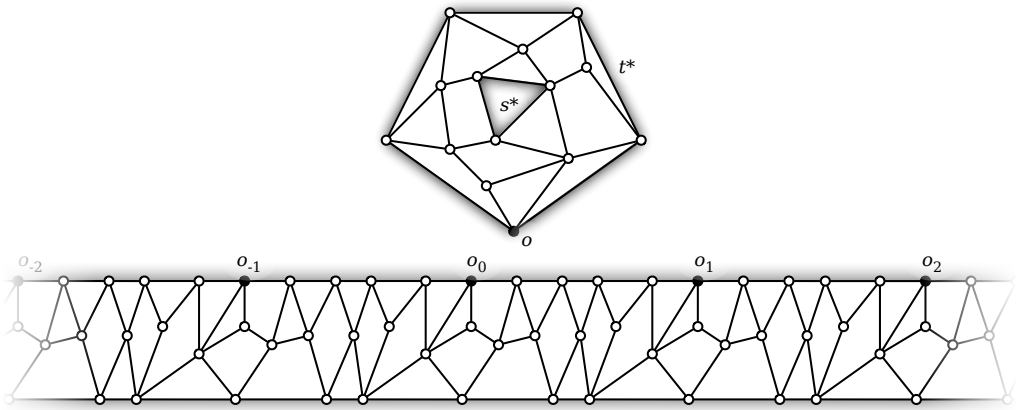

Figure 5. The dual graph G^* and its universal cover $\overline{G}^*$.

not need to choose a flow path P or explicitly maintain the parameter λ!

We can compute the initial predecessor pointers and slacks in $O(n \log n)$ time using Dijkstra's shortest-path algorithm [14]. (This time can be reduced to $O(n)$ if we use the shortest-path algorithm of Henzinger *et al.* [27], but this will be dominated by other parts of the algorithm.) The self-adjusting top tree for L can also be initialized in $O(n \log n)$ time, by inserting the edges one at a time into an initially empty forest. Each iteration of the main loop requires $O(\log n)$ amortized time. Thus, the running time of our algorithm is $O((n+N)\log n)$, where N is the number of pivots.

2.4 Number of Pivots

To complete the analysis of our algorithm, we only need to prove and upper bound on the worst-case number of pivots. Our analysis requires one additional assumption, also made by Borradaile and Klein [5, 6]: ***The target vertex t is incident to the outer face o^*.*** With this assumption in place, we show that the shortest path tree undergoes at most $O(n)$ pivots in the worst case. In fact, we will prove the following stronger statement: Each dual edge pivots into the shortest path tree at most once.[4]

Recall that P is an arbitrary directed path from s to t in G, and π is the corresponding unit flow. Let Π be an arbitrary path from o to another vertex p in the dual graph G^*. We call the integer $\pi(\Pi) = \sum_{e^* \in \Pi} \pi(e)$ the ***crossing number*** of Π.

Lemma 2.6. *Each time $path(\lambda, p)$ changes, its crossing number increases by 1.*

[4]Borradaile (personal communication) has proved that the number of pivots is $O(n)$ even if neither s nor t is incident to o^*. Specifically, each dual edge pivots into T_λ at most three times.

For any dual vertex p and any integer i, let $\boldsymbol{path_i(p)}$ denote the shortest path from o to p in $G^* = G_0^*$ with crossing number i. Because all paths with the same crossing number decrease in length at the same rate, every shortest path $path(\lambda, p)$ is equal to $path_i(p)$ for some integer i.

Consider the annulus obtained by deleting faces s^* and t^* from the plane; the universal cover of this annulus is an infinite planar strip. We define an infinite planar graph $\overline{G}^* = (\overline{V}^*, \overline{E}^*)$ by lifting the dual graph G^* to this universal cover. The vertices and edges of $\overline{G}^*$ are formally defined as follows:

$$\overline{V}^* := \left\{ p_i \;\middle|\; p \in V^* \text{ and } i \in \mathbb{Z} \right\}$$
$$\overline{E}^* := \left\{ p_i \to q_{i+\pi(p\to q)} \;\middle|\; p \to q \in E^* \right\}$$

Informally, $\overline{G}$ can be constructed as follows. Create a doubly-infinite sequence $\ldots, G_{-1}^*, G_0^*, G_1^*, G_2^*, \ldots$ of copies of G^*. For any dual vertex p and any integer i, let p_i denote the corresponding vertex of G_i^*; we call this vertex a ***lift*** of p. Then for every integer i and every dual edge $p \to q$ in P^*, replace the edge $p_i \to q_i$ with $p_i \to q_{i+1}$, and symmetrically replace $q_i \to p_i$ with $q_i \to p_{i-1}$. See Figure 5. Edge costs in $\overline{G}^*$ are inherited from edge costs in G^* in the obvious way: $c(p_i \to q_j) = c(p \to q)$.

There is a natural ***projection*** map $\varpi : \overline{G}^* \to G^*$ that simply drops subscripts: $\varpi(p_i) = p$ and $\varpi(p_i \to q_j) = p \to q$. The preimage $\varpi^{-1}(\Pi)$ of any path Π in G^* is a doubly-infinite set of paths in $\overline{G}^*$, which we call the ***lifts*** of Π. Specifically, if Π starts at p and ends at q, then for any integer i, there is a lift of Π that starts at p_i and ends at $q_{i+\pi(\Pi)}$. Faces s^* and t^* lift to two unbounded faces $\overline{s}^*$ and $\overline{t}^*$; every other face of G^* lifts to an infinite sequence of faces in $\mathcal{G}^*$.

Let $\boldsymbol{\overline{path}(p_i, q_j)}$ denote the shortest path in $\overline{G}^*$ from p_i to q_j. For any vertex p and any integer j,

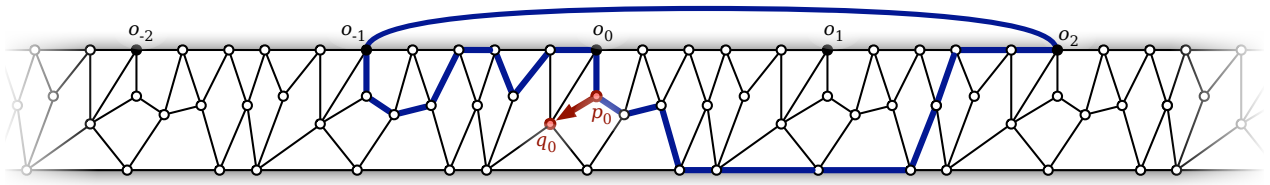

Figure 6. Proof of Lemma 2.7: Indices −2, 0, and 1 are interesting; $i = -2$, $j = 1$, and q_0 lies outside Γ.

$path_j(p)$ is the projection of $\overline{path}(o_0, p_j)$ and therefore also the projection of $\overline{path}(o_i, p_{i+j})$ for all i. It follows that every shortest path $path(\lambda, p)$ is the projection of a shortest path in $\overline{G}^*$.

Lemma 2.7. *Each dual edge $p \to q$ pivots into the shortest path tree T_λ at most once.*

Proof: Fix a dual edge $p \to q$. To simplify the exposition, assume that $\pi(p \to q) = 0$; similar arguments apply when $\pi(p \to q) = 1$ or $\pi(p \to q) = -1$.

Call an integer i *interesting* if $p \to q$ is the last edge of $path_i(q)$, or equivalently, if $p_0 \to q_0$ is the last edge of $\overline{path}(o_{-i}, q_0)$. To prove the lemma, it suffices to show that there is at most one interesting integer i whose successor $i+1$ is boring.

Suppose to the contrary that there are two integers $i < j$ such that i and j are interesting but $i+1$ and $j+1$ are boring. Let Γ be the cycle formed by concatenating the shortest path $\overline{path}(o_{-i}, p_0)$, the reversed shortest path $rev(\overline{path}(o_{-j}, p_0)$, and a path through $\overline{t}^*$ from o_{-j} to o_{-i}. The vertex q_0 does not lie on Γ.

The vertex $o_{-(j+1)}$ lies outside Γ, and the shortest path $\overline{path}(o_{-(j+1)}, q_0)$ does not contain the edge $p_0 \to q_0$. So q_0 must lie outside Γ; see Figure 6. On the other hand, vertex $o_{-(i+1)}$ lies inside Γ, and the shortest path $\overline{path}(o_{-(i+1)}, q_0)$ does not contain the edge $p_0 \to q_0$. So q_0 must lie inside Γ. We have a contradiction. □

Theorem 2.8. *Maximum flows in directed planar graphs can be computed in $O(n \log n)$ time.*

In the appendix, we briefly argue our algorithm is actually identical to Borradaile and Klein's algorithm [5, 6]. However, our proof that each edge in G is saturated at most once is considerably simpler than theirs, in part because they prove a stronger statement called the 'Unusability Theorem': If an edge e is augmented, and later its reversal $rev(e)$ is augmented, then e cannot be augmented again.

3 Parametric Shortest Paths on Surfaces

3.1 Motivation

Chambers *et al.* [10] recently described efficient algorithms to compute maximum flows in graphs of higher genus. Their algorithms recast the maximum problem as a *multi-parametric* shortest path problem with $2g + 1$ parameters in the dual graph G^*. Here, the goal is to find parameter values whose sum is maximized, such that the resulting edge weights do not induce a negative cycle in G^*. Chambers *et al.* solve the resulting $(2g+1)$-dimensional linear optimization problem using either the ellipsoid method [22] or multidimensional parametric search [2, 12], using a single-source shortest path algorithm [35] as a membership oracle. These algorithms are considerably more complex than the algorithm described in this paper, or the equivalent algorithm of Borradaile and Klein. It is natural to ask whether a the parametric approach in this paper can be generalized to obtain simpler and/or more efficient algorithms for the higher-genus setting.

As a first step toward understanding the structure of this multi-parametric shortest-path problem, we first consider the natural generalization of our planar *single*-parameter shortest path problem to higher-genus graphs. Let G be a graph embedded on the torus (the oriented surface of genus 1), let s and t be vertices of G, let P be a directed path from s to t, and let π be the unit flow through P. The edge costs in the dual residual graph G^*_λ are defined exactly as in the planar case $c(\lambda, e^*) = c(e) - \lambda\pi(e)$. The problem is to compute $\lambda_{\max}$, the largest value of λ such that G^*_λ has no negative cycles.

The algorithm in Figure 4 can be generalized to solve this problem. As in the planar case, let T_λ denote the shortest-path tree in G^*_λ rooted at some vertex o, and let L_λ denote the complementary subgraph of G. Euler's formula implies that L_λ is no longer a spanning tree, but rather a spanning tree plus two extra edges [16]. Cabello and Chambers [7] describe a generalization of self-adjusting top trees that can maintain this subgraph as we perform pivots in T_λ, still in $O(\log n)$ time per pivot. Thus, the running time of the generalized algorithm is still $O((n+N)\log n)$, where N is the number of pivots. We omit further details.

3.2 A Bad Example

Unfortunately, Lemma 2.7 is no longer true in this setting. In this section, we describe an infinite family

of parametrized graphs on the torus of the form just described, such that T_λ undergoes $\Omega(n^2)$ pivots, matching the trivial $O(n^2)$ upper bound of Karp and Orlin [31]. We describe the sequence of dual graphs G^*, one for each value of n. Figure 7 shows our abstract graph and its embedding on the torus for the case $n = 5$.

Fix a positive integer n. We start by defining an *undirected* graph G^* with vertices $o, o^+, o^-, p_0, p_1, \ldots, p_{2n}$, $q_1, \ldots, q_n, r_0, r_1, \ldots, r_{2n}$. We assign the following costs to the edges of G^*, for all i:

$$\begin{aligned}
c(o\,o^+) &= c(o\,o^-) = c(o^+o^-) = \infty \\
c(o\,p_0) &= c(o\,r_0) = 0 \\
c(p_{i-1}\,p_i) &= c(r_{i-1}\,r_i) = n \\
c(p_{2n}\,q_i) &= 1 \\
c(r_{2n}\,q_i) &= 2 + \frac{i-1}{n-1} \\
c(p_{2i-2}\,p_{2i}) &= \begin{cases} 6n-4i+1 & \text{if } i \text{ is even,} \\ 6n-4i+4 & \text{if } i \text{ is odd.} \end{cases} \\
c(r_{2i-2}\,r_{2i}) &= \begin{cases} 6n-4i+4 & \text{if } i \text{ is even,} \\ 6n-4i+1 & \text{if } i \text{ is odd.} \end{cases}
\end{aligned}$$

The costs $c(r_{2n}\,q_i)$ are non-integral only to ensure that at most one pivot occurs at each critical value of λ. If this is not a concern, we can set $c(r_{2n}\,q_i) = 2$ for all i.

We embed G^* on the torus as follows. First we embed the cycles $op_0p_1\cdots p_{2n}q_1r_{2n}r_{2n-1}\cdots r_0$ and oo^+o^- so they are non-contractible and in independent homotopy classes. Then we spiral the paths $p_0p_2p_4\cdots p_{2n}$ and $r_0r_2r_4\cdots r_{2n}$ around the torus in opposite directions. Finally, we embed all paths $p_{2n}q_ir_{2n}$ in parallel.

To define the parametrized graph G^*_λ, we break each undirected edge of G^* into two symmetric directed edges with the same initial weight. For each directed edge e, we set $c(\lambda, e) := c(e) - \lambda \cdot \pi(e)$, where $\pi(p_{2i-2}\to p_{2i}) = \pi(r_{2i-2}\to r_{2i}) = 1$ and $\pi(p_{2i}\to p_{2i-2}) = \pi(r_{2i}\to r_{2i-2}) = -1$ for all i, and $\pi(e) = 0$ for all other edges e. Let s^* be the face incident to both o and p_1, and let t^* be the unique face incident to both o and r_1. Let P^* be the set of dual edges e^* with $\pi(e^*) = 1$; we easily confirm that P^* is dual to a directed path from s to t in the primal graph G. See Figure 7.

Theorem 3.1. *For the parametrized graph G^*_λ described in the text, the shortest path tree rooted at o pivots $\Omega(n^2)$ times.*

Proof: We easily observe that $\lambda_{\max} = 4n+1$. Specifically, the final graph G^*_{4n+1} contains the zero-length cycle $p_{2n-2}\to p_{2n-1}\to p_{2n}\to p_{2n-2}$ if n is even, and the zero-length cycle $r_{2n-2}\to r_{2n-1}\to r_{2n}\to r_{2n-2}$ if n is odd.

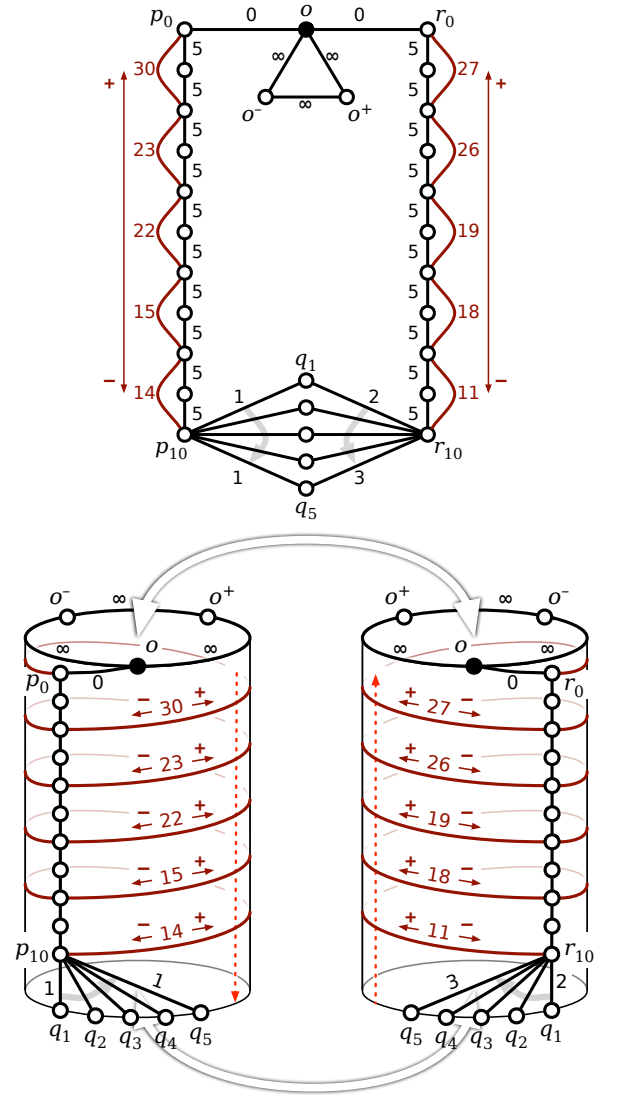

Figure 7. Top: The abstract graph G^* when $n = 5$. Bottom: An embedding of G^* on the torus. Corresponding boundary curves on the cylinders are identified. Arrows indicate the effect of parametrization.

Vertices p_{2n} and r_{2n} are the only possible predecessors for each vertex q_i. Tedious calculation implies that $pred(\lambda, q_i)$ alternates between these possibilities exactly n times. Specifically, $pred(8k, q_i) = p_{2n}$ for every integer $0 \le k \le n/2$, and $pred(8k+4, q_i) = r_{2n}$ for every integer $0 \le k \le (n-1)/2$. There are n vertices q_i, so the total number of pivots is at least n^2. (More careful analysis implies that the number of pivots is exactly $(n+1)^2$.) □

Acknowledgements. I would like to thank Amir Nayyeri, Aparna Sundar, Chandra Chekuri, Cora Borradaile, and Erin Chambers for helpful discussions.

References

[1] U. A. Acar, G. E. Blelloch, R. Harper, J. L. Vittes, and S. L. M. Woo. Dynamizing static algorithms, with applications to dynamic trees and history independence. *Proc. 15th Ann. ACM-SIAM Symp. Discrete Algorithms*, 531–540, 2004.

[2] 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⟩.

[3] S. Alstrup, J. Holm, K. De Lichtenberg, and M. Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms* 1(2):243–264, 2005.

[4] 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.

[5] 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⟩.

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

[7] 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.

[8] P. J. Carstensen. Complexity of some parametric integer and network programming problems. *Math. Program.* 26:64–75, 1983.

[9] P. J. Carstensen. *The complexity of some problems in parametric, linear, and combinatorial programming*. Ph.D. thesis, Dept. of Mathematics, Univ. of Michigan, 1983.

[10] E. W. Chambers, J. Erickson, and A. Nayyeri. Homology flows, cohomology cuts. *Proc. 42nd Ann. ACM Symp. Theory Comput.*, 273–282, 2009. Full version available at http://www.cs.uiuc.edu/~jeffe/pubs/surflow.html.

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

[12] 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.

[13] G. B. Dantzig and D. R. Fulkerson. On the max-flow min-cut theorem of networks. *Linear Inequalities and Related Systems*, 215–221, 1956. Annals of Mathematical Studies 38, Princeton University Press.

[14] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271, 1959.

[15] P. Elias, A. Feinstein, and C. E. Shannon. Note on maximum flow through a network. *IRE Trans. Inform. Theory* IT-2:117–119, 1956.

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

[17] 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.

[18] L. R. Ford. Network flow theory. Paper P-923, The RAND Corporation, Santa Monica, California, August 14, 1956. Cited in [40].

[19] 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.

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

[21] A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by cancalling negative cycles. *J. Assoc. Comput. Mach.* 36(4):873–886, 1989.

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

[23] D. Gusfield. Sensitivity analysis for combinatorial optimization. Memorandum UCB/ERL M80/22, Electronics Research Laboratory, 1980. Cited in [8, 9].

[24] 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 [40].

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

[26] 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.

[27] 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.

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

[29] L. Janiga and V. Koubek. Minimum cut in directed planar networks. *Kybernetika* 28(1):37–49, 1992.

[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] R. M. Karp and J. B. Orlin. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Appl. Math.* 3:37–45, 1981.

[32] S. Khuller and J. Naor. Flow in planar graphs: A survey of recent results. *Planar Graphs*, 59–84, 1993. DIMACS Series in Discrete Math and Theoretical Computer Science 9, AMS.

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

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

[35] 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. *Proc. 20th Ann. ACM-SIAM Symp. Discrete Algorithms*, 236–245, 2009.

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

[37] G. L. Miller and J. Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.* 24(5):1002–10017, 1995.

[38] K. Mulmuley and P. Shah. A lower bound for the shortest path problem. *J. Comput. Syst. Sci.* 63(2):253–267, 2001.

[39] 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.

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

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

[42] D. M. Y. Sommerville. *An Introduction to the Geometry of N Dimensions*. Methuen & Co. Ltd, London, 1929. Republished by Dover, 1958.

[43] R. E. Tarjan and R. F. Werneck. Self-adjusting top trees. *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms*, 813–822, 2005.

[44] 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.

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

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

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

[48] H. Whitney. Planar graphs. *Fundamenta mathematicae* 21:73–84, 1933.

[49] N. E. Young, R. E. Tarjan, and J. B. Orlin. Faster parametric shortest path and minimum balance algorithms. *Networks* 21(2):205–221, 1991.

Appendix: Comparison with Borradaile and Klein

Borradaile and Klein present their maximum-flow algorithm as an instance of the Ford-Fulkerson augmenting-path approach. Their algorithm begins by computing an initial flow $\phi(0,\cdot)$ with value 0, by computing a shortest-path tree in G^* rooted at some dual vertex o incident to t^*, exactly as described in Section 2.1. Khuller *et al.* [33] prove that if G is embedded with o^* as the outer face, the residual graph of $\phi(0,\cdot)$ has no clockwise cycles.

After the initialization phase, Borradaile and Klein's algorithm repeatedly augments along the *leftmost* s-to-t path in the current residual graph. Augmenting along the leftmost residual path ensures that the residual graph has no clockwise cycles; indeed, this invariant can be taken as a definition of 'leftmost'. Their algorithm maintains a spanning tree of G that is guaranteed to contain the leftmost path.

Our algorithm can also be seen as an instance of the augmenting-path approach, if we interpret the slack of any dual e^* as the residual capacity of the corresponding primal edge e. In each iteration of the main loop, our algorithm is pushing just enough flow through the augmenting path LP to saturate one edge.

Moreover, our algorithm maintains the invariant that the residual graph has no clockwise cycles. Let C be a clockwise cycle in the current residual graph G_λ, with some face f in its interior. Lemma 2.1 implies that the dual cocycle C^* is an (o, f^*)-cut. Because the edges of T_λ are oriented away from o, at least one edge in T_λ (on the path from o to f^*) is also in C^*. But the edges of T_λ have zero slack; thus, at least one edge of C must be saturated. Similar arguments imply that our loose path LP is the leftmost s-to-t residual path in G, and that our spanning tree L_λ of loose edges is the spanning tree of G maintained by Borradaile and Klein's algorithm. Thus, our algorithm is essentially identical to Borradaile and Klein's.

(There is one unimportant difference between the two algorithms. In our notation, Borradaile and Klein unnecessarily maintain the shortest-path tree T_λ in a separate dynamic tree data structure to detect the stopping condition described by our Lemma 2.3.)