Holiest Minimum-Cost Paths and Flows in Surface Graphs* (full version)

Jeff Erickson† Kyle Fox‡ Luvsandondov Lkhamsuren§

November 3, 2017

Abstract

Let G be an edge-weighted directed graph with n vertices embedded on a surface of genus g. We describe a simple deterministic lexicographic perturbation scheme that guarantees uniqueness of minimum-cost flows and shortest paths in G. The perturbations take $O(gn)$ time to compute. We use our perturbation scheme in a black box manner to derive a deterministic $O(n \log\log n)$ time algorithm for minimum cut in *directed* edge-weighted planar graphs and a deterministic $O(g^2 n \log n)$ time proprocessing scheme for the multiple-source shortest path problem of computing a shortest path oracle for all vertices lying on a common face of a surface embedded graph. The latter result yields faster deterministic near-linear time algorithms for a variety of problems in constant genus surface embedded graphs.

Finally, we open the black box in order to generalize a recent linear-time algorithm for multiple-source shortest paths in unweighted undirected planar graphs to work in arbitrary surfaces. Our algorithm runs in $O(g^2 n \log g)$ time in this setting, and can be used to give improved linear time algorithms for several problems in unweighted undirected surface embedded graphs of constant genus including the computation of minimum cuts, shortest topologically non-trivial cycles, and minimum homology bases.

*This work was initiated at Dagstuhl seminar 16221 "Algorithms for Optimization Problems in Planar Graphs". The latest full version of this paper can be found at https://utdallas.edu/~kyle.fox/publications/unique-flow.pdf. The research presented in this paper was partially supported by NSF grants CCF-1408763 and IIS-1447554.

†Department of Computer Science, University of Illinois, Urbana-Champaign; jeffe@illinois.edu.

‡Department of Computer Science, The University of Texas at Dallas; kyle.fox@utdallas.edu. Portions of this work were done while the author was a postdoctoral associate at Duke University.

§Airbnb; lkhamsurenl@gmail.com. Portions of this work were done while this author was a student at the University of Illinois at Urbana-Champaign.

A Introduction

Many recent combinatorial optimization algorithms for directed surface embedded graphs rely on a common assumption: the shortest path between any pair of vertices is unique. The most commonly applied consequence of this assumption is that the shortest paths entering (or leaving) a common vertex do not cross one another. From this consequence, one can prove near-linear running time bounds for a variety of problems, including the computation of maximum flows [4, 6, 22, 24] and global minimum cuts [51, 52] in directed planar (genus 0) graphs as well as the computation of minimum cut oracles in planar and more general embedded graphs [3, 7] (see also Wulff-Nilsen [66]).

This assumption is also used in algorithms for the multiple-source shortest path problem introduced for planar graphs by Klein [45]. In the multiple-source shortest path problem, one is given a surface embedded graph $G = (V, E, F)$ of genus g with vertices V, edges E, and faces F. The goal is to compute a representation of all shortest paths from vertices on a common face $r \in F$ to all other vertices in the graph. Assuming uniqueness of shortest paths, multiple-source shortest paths can be computed in only $O(gn \log n)$ time [11, 45]. Algorithms for this problem can be used to solve a variety of problems in planar and more general surface embedded graphs of constant genus in near-linear time. Such results include the computation of shortest cycles with non-trivial topology [2, 11, 25, 28, 31, 33], the computation of maximum flows and minimum cuts [5, 13, 15, 26, 28, 40, 47], the computation of exact and approximate distance oracles [10, 43, 53], and even the computation of *single*-source shortest paths [46, 54].

Unfortunately, it is often difficult to actually enforce the assumption that shortest paths are unique. One popular method is to add tiny random perturbations to the lengths of edges, and then apply a variant of the Isolation Lemma of Mulmuley *et al.* [55] to argue that shortest paths are unique *with high probability*. This method is used directly by Erickson [24], Mozes *et al.* [51, 52], Cabello *et al.* [11], and the numerous papers that rely on the latter result.

As an alternative to using randomness, one may instead use a lexicographic perturbation scheme where one redefines edge lengths to be multidimensional vectors so that comparisons can be done lexicographically. One such scheme was proposed by Charnes [16] and Dantzig *et al.* [19], and variants of it have been used for computing minimum cut oracles in planar graphs [7, 34, 66]. In short, the scheme turns every edge length into an $n + 1$-dimensional vector where n is the number of edges in the graph. The first component of the vector is the true length of the edge, but then there is a single other component set to 1 based purely on the edge for which we are reassigning the length. Naively implementing the scheme adds an $O(n)$ time overhead to all operations involving edge length. There are faster ways to use the scheme depending on the application one has in mind. In particular, Cabello *et al.* [11] implement the scheme with only an $\log n$ factor increase in the running time of their multiple-source shortest path algorithm. However, these fast implementations require some fairly heavy machinery, and even implementing Dijkstra's [20] algorithm for single-source shortest paths requires that same $\log n$ factor increase in the running time and the use of relatively complex dynamic tree data structures [37, 61, 62]; see Cabello *et al.* [11, Section 6.2].

Parametric shortest paths and the leafmost rule. Several algorithms for multiple-source shortest paths in embedded graphs [11, 22, 45] and maximum flows in planar graphs [4, 22, 24] rely (at least by some interpretations) on the parametric shortest path framework introduced by Karp and Orlin [42, 67]. In short, these algorithm redefine the length of a subset of edges to increase or decrease by an amount equal to some parameter λ. The algorithms then continuously increase λ while maintaining a shortest path tree T. At certain values of λ, an edge will *pivot* into T while another edge pivots out. Uniqueness

of shortest paths guarantees the total number of pivots to be small for the algorithms mentioned above.

That said, one can sometimes avoid the need for unique shortest paths by utilizing properties of planar embeddings. Klein [45], Borradaile and Klein [4], and Eisenstat and Klein [22] all give efficient algorithms that successfully use the parametric shortest path framework without doing anything explicit to guarantee unique shortest paths. In particular, Eisenstat and Klein [22] give linear-time maximum flow and multiple-source shortest path algorithm that *cannot* take advantage of the perturbation schemes mentioned above, because they crucially rely on the edge capacities/lengths being small non-negative integers. (Weihe [64] also describes a linear-time maximum flow algorithm for unweighted undirected planar graphs, and Brandes and Wagner [8] give an algorithm for unweighted *directed* planar graphs.)

Instead of using perturbation schemes, these algorithms all take advantage of the *leafmost rule* for selecting edges to pivot into the shortest path tree T. The leafmost rule works as follows: The edges outside of T form a spanning tree C of the planar dual graph. Consider rooting C at some dual vertex (primal face); the root we choose depends upon the particular algorithm we are attempting to implement. When λ reaches a value that requires pivoting an edge into T but there are multiple appropriate candidate edges to choose from, the leafmost rule dictates that we should always select the candidate edge lying closest to a leaf of C. As a result, these algorithms all maintain *leftmost* shortest path trees. We note the leafmost rule bares a strong resemblance to Cunningham's [18] rule for maintaining a *strongly feasible basis* during network simplex.

Despite these successes, the leafmost rule and leftmost shortest path trees still do not present an ideal solution for algorithms requiring unique shortest paths. For one, these algorithms need to be designed with leftmost shortest path trees in mind. In contrast, random perturbations and lexicographic perturbation schemes can be implemented with only minor changes in how comparisons and basic arithmetic operations are performed. And perhaps more seriously, there is no obvious generalization of the leftmost shortest path trees or the leafmost rule for pivots in surface embedded graphs of non-zero genus. In particular, the complement of a spanning tree is not itself a tree in this case. Certain algorithms such as the multiple-source shortest path algorithm of Cabello *et al.* [11] appear to crucially rely on a guarantee that shortest paths really are unique.

A.1 Our results

Let G be a graph of size n embedded in an orientable surface of genus g with lengths on the edges. We present a deterministic lexicographic perturbation scheme that guarantees uniqueness of shortest paths despite using only $O(g+1)$-dimensional vectors for the perturbed edge lengths. The perturbation terms we use are all integers of absolute value $O(n)$, so our scheme can be employed in any combinatorial algorithm implemented in the word RAM model. Using our scheme increases the asymptotic running time of such algorithms by at most a factor of g.

As detailed in Section C, the perturbation vectors can be computed in $O(gn)$ time using a simple algorithm. In short, we compute a $2g$-bit signature $[e]$ for each edge e so that the sum of edge signatures along a cycle characterizes the *homology class* of that cycle with coefficients in $\mathbb{Z}$. A cycle's homology class describes how it wraps around the holes on a surface. We also compute a single integer $x(e)$ for each edge e so that given a cycle γ bounding a subset of faces $F' \subseteq F$, the sum of these integers along γ is equal to or the opposite of the number of faces in F'. This latter assignment of integers is inspired in part by results of Park and Philips [57] and Patel [58] on the minimum quotient and sparsest cut problems in planar graphs. Our perturbation vectors contain both $[e]$ and $x(e)$, and it is not difficult to show that every (simple) cycle in G has non-zero cost according to our lexicographic perturbation scheme. Uniqueness of shortest paths follows as an easy consequence.

In fact, our scheme can be used to modify the *costs* of edges in the more general minimum-cost flow problem, guaranteeing that the minimum-cost flow itself is unique. It turns out that our scheme

encourages the selection of leftmost shortest paths or minimum-cost flows in planar graphs, so we refer to the unique optimal solutions to these two problems as *homologically lexicographic least leftmost* minimum-cost paths and flows, or *holiest* paths and flows, for short.

After describing our perturbation scheme for computing holiest paths and flows, we turn to its applications. Using our scheme in a black box manner, we immediately derandomize the recent $O(n \log\log n)$ time minimum cut algorithm for directed planar graphs by Mozes *et al.* [51, 52].[1]

Our scheme can also be used in the multiple-source shortest path algorithm of Cabello *et al.* [11] for arbitrary surface embedded graphs, bringing its total running time to $O(g^2 n \log n)$. Compared to the deterministic perturbation scheme they consider, our alternative provides a factor $(\log n)/g$ improvement in running time, and the implementation is considerably simpler. In turn, we obtain the same $(\log n)/g$ factor improvement to the deterministic versions of nearly every algorithm that uses their data structure. Cabello *et al.* actually require a slightly stronger condition than mere uniqueness of shortest paths, but we are able to show our scheme guarantees the condition holds in Section E. The exposition in that section also helps set us up for our remaining results.

It turns out that holiest paths and flows are not only leftmost objects of minimum-cost, but our perturbation scheme also *forces* the aforementioned parametric shortest path based algorithms to choose leafmost edges during pivots. See Section F. Based on this observation, we generalize the linear-time multiple-source shortest path algorithm of Eisentstat and Klein [22] for small integer edge lengths so that it works in surface embedded graphs of arbitrary genus. Our generalization runs in $O(g(gn \log g + L))$ time where L is the sum of the integer edge lengths. Like Eisenstat and Klein, we must assume every edge has a reversal, essentially modeling unweighted undirected graphs in the case that edge lengths are all 1.

The high level idea behind our algorithm is to generalize the leafmost rule using our new perturbation scheme. When we must pivot an edge into the holiest shortest path tree T, we partition the set of candidate edges into collections based on the homology class of their fundamental cycles with T. We pick a collection based on the homology signature portion of our scheme's perturbation vectors, and then essentially apply the leafmost rule to edges *within that collection* to select the one that enters T. Finding the leafmost edge requires individually checking edges to see which ones can be pivoted into T. Fortunately, we can charge the time spent checking these edges to changes in the *homotopy* class of the holiest paths to these edges' endpoints. We give our algorithm and analysis in Section G.

Finally, using our linear-time algorithm for multiple-source shortest paths, we immediately obtain new linear time algorithms for a variety of problems in unweighted undirected surface embedded graphs, including the computation of s,t- and global minimum cuts, shortest cycles with non-trivial embeddings, and shortest homology bases. By combining known works, one could obtain $g^{O(g)}n$ time algorithms for each of these cases, but our cut algorithms run in $2^{O(g)}n$ time, and our other algorithms run in $O(\text{poly}(g)\ n)$ time. In particular, ours are the first algorithms for the latter problems that simultaneously have polynomial dependency on g and linear dependency on n.

A.2 Additional related work

Although they may sometimes go by different names such as *uppermost* or *rightmost*, the idea of computing leftmost paths and flows in planar graphs appears as far back as the original maximum flow-minimum cut paper of Ford and Fulkerson [32]. Several researchers have designed efficient algorithms for specializations of the maximum flow problem in planar graphs using this idea [1, 4, 35, 39, 59, 64, 65]. There is a deep connection between flows in planar graphs and shortest paths in their duals (see, for

[1] We admit that Mozes *et al.* were aware of the current work as they were writing their paper, so they may not have felt a strong need to derandomize their algorithm themselves.

example, Venkatesan [63]). As far as we are aware, though, Klein [45] was the first to apply the idea of directly computing ____most shortest path trees.

Khuller, Naor, and Klein [44] observed that the set of integral *circulations* in a planar graph form a distributed lattice, and solutions to the minimum cost circulation problem form a sublattice. Indeed, a planar circulation is the boundary of a potential function (or 2-chain) on the faces, and the meet and join can be defined by taking the component-wise max and min of the potential function, respectively. Many of the algorithms mentioned above actually find the top or bottom element in the (sub)lattice. Depending on which specifics one chooses, our lexicographic perturbation scheme simply enforces that one choose the minimum flow or circulation according to this sublattice. Matuschke and Peis [49] show that the left/right relation on s,t-paths in planar graphs also forms a lattice.

B Preliminaries

We begin with an introduction to surface embedded graphs. For more background we refer the reader to books and surverys [17, 21, 36, 50, 56, 68] related to the topic.

Surfaces. A ***surface*** Σ or 2-manifold with boundary is a compact Hausdorff space where every point lies in an open neighborhood homeomorphic to either the Euclidean plane or the closed half plane. The points whose neighborhoods are homeomorphic to the closed half plane constitute the ***boundary*** of the surface. Every component of the boundary is homeomorphic to the unit circle. A ***cycle*** in the surface Σ is a continuous function $\gamma : S^1 \to \Sigma$ where S^1 is the unit circle. Cycle γ is ***simple*** if γ is injective. A ***path*** P in the surface Σ is a continuous function $P : [0,1] \to \Sigma$; again, p is simple if it is injective. A ***loop*** is a path P such that $P(0) = P(1)$; in other words, it is a cycle with a designated base point. The ***genus*** of the surface Σ, which we denote as g, is the maximum number of pairwise disjoint simple cycles $\gamma_1, \ldots, \gamma_g$ in Σ such that $\Sigma \setminus (\gamma_1 \cup \cdots \cup \gamma_g)$ is connected. Surface Σ is ***non-orientable*** if any subset of Σ is homeomorphic to the Möbius band. Otherwise, Σ is ***orientable***. Up to homeomorphism, a surface is characterized by its genus, the number of boundary components, and whether or not it is orientable. We directly work only with orientable surfaces in this paper.[2]

Let P_1 and P_2 be two paths in Σ. Paths P_1 and P_2 ***cross*** if no continuous infinitesimal perturbation makes them disjoint. Otherwise, we call them ***non-crossing***. They are ***homotopic*** if one can be continuously deformed into the other without changing their endpoints. More formally, there must exist a ***homotopy*** between them, defined as a continuous map $h : [0,1] \times [0,1] \to \Sigma$ such that $h(0,\cdot) = p$ and $h(1,\cdot) = q$. Homotopy defines an equivalence relation over the set of paths with any fixed pair of endpoints. A cycle is ***contractible*** if it is homotopic to a constant map, and a loop is contractible if it is homotopic to its base point. The concatenation of a path P and loop γ with common endpoint is homotopic to P if and only if γ is contractible.

Graph embeddings. The ***surface embedding*** of an *undirected* graph G with vertex set V and edge set E is a drawing of G on a surface Σ which maps vertices to distinct points on Σ and edges to internally disjoint simple paths whose endpoints lie on their incident vertices' points. A ***face*** of the embedding is a maximally connected subset of Σ that does not intersect the image of G. An embedding is ***cellular*** if every face is homeomorphic to an open disc. In a cellular embedding, every boundary component is

[2] Cabello *et al.* [11] describe a reduction for their multiple-source shortest path algorithm in graphs embedded in non-orientable surfaces to the same problem in graphs embedded in orientable surfaces. We may apply our perturbation scheme or linear-time algorithm *after* applying their reduction in order to extend at least some of our results to graphs embedded in non-orientable surfaces.

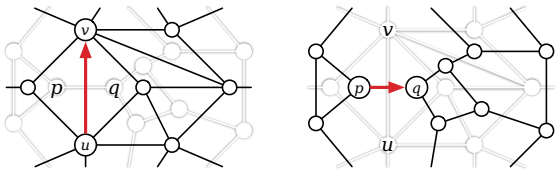

Figure 1. A dart $u{\to}v$ in the primal graph G and the corresponding dart $p{\uparrow}q$ in the dual graph G^*.

covered by the image of a cycle in G. Let F be the set of faces of a cellular embedding, and let b be the number of boundary components. By Euler's formula, $|V|-|E|+|F| = 2-2g-b$.

To more easily support directed graphs, we will assume G is connected and that its embedding is given as a ***rotation system***. These are sometimes referred to as ***combinatorial embeddings*** as well (see, for example, Eisenstat and Klein [22]). Let $\vec{E}$ denote a collection of "directed edges" we refer to as ***darts***. Let $rev : \vec{E} \to \vec{E}$ be an involution on the darts we refer to as their ***reversals***. Each ***edge*** e is a cycle in the involution rev. We refer to one dart in e's involution as the ***canonical dart*** of e and denote it by $\vec{e}$. In addition to rev, we have a permutation $\pi : \vec{E} \to \vec{E}$. Each cycle of π gives the counterclockwise cyclic ordering of darts "directed into" a vertex v. We refer to v as the ***head*** of the darts in v's permutation cycle. Vertex v is the ***tail*** of these darts' reversals. Cycles of the permutation $rev \circ \pi$ give the *clockwise* ordering of darts around each face of the embedding. We use the notation $G = (V, E, F)$ to denote a surface embedded graph G with vertex set V, edge set E, and face set F. From here on, we refer to such triples simply as ***graphs***. In this setting, we can actually define the ***genus*** g of G to be the *solution* to $|V|-|E|+|F| = 2-2g$.

Given a graph $G = (V, E, F)$, we define the ***dual graph*** $G^* = (F, E, V)$. The given graph G is sometimes called the ***primal*** graph. G^* contains a vertex for every face of G, an edge for every edge of G, and a vertex for every face of G. Two dual vertices are connected by a dual edge if and only if the corresponding primal faces are separated by the corresponding primal edge. In terms of combinatorial embeddings, the vertices of the dual graph are the orbits of the permutation $rev \circ \pi$; moreover, the cycles of $rev \circ \pi$ define the cyclic order of darts directed into each dual vertex. The drawing of dart d in the dual graph goes left to right across the drawing of d in the primal graph.

For notational convenience, we will not distinguish between primal faces and dual vertices, primal and dual darts/edges, or dual vertices and primal faces. However, we will generally use the variables u, v, w, x, and y to denote primal vertices/dual faces, and the variables o, p, q, and r to denote dual vertices/primal faces. We let $u{\to}v$ denote a dart with tail u and head v in the primal graph, and let $p{\uparrow}q$ denote a dart with tail p and head q in the dual graph. Finally, uv and $p|q$ denote edges between vertices u and v or between dual vertices p and q, respectively. Their canonical darts are $u{\to}v$ and $p{\uparrow}q$, respectively.

Flows, homology, and final definitions. Flows are naturally defined either as *non-negative* functions on the darts (without loss of generality equal to zero on at least one dart of each edge) or as *anti-symmetric* functions on the darts (where the values on the two darts of each edge sum to zero). It will prove convenient to use the non-negative formulation to describe flows in the primal graph G and the antisymmetric formulation to describe flows in the dual graph G^*.[3]

A (primal) ***flow*** $f : \vec{E} \to \mathbb{R}^+$ is an assignment of non-negative real values to the darts of G. The ***imbalance*** $\delta f : V \to \mathbb{R}$ of flow f is the net flow going into each vertex. Formally, $\delta f(v) = \sum_{u\to v} f(u{\to}v) - \sum_{v\to w} f(v{\to}w)$. Flow f is a ***circulation*** if $\delta(v) = 0$ for all $v \in V$. A ***potential function*** $\alpha : F \to \mathbb{R}$ is an assignment of real values to *faces* of G. We say flow f is a ***boundary flow*** of potential function α if for every edge $uv = q|p$, we have $f(u{\to}v) - f(v{\to}u) = \alpha(p) - \alpha(q)$. In other words, high potentials

[3]This apparent asymmetry is actually a consequence of LP duality. If we formulate minimum-cost flows in G as a linear program using one formulation, the dual LP describes minimum-cost flows in G^* in the other formulation!

to the *right* of edges encourage high flow values through their canonical darts and low flow values through canonical darts' reversals. All boundary flows are circulations. Those familiar with concepts from algebraic topology may recognize the similarity between flows, imbalances, and potentials functions with 1-chains, boundaries of 1-chains, and 2-chains, respectively.[4] Two flows f_1 and f_2 are ***homologous*** if their componentwise difference is the boundary of some potential function. Similar to homotopy, homology defines an equivalence relation over the set of flows that is isomorphic with $\mathbb{R}^{2g}$.

A ***dual flow*** $x : \vec{E} \to \mathbb{R}$ assigns real values to the darts of the *dual* graph G^* such that $x(d) = -x(rev(d))$ for every dart d. Equivalently, we consider a dual flow to be a function on the *edges* of G^* by defining $x(e) = x(\vec{e})$. The ***dual imbalance*** $\partial x : F \to \mathbb{R}$ of dual flow x is the total dual flow going clockwise around each dual face, or equivalently, into each dual vertex. Formally, $\partial x(p) = \sum_{q \uparrow p} x(q \uparrow p)$. Let $F' \subseteq F$ be any subset of faces. Somewhat abusing notation, we let $\partial^-(F')$ denote the ***clockwise neighborhood*** of F' so that $\partial^-(F') = \{p \uparrow q : p \notin F', q \in F'\}$. Thus, darts of $\partial^-(F')$ are directed clockwise around the boundary of F' in the primal graph G and enter F' in the dual graph G^*.

Let $c : \vec{E} \to \mathbb{R}$ be a ***dart cost function***, let $\mu : \vec{E} \to \mathbb{R}^+$ be a ***dart capacity function***, and let $b : V \to \mathbb{R}$ be a ***vertex demand function***. The ***cost*** of a flow f is $c(f) = \sum_{d \in \vec{E}} f(d) \cdot c(d)$. A flow f is feasible with respect to μ and b if for all darts $d \in \vec{E}$ we have $f(d) \leq \mu(d)$ and for all vertices $v \in V$ we have $\delta f(v) = b(v)$. A ***minimum-cost flow*** with respect to c, μ, and b is a feasible flow of minimum cost (if it exists).

A (directed) ***path*** P in G is a sequence of darts $\langle v_0 \to v_1, v_1 \to v_2, \ldots, v_{k-1} \to v_k \rangle$ where consecutive darts share vertices. We often abuse terminology and identify a path with its drawing in G's embedding. A path is ***simple*** if it does not repeat any vertices, except possibly its first and last vertex. The ***concatenation*** of paths P_1 and P_2 is denoted $P_1 \circ P_2$. Path P is a ***cycle*** if $v_0 = v_k$. Abusing notation, we may treat P as a flow where $P(d)$ is equal to the number of times dart d appears in P. Given a vertex $s \in V$, let $\mu : \vec{E} \to \mathbb{R}^+$ be a capacity function where $\mu(d) = \infty$ for all $d \in \vec{E}$, and let $b : V \to \mathbb{R}$ be a demand function where $b(v) = 1$ for all $v \neq s$ and $b(s) = 1 - |V|$. Given dart costs $c : \vec{E} \to \mathbb{R}$ where no cycle has negative cost, the ***shortest paths*** from s to all other vertices can be defined as the set of paths starting at s and composing the minimum-cost flow with respect to μ and b. Let $d_c(s,t)$ denote the distance from s to t according to costs c. Let $\boldsymbol{\sigma(s,t)}$ denote the shortest path from s to t.

A ***spanning tree*** T of G is a subset of edges that form a tree containing every vertex. If we assign a ***root*** s to T, then we consider the darts of T to be oriented away from the s. Given an edge $e \notin T$, the ***fundamental cycle*** of e with T is the unique simple cycle of edges in $T + e$. If $e \in T$, then its fundamental cycle is empty. Given a dart d of an edge e, its fundamental cycle with T is the orientation of e's fundamental cycle that contains d. A ***spanning cotree*** C of G is a subset of edges that form a spanning tree in the dual graph. A ***tree-cotree decomposition*** [23] of G is a partition of E into 3 disjoint edge subsets T, L, C, where T is a spanning tree of G, C is a spanning cotree, and L is a set of $2g$ leftover edges.

Given a vector a, let a_i denote the ith component of a. Let (T, L, C) be an arbitrary tree-cotree decomposition of G. Let $\Gamma = \langle \gamma_1, \gamma_2, \ldots, \gamma_{2g} \rangle$ be some ordering of the $2g$ dual fundamental cycles of edges in L with C, and orient each γ_i in an arbitrary direction. The ***homology signature*** $[e]$ of an edge $e \in E$ with respect to Γ is a $2g$-dimensional integer vector where $[e]_i = 1$ if $\vec{e} \in \gamma_i$, $[e]_i = -1$ if $rev(\vec{e}) \in \gamma_i$, and $[e] = 0$ otherwise. We can compute homology signatures in $O(gn)$ time. Given a dart d of edge e, we define the homology signature of d so that $[d] = [e]$ if $d = \vec{e}$ and $[d] = -[e]$ otherwise. Homology signatures given an implicit representation of a *cohomology basis* in G. See Erickson and Whittlesey [30] and subsequest papers [2, 9, 14, 28]. The homology signature of a flow f is $[f] = \sum_{d \in \vec{E}} f(d) \cdot [d]$. Finally, we have the following lemma, easily derived by modifying known results for homology signatures.

[4] This similarity is somewhat more natural with the antisymmetric formulation of flows.

Lemma B.1. *Let f_1 and f_2 be two flows. Flow f_i is the boundary of some potential function if and only if $[f_i] = 0$. Further, f_1 and f_2 are homologous if and only if $[f_1] = [f_2]$.*

C Holiest Perturbation

Let $G = (V, E, F)$ be a graph of size n and genus g. Our lexicographic perturbation scheme relies on the properties of certain dual flows we refer to as drainages. Given a designated face $r \in G$, we define a ***drainage*** as a dual flow x where $\partial x(q) < 0$ for all $q \in F \setminus \{r\}$. The definition of a drainage immediately implies $\partial x(r) = -\sum_{q \in F \setminus \{r\}} \partial x(q)$. Because r is the only face with positive dual imbalance with respect to x, we refer to r as the ***sink*** of the drainage x.

We now describe our perturbation scheme. Let $c : \vec{E} \to \mathbb{R}$ be a dart cost function. We compute a set of homology signatures for the darts in $O(gn)$ time as described in Section B. We then compute a drainage x of G^* in $O(n)$ time. While any drainage will do, we describe one here that is easily computed. We begin by computing an arbitrary spanning tree C of G^*. Let $r \in F$ be an arbitrary face. We define x as if each face $p \in F \setminus \{r\}$ is sending one unit of flow along C to r.

Formally, we set the dual flow for each dart $p{\uparrow}q \in \vec{E}$ as follows. We consider the sink r to be the root of the cotree C. If $(p{\uparrow}q)$'s edge is not in C, then $x(p{\uparrow}q) = 0$. Otherwise, if q is the parent of p in the dual spanning tree C, then $x(p{\uparrow}q)$ is the number of descendants of p (including p itself) in C; otherwise, $x(p{\uparrow}q)$ is the negation of the number of descendants of q in C. One may easily verify that $\partial x(q) = -1$ for all $q \in F \setminus \{r\}$ and $\partial x(r) = \ell - 1$.

We now redefine the costs of darts in G. Intuitively, we add a sequence of progressively smaller infinitesimal values to the cost of each dart based partially on the homology signatures of their edges and the dual flow they carry from the drainage x. More concretely, we define a new dart cost function $c' : \vec{E} \to \mathbb{R} \times \mathbb{N}^{2g+2}$ as follows. Let $\frown$ denote the concatenation of two vectors, and define

$$c'(d) := (c(d), 1) \frown [d] \frown (x(d))$$

The definition for the cost of a flow f can be modified easily to work with our new cost function: $c'(f)$ is the vector $\sum_{d \in \vec{E}} f(d) \cdot c'(d)$. We refer to the components of dart and flow costs determined by homology signatures as the ***homology parts*** of the costs and last component as the ***face part***. Comparisons between dart and flow costs are performed lexicographically. As a consequence, any minimum-cost flow with respect to c' is also a minimum-cost flow with respect to the original scalar cost function c. Intuitively, minimizing the cost of a flow according to c' first minimizes the original cost according to c, then minimizes the sum of the darts' flow values, then lexicographically minimizes the homology class of the flow, and finally chooses the leftmost flow subject to all other conditions. In particular, when $g = 0$, one is computing a leftmost minimum-cost flow (after also minimizing the sum of darts' flows). The following lemma is immediate.

Lemma C.1. *For any flow f and $j \in \{1, \ldots, 2g\}$, we have $[f]_j = c'_{j+2}(f)$. In addition,*

$$c'_{2g+3}(f) = \sum_{e \in E} [f(\vec{e}) \cdot x(\vec{e}) + f(rev(\vec{e})) \cdot x(rev(\vec{e}))].$$

Computing the perturbations takes $O(gn)$ time total. The time for every addition, multiplication, and comparison is now $O(g)$ instead of $O(1)$. For planar graphs in particular, this scheme requires only linear preprocesing time, and combinatorial algorithms relying on the new costs do not have higher asymptotic running times. Note that the cost of each dart d is strictly larger than $(c(d), 0, \ldots, 0)$. No negative-cost directed cycles are created, even when some directed cycles had length 0 originally, meaning shortest paths are still well-defined. In fact, the perturbation scheme does not create any new negative-cost darts,

so combinatorial shortest path algorithms relying on non-negative dart costs still function correctly. As stated, however, these algorithms and those for negative costs do slow down by a factor of g.

C.1 Analysis

We now prove our perturbation scheme guarantees uniqueness of minimum-cost flows and shortest paths as promised. We begin by discussing the former as the latter follows as an easy consequence. The key observation behind our proof is that drainages encode the total imbalance of vertices lying on one side of a dual cut. In turn, we use this observation to show that every non-trivial circulation has a part of non-zero cost after using our perturbation scheme. This former observation is a slight generalization of one by Patel [58, Lemma 2.4] who in turn generalized a result for planar graphs by Park and Phillips [57]. We could use Patel's result directly for the tree-based drainage described above. However, we are able to give a short proof of the more general lemma below.

Lemma C.2. *Let x be a drainage with sink r, and let $F' \subseteq F$. We have*

$$\sum_{d \in \partial^-(F')} x(d) = \sum_{q \in F'} \partial x(q) = -\sum_{q \in F \setminus F'} \partial x(q).$$

In particular, for non-empty $F' \neq F$, we have $\sum_{d \in \partial^-(F')}$ is positive if $r \in F'$ and negative otherwise.

Proof: We have

$$\begin{aligned}
\sum_{d \in \partial^-(F')} x(d) &= \sum_{p\uparrow q \in \vec{E} : p \notin F', q \in F'} x(p\uparrow q) + \sum_{p|q \in E : p, q \in F'} [x(p|q) - x(p|q)] \\
&= \sum_{p\uparrow q \in \vec{E} : p \notin F', q \in F'} x(p\uparrow q) + \sum_{p|q \in E : p, q \in F'} [x(p\uparrow q) + x(q\uparrow p)] \\
&= \sum_{p\uparrow q \in \vec{E} : q \in F'} x(p\uparrow q) \\
&= \sum_{q \in F'} \sum_{p\uparrow q \in \vec{E}} x(p\uparrow q) \\
&= \sum_{q \in F'} \partial x(q).
\end{aligned}$$

The final statement follows easily from the definition of drainages. □

Fix a cost function $c : \vec{E} \to \mathbb{R}$, and let c' be the perturbation of c defined above. Also fix a capacity function $\mu : \vec{E} \to \mathbb{R}^+$ and a demand function $b : V \to \mathbb{R}$. We give the following lemma, which immediately implies the uniqueness of minimum-cost flows.

Lemma C.3. *Let f_1 and f_2 be distinct feasible flows with respect to μ and b. There exists a feasible flow f such that at least one of $c'(f) < c'(f_1)$ or $c'(f) < c'(f_2)$ is true.*

We emphasize that our perturbation scheme does not guarantee *all* feasible flows have distinct costs, and it may be that $c'(f_1) = c'(f_2)$. However, we would then have f costing strictly less than both f_1 and f_2, implying neither f_1 nor f_2 is a minimum-cost feasible flow.

Proof: We will prove existence of a circulation $\hat{f}$ such that $f_i + \hat{f}$ is feasible for some $i \in \{1, 2\}$ and $c'(\hat{f}) < 0$. We set $f = f_i + \hat{f}$, proving the lemma.

Let $\tilde{f} = f_2 - f_1$. Both f_2 and f_1 are feasible with respect to demand function b, so $\tilde{f}$ must be a non-trivial circulation. Further, for any dart d and scalar a with $0 \le a \le 1$, we have

$$f_1(d) + a\tilde{f}(d) = f_1(d) + a(f_2(d) - f_1(d)) \ge \min\{f_1(d), f_1(d) + f_2(d) - f_1(d)\} \ge 0 \text{ and}$$
$$f_1(d) + a\tilde{f}(d) = f_1(d) + a(f_2(d) - f_1(d)) \le \max\{f_1(d), f_1(d) + f_2(d) - f_1(d)\} \le \mu(d).$$

In other words, we can add any circulation consisting of scaled down components of $\tilde{f}$ to f_1 and still have a feasible flow. Similarly, we can add any circulation consisting of scaled down components of $-\tilde{f}$ to f_2 and still have a feasible flow. We now consider two cases.

Case 1: $[\tilde{f}] \neq 0$. Let $[\tilde{f}]_j$ be non-zero. Lemma C.1 implies $c'_{j+2}(\tilde{f})$ is also non-zero, further implying $c'(\tilde{f})$ is itself non-zero. If $c'(\tilde{f}) < 0$, then let $\hat{f} = \tilde{f}$. Otherwise, let $\hat{f} = -\tilde{f}$.

Case 2: $[\tilde{f}] = 0$. We consider two subcases.

First, suppose there exists an edge $e \in E$ such that $\tilde{f}(\vec{e}) = \tilde{f}(rev(\vec{e})) > 0$. Let $f_e : \vec{E} \to \mathbb{R}$ be a flow that is everywhere-zero except $f_e(\vec{e}) = f_e(rev(\vec{e})) = \tilde{f}(\vec{e})$. Then, $c'_2(f_e) = 2\tilde{f}(\vec{e})$, implying $c'(f_e)$ is non-zero. If $c'(f_e) < 0$, then let $\hat{f} = f_e$. Otherwise, let $\hat{f} = -f_e$.

Now, suppose there is no such edge e as defined above. Then, Lemma B.1 implies $\tilde{f}$ is a boundary flow for some non-trivial potential function α. Let $\underline{\alpha} = \min_{q \in F} \alpha(q)$ and $\overline{\alpha} = \max_{q \in F} \alpha(q)$. Because α is non-trivial, at least one of $\underline{\alpha}$ and $\overline{\alpha}$ is non-zero. Assume $\overline{\alpha} \neq 0$; the other case is similar. Let $F_{\overline{\alpha}} = \{q \in F : \alpha(q) = \overline{\alpha}\}$, and let $\vec{E}_{\overline{\alpha}} = \{p \uparrow q \in \vec{E} : p \in F \setminus F_{\overline{\alpha}}, q \in F_{\overline{\alpha}}\}$. For each dart $p \uparrow q \in \vec{E}_{\overline{\alpha}}$, we have $\tilde{f}(p \uparrow q) - \tilde{f}(q \uparrow p) > 0$, because $\alpha(q) > \alpha(p)$.

Let $\varepsilon = \min_{d \in \vec{E}_{\overline{\alpha}}} (\tilde{f}(d) - \tilde{f}(rev(d)))$. For each dart $d \in \vec{E}_{\overline{\alpha}}$, let $a_d = \varepsilon / (\tilde{f}(d) - \tilde{f}(rev(d)))$. Note that $0 \le a_d \le 1$. Finally, let $f_\varepsilon : \vec{E} \to \mathbb{R}$ be a flow that is everywhere-zero except for each dart $d \in E_{\overline{\alpha}}$, we have $f_\varepsilon(d) = a_d \tilde{f}(d)$ and $f_\varepsilon(rev(d)) = a_d \tilde{f}(rev(d))$; in other words, $f_\varepsilon(d) - f_\varepsilon(rev(d)) = \varepsilon$.

Let x be the drainage used to define c'. Lemmas C.1 and C.2 imply $c'_{2g+3}(f_\varepsilon) = \varepsilon \sum_{d \in \partial^-(F_{\overline{\alpha}})}$. Set $F_{\overline{\alpha}}$ is a non-empty strict subset of F, because $\tilde{f}$ is non-trivial. Therefore, Lemma C.2 also implies $c'_{2g+3}(f_\varepsilon)$ is non-zero, meaning $c'(f_\varepsilon)$ is also non-zero. If $c'(f_\varepsilon) < 0$, then let $\hat{f} = f_\varepsilon$. Otherwise, let $\hat{f} = -f_\varepsilon$. □

Theorem C.4. *Let $G = (V, E, F)$ be a graph of genus g, let $c : \vec{E} \to \mathbb{R}$ be a dart cost function, and let $c' : \vec{E} \to \mathbb{R}$ be the output of our lexicographic perturbation scheme on c. Let $\mu : \vec{E} \to \mathbb{R}^+$ and $b : V \to \mathbb{R}$ be a dart capacity and vertex demand function, respectively. The minimum-cost feasible flow with respect to c', μ, and b is unique and is a minimum-cost feasible flow with respect to c as well.*

Recall, the shortest s,t-path problem is a special case of minimum-cost flow where for each dart d, $\mu(d) = \infty$. In addition, all demands are zero except $b(t) = -b(s) = 1$. Every directed cycle has its cost strictly increase, so if c has no negative-length directed cycles, then c' has no negative or even *zero*-length directed cycles. Any feasible flow with a directed cycle γ can be made cheaper by removing γ. The unique minimum-cost flow with respect to c' guaranteed by Theorem C.4 is a directed path from s to t.

Corollary C.5. *Let $G = (V, E, F)$ be a graph of genus g, let $c : \vec{E} \to \mathbb{R}$ be a dart cost function, and let $c' : \vec{E} \to \mathbb{R}$ be the output of our lexicographic perturbation scheme on c. Let $s, t \in V$. The shortest s,t-path with respect to c' is unique and is a shortest s,t-path with respect to c as well.*

From here on, we refer to the unique minimum-cost flows and shortest paths guaranteed by our perturbation scheme as ***holiest*** flows and paths.

D Minimum Cut in Directed Planar Graphs

As discussed in the introduction, our perturbation scheme can be used in a black box fashion to immediately derandomize the $O(n \log\log n)$ time minimum cut algorithm of Mozes *et al.* [51, 52] for directed planar graphs. The only change necessary to derandomize their algorithm is to guarantee uniqueness of shortest paths in the *dual graph*.

Corollary D.1. *Let $G = (V, E, F)$ be a planar graph of size n, and let $c : \vec{E} \to \mathbb{R}$ be a dart cost function. There exists a deterministic algorithm that computes a global minimum cut of G with respect to c in $O(n \log\log n)$ time.*

E Multiple-Source Shortest Paths

Our scheme can be used in a black box fashion in the multiple-source shortest path algorithm of Cabello *et al.* [11]. However, they depend on slightly more than just uniqueness of shortest paths. Our perturbation scheme does guarantee this property, but we must first describe their algorithm in order to explain what that property is and why our scheme guarantees it. Understanding their algorithm is also a crucial first step in describing our linear-time algorithm for unit cost embedded graphs of constant genus. In order to more cleanly explain our linear-time algorithm in latter sections, we describe a slight variant of Cabello *et al.*'s algorithm. This variant is based on linear-time multiple-source shortest path algorithm of Eisenstat and Klein [22] for planar graphs with small integer dart costs.

Let $G = (V, E, F)$ be an embedded graph of size n and genus g, and let $c : \vec{E} \to \mathbb{R}$ be a cost function on the darts. Let $r \in F$ be an arbitrary face of G from whose vertices we want to preprocess shortest paths with regard to c. The multiple-source shortest path algorithm begins by computing a shortest path tree T rooted at an arbitrary vertex of r. The algorithm proceeds by iteratively changing the source of the shortest path tree to each of the vertices in order around r; each change is implemented as a sequence of ***pivots*** wherein one dart $x{\to}y$ enters T and another dart $w{\to}y$ leaves.

Consider one iteration of the algorithm where the source moves from a vertex u to a vertex v. Immediately before moving the source, the algorithm performs a ***special pivot***. Let $x{\to}v$ be the predecessor dart of v in T. During the special pivot, the algorithm removes dart $x{\to}v$ from T and adds dart $v{\to}u$; afterward, T is rooted at v. Let $\lambda \in \mathbb{R}$, and let $c_\lambda : \vec{E} \to \mathbb{R}$ be a parameterized cost function where $c(v{\to}u) = \lambda$ and $c_\lambda(d) = c(d)$ for all $d \neq v{\to}u$. This special pivot is accompanied by temporarily redefining the dart costs in terms of c_λ with λ initially set to $-d_c(u, v)$. Changing the costs in this way guarantees that T is a shortest path tree rooted at v *given dart $v{\to}u$ has cost λ*.

Conceptually, the rest of the iteration is performed by continuously increasing λ until it reaches $c(v{\to}u)$, the original cost of $v{\to}u$, and maintaining T as a shortest path tree as λ is increased. Following convention from Cabello *et al.* [11], we say a vertex x is ***red*** if the v to x path in T uses dart $v{\to}u$; otherwise, the vertex is ***blue***. Let d_λ denote d_{c_λ} for simplicity. Define the ***slack*** of dart $x{\to}y$ with regard to λ to be

$$slack_\lambda(x{\to}y) := d_\lambda(v, x) + c_\lambda(x{\to}y) - d_\lambda(v, y).$$

For any $\lambda \in \mathbb{R}, x{\to}y \in \vec{E}$, we have $slack_\lambda(x{\to}y) \geq 0$. We say dart $x{\to}y$ is ***tense*** if $slack_\lambda(x{\to}y) = 0$. A spanning tree T' rooted at v is a shortest path tree if and only if every dart in T' is tense. Dart $x{\to}y$ is ***active*** if $slack_\lambda(x{\to}y)$ is decreasing in λ. A dart $x{\to}y$ is active if and only if x is blue and y is red [11, Lemma 3.1]. All active darts see the same rate of slack decrease as λ rises.

As λ increases, it reaches certain critical values where an active dart $x{\to}y$ becomes tense. The algorithm then performs a pivot by inserting $x{\to}y$ into T and removing the original predecessor $w{\to}y$ of y. Because $x{\to}y$ is tense when the pivot occurs, T remains a shortest path tree rooted at v. Using

appropriate dynamic-tree data structures [37, 60–62], these critical values for λ can be computed and pivots can be performed in amortized $O(\log n)$ time per pivot. Eisenstat and Klein use simpler data structures for the case of planar graphs with small integer costs; see Section F for details.

Across all iterations, the total number of pivots is $O(gn)$ [11, Lemma 4.3]. Therefore, between performing pivots and some $O(gn \log n)$ time additional work, the algorithm of Cabello *et al.* spends $O(gn \log n)$ time total. However, their algorithm and analysis depend upon two genericity assumptions: all vertex-to-vertex shortest paths are unique, and exactly one dart becomes tense at each critical value of λ.

Suppose we apply our lexicographic perturbation scheme so we are maintaining the holiest shortest path tree T. Let $c' : \vec{E} \to \mathbb{R} \times \mathbb{N}^{2g+2}$ be the perturbed costs. Corollary C.5 guarantees that the first assumption is enforced. For the second assumption, we prove the following lemma.

Lemma E.1. *Consider an iteration where the source of T moves from vertex u to vertex v. For each value of λ, there is at most one active dart of minimum slack.*

Proof: Suppose there is at least one active dart. Let $\mu : \vec{E} \to \mathbb{R}^+$ be the dart capacities and $b : V \to \mathbb{R}$ be the vertex demands for shortest paths rooted at v, and let f be the minimum-cost flow with respect to c'_λ, μ, and b that uses darts of T. Let $b_p : V \to \mathbb{R}$ ('p' stands for pivot) be a vertex demand function that is zero-everywhere except $b'(u) = -b'(v) = 1$, and let $c_p : \vec{E} \to \mathbb{R}^+$ be a *residual* cost function where $c_p(v{\to}u) = \infty$, $c_p(d) = -c'_\lambda(rev(d))$ if $rev(d) \neq v{\to}u$ and $rev(d)$ lies on a shortest path according to c'_λ, and $c_p(d) = c'_\lambda(d)$ otherwise. Finding the active dart of minimum slack is equivalent to computing a holiest flow with respect to c_p, μ, and b_p. However, the holiest flow is unique by Theorem C.4. □

After applying our perturbation scheme, the time to do basic operations on costs increases by a factor of g. In addition, the graph plus dart costs now take $O(gn)$ space to store. We conclude:

Theorem E.2. *Let $G = (V, E, F)$ be a graph of size n and genus g, let $c : \vec{E} \to \mathbb{R}$ be a dart cost function, and let $r \in F$ be any face of G. We can deterministically preprocess G in $O(g^2 n \log n)$ time and $O(gn)$ space so that the length of the shortest path from any vertex incident to r to any other vertex can be retrieved in $O(g \log n)$ time.*

F The Leafmost Rule in Planar Graphs

In the previous section, we discussed using our perturbation scheme to efficiently solve the multiple-source shortest path problem in embedded graphs. Curiously, perturbations (deterministic or randomized) are not required for algorithms designed to compute multiple-source shortest paths exclusively in *planar* graphs [22, 45]. In fact, the linear-time algorithm of Eisenstat and Klein [22] *cannot* rely on perturbation schemes as it crucially depends upon all dart costs being small non-negative integers. Instead, these algorithms rely on the *leafmost* rule for selecting darts to pivot into the shortest path tree. The leafmost rule is also used in efficient s,t-maximum flow algorithms based on parametric shortest paths in the dual graph [4, 22, 24].

Let $G = (V, E, F)$ be a connected planar graph of size n, and let $c : \vec{E} \to \mathbb{R}$ be a cost function on the darts. Let T be the shortest path tree maintained while running Section E's multiple-source shortest path tree algorithm along face $r \in F$. Suppose we are moving the source of T from vertex u to vertex v, and let $v{\to}u = r{\uparrow}q$. Let C be a spanning tree of G^* complementary to T with darts oriented toward r. In other words, the darts of T and C belong to distinct edges. The set of active darts are precisely the darts in the q,r-path through C [11]. In this setting, the leafmost rule says we should pivot in the active dart of minimum slack that lies closest to a leaf of C; in other words, we choose the minimum slack dart encountered first on the q to r path through C. Following the leafmost rule guarantees we always maintain *leftmost* shortest path trees.

F.1 Modifying the perturbation scheme to explain the leafmost rule

We briefly return to the problem of computing multiple-source shortest paths around a face r in a graph $G = (V, E, F)$ of arbitrary genus g. Consider the following slight modifications to our perturbation scheme described in Section C. First, the drainage x used to define the perturbed costs is required to use r as its sink. We can easily compute such an drainage using a dual spanning tree as described in Section C. Second, we forgo the $+1$ added as the second component to each dart's cost, instead only using $2g + 1$ integers based on the homology signatures and x to perturb each dart's cost. Let $c' : \vec{E} \to \mathbb{R} \times \mathbb{N}^{2g+1}$ be the resulting perturbed costs.

The first modification described above has no effect on our scheme's guarantee for minimum-cost flows and shortest paths. The second modification may have consequences, though. Namely, our scheme may have introduced negative cost directed cycles where the original cost function had zero cost cycles, and minimum-cost flows and "shortest paths" may not be unique if a dart and its reversal both have zero cost. From this point forward, we will work under the assumption that the every directed cycle has strictly positive cost according to the unperturbed cost function c.[5]

Now, let us return to the planar setting for the rest of this section (so $g = 0$). We claim the pivots chosen using the leafmost rule with dart costs c are actually the same as the unique pivot choices guaranteed using perturbed costs c'. We state this claim formally in the following lemma.

Lemma F.1. *Consider an iteration of the multiple-source shortest path algorithm in planar graphs where the source of T moves from vertex u to vertex v. For each value of λ, the leafmost active dart of minimum slack according to c is also the unique active dart of minimum slack according to c'.*

Proof: We may assume the lemma holds inductively across earlier iterations of the algorithm and earlier values of λ within the current iteration, meaning the current holiest path tree T is the same for the algorithm using the leafmost rule with c and the algorithm using perturbed costs c'. Recall, the drainage x used to define c' has sink r. Lemma C.2 essentially states that the exact choice of x does not matter; all drainages with equivalent dual imbalances result in the same unique minimum-cost flows and shortest paths in G. Therefore, we may assume without loss of generality that x is non-zero only within the complementary dual spanning tree C of T.

Now, suppose there are two active darts $x \to y$ and $x' \to y'$ of minimum slack according to c. No darts along $\sigma(v, x)$, $\sigma(v, x')$, $\sigma(u, y)$, or $\sigma(u, y')$ have a non-zero face part to their perturbed costs. Therefore, the face parts of $slack_\lambda(x \to y)$ and $slack_\lambda(x' \to y')$ are equal to the face parts of $c'(x \to y) - \lambda$ and $c'(x' \to y') - \lambda$, respectively. That perturbation term is lower for whichever dart of $x \to y$ and $x' \to y'$ lies closer to the leaf of C. □

The intuition provided by Lemma F.1 will prove crucial in the next section where we describe our linear-time multiple-source shortest path algorithm for unit dart costs in embedded graphs of constant genus. While there does not appear to be an intuitive definition of leafmost in higher genus embedded graphs, we *do* have an equivalent perturbation scheme already defined for that setting. Our goal will be to efficiently find the unique pivots guaranteed by our use of the scheme.

Remark We briefly remark that our perturbation scheme also explains the use of the leafmost rule in efficient s, t-maximum flow algorithms based on parametric shortest paths in the dual graph [4, 22, 24]. For this problem, the drainage is actually computed in the *primal* graph using t as the sink.

[5] Some form of this assumption appears necessary to guarantee correctness of Eisenstat and Klein's [22] planar graph multiple-source shortest path algorithm when it is asked to work with small non-negative integer dart costs. The algorithm appears to handle zero-cost darts gracefully in most cases, but if our assumption is not meant, then it may add an entire directed cycle to the shortest path tree, disconnecting the tree during the last pivot that adds a dart to that cycle. Intuitively, a leftmost shortest path should go around a zero-cost clockwise cycle an infinite number of times.

G Linear-time Multiple-Source Shortest Paths for Small Integer Costs

Let $G = (V, E, F)$ be an embedded graph of size n and constant genus g and let $r \in F$ be a face of G. Let $c : \vec{E} \to \mathbb{R}$ be a dart cost function where each $c(d)$ is a *small non-negative integer*. Let L be the sum of the dart costs. We now describe an algorithm for computing multiple-source shortest paths in this setting that runs in $O(g(gn \log g + L))$ time. Like Eisenstat and Klein [22], we primarily focus on computing an initial shortest path tree and then performing the pivots needed to move the source of the tree around r.

We will address computing shortest path distances later.

Let $c' : \vec{E} \to \mathbb{R} \times \mathbb{N}^{2g+1}$ be the perturbed costs computed using the *modified* scheme presented in Section F.1. Namely, c' is computed using a drainage x with sink r, and its $2g+1$ perturbation terms are based only on homology signatures and x. Let T be the shortest path tree maintained while running Section E's multiple-source shortest path tree algorithm along face $r \in F$. Suppose we are moving the source of T from vertex u to vertex v, and let $v{\to}u = r{\uparrow}q$. Let c'_λ be the cost function parameterized by λ as described in Section E.

For planar graphs, Eisenstat and Klein [22] explicitly maintain the slacks of the darts based on the original cost function c. In other words, they maintain the first component the slacks according to c'_λ. We will refer to these values as the ***original slacks*** and slacks defined using every component of c'_λ as the ***perturbed slacks***. Recall, the edges outside of T form a spanning tree C in G^* if G is planar. To find pivots, Eisenstat and Klein walk a pointer up the directed dual path from q to r in C. When they find a dart $x{\to}y$ with 0 original slack, they perform a pivot by adding $x{\to}y$ to T and removing the old predecessor dart $w{\to}y$ from T. According to Lemma F.1, this pivot is exactly the pivot *required* by using the perturbed slacks. After performing the pivot, Eisenstat and Klein reset their pointer to continue the walk from the first dart that only appears in the new q to r path in C. If their walk reaches r, then every dart along the current q to r path has positive original slack. They decrement their stored slack values for every dart in the path, increment the slacks for those darts' reversals, and start a new walk from q.

As described above, their algorithm does not appear to generalize cleanly to higher genus surfaces. The dual complement to T is no longer a spanning tree, so it is not clear what route a pointer should take. In particular, it is completely unclear what the leafmost dart of 0 original slack should be, especially when the set of active darts may not even be a connected subgraph of G^*. While leafmost may not cleanly generalize, however, our perturbation scheme is already defined for higher genus embeddings.

We now present some useful observations, slightly modifying conventions and terminology from Erickson and Har-Peled [27] and Cabello *et al.* [11]. Let X be the set of edges complementary to T. We refer to X as a ***cut graph***; removing the dual embedding of X cuts the underlying surface Σ into a disk. Let $\bar{X}$ be the ***2-core*** of X obtained by repeatedly removing vertices of degree 1 except for q and r until no others remain. We refer to the dual forest of removed edges as the ***hair*** H of X. The 2-core $\bar{X}$ consists of up to $6g+1$ dual paths $\pi_1, \pi_2, \ldots$ that meet at up to $4g+2$ dual vertices (see Erickson and Har-Peled [27, Lemma 4.2]). Each of these $4g+2$ dual vertices except possibly q and r has degree at least 3. We refer to the (unoriented) dual paths $\pi_1, \pi_2, \ldots$ as ***cut paths***. The endpoints of the cut paths are called ***branching vertices***. We let π_q denote the (possibly trivial) maximal subpath of X with one endpoint on q that contains at most one vertex that is either degree 3 or equal to r. The 2-core $\bar{X}$ of X is useful to us, because the set of edges containing active darts is precisely the set of edges in a subset of cut paths [11, Section 4.2].

An edge $e \notin T$ is in $\bar{X}$ if and only if it forms a non-contractible fundamental cycle with T with respect to $\Sigma - r$ or it forms a contractible fundamental cycle with T and lies on π_q (see Cabello *et al.* [11, Section 4.1]). We have the following lemmas.

Lemma G.1. *Let $d = x{\to}y$ be an active dart, and let h be the homology signature of d's fundamental cycle with T. Signature h is equal to the homology part of $slack_\lambda(d) + \lambda - c'(v{\to}u)$.*

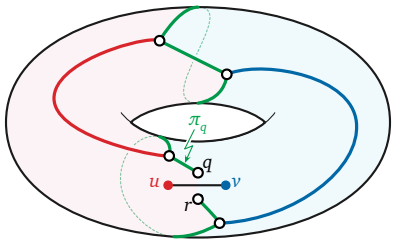

Figure 2. The 2-core $\bar{X}$ of a cut graph X on the torus. Cut paths between red vertices are red; cut paths between blue vertices are blue; and cut paths containing active darts are green.

Lemma G.2. *The active dart with minimum perturbed slack is the active dart with minimum unperturbed slack whose directed fundamental cycle with T has lexicographically least homology signature.*

Lemma G.3. *Fix an orientation of some cut path π_i, and let d_1 and d_2 be darts of that oriented cut path. Let γ_1 and γ_2 be directed fundamental cycles of d_1 and d_2 respectively with T. We have $[\gamma_1] = [\gamma_2]$.*

Proof: Let (T, L, C) be a tree-cotree decomposition of G where X contains all edges of C and L. We can define a set of homology signatures $[\cdot]'$ using (T, L, C). For any set of dual cycles $\gamma_1^*, \gamma_2^*, \ldots$ in X and edge e in X, the set of cycles passing through e is determined by which cut path e belongs to. Therefore $[d_1]' = [d_2]'$. Because $[d]' = 0$ for every dart in T and its reversal, we have $[\gamma_1]' = [\gamma_2]'$. Lemma B.1 implies γ_1 and γ_2 are homologous and therefore $[\gamma_1] = [\gamma_2]$. □

Lemma G.4. *Fix a homology signature h. Let $\Pi = \{\pi_{i_1}, \pi_{i_2}, \ldots\}$ be the set of cut paths containing active darts whose perturbed slacks have homology part h, and assume each cut path is oriented so its darts have blue tails and red heads in G. Finally, let $D = \langle d_1, d_2, \ldots \rangle$ be the sequence of darts within these oriented cut paths in increasing order of perturbed slack. The darts within any one oriented cut path $\pi \in \Pi$ appear as a consequence subsequence of D.*

Proof: We assume Π contains at least one cut path π; otherwise, the proof is trivial. As in the proof of Lemma G.3, let $[\cdot]'$ be a set of homology signatures defined using (T, L, C). Let h' be the homology part of any dart in π. As in the proof of Lemma F.1, we may assume (within the confines of the current proof) that the drainage x used to define c' is non-zero only within edges of X. In particular, the face part of each dart d's perturbed slack is equal to the face part of $c'(d)$. Every dart of every oriented cut path in Π has the same homology part in their slack, so these face parts entirely determine the order of the darts in sequence D. We have two cases to consider.

Suppose $h' = 0$. Let d be any dart in D. By Lemma B.1, the oriented fundamental cycle γ of d in T is the boundary of some potential function α. Assume without loss of generality that $\alpha(r) = 0$. Cycle γ is simple and goes counterclockwise around a set of faces *excluding* r, so $\alpha(p) \in \{-1, 0\}$ for every face $p \in F$. Because γ includes dart $q\uparrow r$, we have $\alpha(q) = -1$. Therefore, d lies on a simple q to r path through X. Because $h' = 0$ and X is a spanning tree plus $2g$ edges, π *does not* lie on a simple dual cycle of X. The drainage x defining c' must assign increasing face perturbations along the darts on the q, r-path. Therefore, darts have increasing face perturbations as you follow π, and the darts in any other cut path $\pi' \in \Pi$ all have strictly smaller or strictly larger face perturbations than those in π.

Now, suppose $h' \neq 0$. The signature $[d]'$ of a dart d in π is determined entirely by the set of dual fundamental cycles of L with C that d belongs to. Since $h' \neq 0$, there exists at least one oriented dual fundamental cycle containing every dart of every cut path of Π. Let γ be one such dual cycle. Let $p \in F$ be the least common ancestor of all edges in C carrying γ. The dual darts with positive value in drainage x are all oriented toward p. Therefore, as we walk along γ starting at p, we encounter darts of negative but increasing face part in their costs, a dart of 0 face part in its cost (from L), and then darts of increasing

positive face part in their costs. Once again, darts have increasing face parts as you follow π, and the darts in any other cut path $\pi' \in \Pi$ all have strictly smaller or strictly larger face parts than those in π. □

Lemma G.5. *Suppose λ becomes large enough that its unperturbed and homology parts become equal to their counterparts in $c'(v \to u)$. Then, there are no more pivots to perform in the current iteration.*

Proof: Consider any active dart d. The homology and face parts of $slack_\lambda(d)$ are equal to the their counterparts in the cost of $\gamma = \sigma(v,x) \circ x \to y \circ rev(\sigma(u,y)) \circ u \to v$ according to c'_λ. If the homology part of $slack_\lambda(d)$ is positive, then d will never be tense in the current iteration. Therefore, the homology part of $c'_\lambda(\gamma)$ must be zero for d to become tense in the current iteration. However, the homology part of $c'_\lambda(\gamma)$ is equal to its counterpart in $c'(\gamma)$. By Lemmas C.1 and B.1, γ is a boundary circulation. Because $q \uparrow r$ is in γ, the dual darts of γ must enter a strict subset of faces $F' \subseteq F$ containing r. Lemma C.2 implies $c'(\gamma)$ has strictly positive face part. In particular, λ cannot rise high enough to make d tense before it reaches $c'(v \to u)$. □

Based on the previous observations, we use the following strategy to compute multiple-source shortest paths and analyze our algorithm's running time. In the zeroth subiteration immediately after the special pivot, we attempt to raise λ so its homology part is equal to the homology part of $c'(v \to u)$; Lemma G.1 implies we cannot do so without pivoting darts of 0 unperturbed slack and negative fundamental homology signature. By Lemmas G.3 and G.4, we may guide our search by ordering a subset of the cut paths and searching along them one-by-one for darts to pivot.

In subsequent subiterations, we attempt to increase λ by exactly 1. At the beginning of each such subiteration, the fundamental cycle homology signature of each dart is equal to the homology part of its perturbed slack. In particular, darts with negative signature have strictly positive unperturbed slack. Therefore, we search for pivots beginning with active darts with non-negative fundamental cycle homology signatures before continuing the search on negative signature darts that may prevent us from increasing λ by 1. Again, we order some cut paths and then search along them one-by-one. However, we must treat π_q with care as described below. By Lemma G.5, we can end the main iteration once a subiteration ends with λ's unperturbed part equal to $c(v \to u)$.

Let $x \to y = o \uparrow p$ be a dart, and let G be embedded in surface Σ. To analyze the running time, we define a concept called *restricted homotopy* relative to $x \to y$. Two shortest paths to x are restricted homotopic if they are homotopic in $\Sigma - \{o, r\}$. Because holiest paths to x do not cross, the restricted homotopy class of x changes $O(g)$ times (see Cabello *et al.* [11, Lemma 4.3]). Essentially, every time our algorithm needs to touch an edge e we can charge to a change in restrited homotopy class for $\vec{e}$ or $rev(\vec{e})$.

If a dart d^+ pivots into T, and d^+ lies on some cut path *other than* π_q, then we can charge to restricted homotopy changes for *every* edge with a dart becoming active or becoming inactive during the pivot; indeed we can even charge to changes in unrestricted homotopy in $\Sigma - \{r\}$. If d^+ lies on π_q, we have to be more careful. Suppose dart d^- leaves T during the pivot. We use a strategy similar to Eisenstat and Klein [22] of only touching darts that lie on the path from q to d^+ in π_q before the pivot or lie on the path from q to d^- *after* the pivot. We *must* charge to restricted homotopy changes in this case, and we note that doing so is a reinterpretation of Eisenstat and Klein's observation that darts on the difference of the two π_qs switch between being 'right-to-left' or 'left-to-right' relative to T.

The charging scheme described above accounts for the time it takes to walk along cut paths searching for darts to pivot into T, the time spent touching darts and modifying cut paths during these pivots, and the time spent explicitly updating the unperturbed slack of every dart in the graph when we reach the end of a subiteration. All of these operations take $O(g(n+L))$ time total. The additional $O(g^2 n \log g)$ in the running time comes from maintaining information about the cut paths including the order we should search them for pivots.

G.1 The algorithm

We begin by describing the data structures used by our algorithm. These data structures extend the ones used by Eisenstat and Klein [22] to work with more general embedded graphs. As a general rule, we explicitly stores lots of information about cut paths other than π_q. Handling π_q requires more care as we do not have time to explicitly maintain it or even remember both of its endpoints at certain moments in the algorithm.

- For each vertex $v \in V$, we store its predecessor dart $pred(v)$ in the shortest path tree T.
- For each face $p \in F$ we store its successor dart $succ(p)$ in an arbitrary spanning tree of X rooted at r as well as a boolean $visited(p)$. These values will aid us in maintaining an implicit list of darts along π_q.
- We maintain the unperturbed part of parameter λ as $uparameter$.
- We maintain a ***reduced cut graph*** $\tilde{X} = (\tilde{V}, \tilde{E})$, an embedded graph of genus g with a vertex for every vertex of the 2-core $\bar{X}$ and an edge $\tilde{e}_i$ for every cut path π_i. We refer to vertices and edges of $\tilde{X}$ as cut vertices and cut edges, respectively. As in all embedded graphs, each cut edge $\tilde{e}_i$ has two ***cut darts*** representing the two orientations of π_i.
- For each dart $d \in \vec{E}$ we store its *unperturbed* slack $uslack(d)$ as an integer. If d's edge lies on a cut path other than π_q, we also store $cutdart(d)$, the cut dart for d's oriented cut path; otherwise, we set $cutdart(d)$ to NULL.
- For every cut dart $\tilde{d}$, we maintain the value $cyclesig(\tilde{d})$ equal to the fundamental cycle homology signatures of the darts in its oriented cut path. If $\tilde{d}$'s cut path is not π_q, then we store its darts in a list $darts(\tilde{d})$.
- We maintain a balanced binary tree $\mathcal{A}$ containing ordered lists of cut darts. The cut darts within each list all have the same $cyclesig$ value, and cut darts contain a pointer $treenode$ back to their tree node. The lists for negative $cyclesig$s *always* come after lists for non-negative $cyclesig$s. Otherwise, lists are ordered by $cyclesig(\cdot)$ lexicographically.
- Finally, we maintain one (pointer to a) cut dart $\tilde{d}^*$ and face $p^* \in F$ as our ***fingers***. We use both fingers to track progress as we investigate slacks of active darts, looking for the next one to pivot into T. In a sense, the fingers act as proxies for the homology and face components of parameter λ. In particular, the homology part of $c'(v \to u) - \lambda = cyclesig(\tilde{d}^*)$ (although the face part might be negative). Lemma C.1 implies the homology part of the dart slacks in $\tilde{d}^*$'s cut path are all zero. If $\tilde{d}^*$ represents an orientation of π_q, then we also have that p^* lies on $\tilde{d}^*$'s cut path π^*, and darts prior to p^* along π^* have a negative face part to their slack, meaning the unperturbed part of their slack is strictly positive.

The algorithm begins by computing the initial holiest tree T rooted at some vertex u on r. We cannot simply apply the linear time algorithm of Henzinger *et al.* [38], because the perturbed cost of some darts may be negative. Therefore, we begin by computing *some* shortest path tree rooted at r using the original costs c in $O(n)$ time. Let $H \subseteq \vec{E}$ be the subset of darts with 0 original slack. The holiest tree uses only darts of H. Further, our assumption that G contains no zero-cost cycles guarantees H is acyclic. We compute the holiest tree T from u in $O(gn)$ time using the standard shortest path tree algorithm for directed acyclic graphs on H. Surprisingly, the initial computation of T is the *only* time our algorithm will explicitly refer to the face components of the darts' perturbed costs.

The rest of the data structures can be filled in $O(gn)$ time. There are no active darts, so the binary search tree $\mathcal{A}$ is initially empty. Similarly, the fingers $\tilde{d}^*$ and p^* are initially set to NULL. All that remains is to describe how to find the next dart to pivot into T, and how to perform the pivot efficiently.

Finding a pivot Suppose we are moving the source of T from vertex u to vertex v, and let $v{\to}u = r{\uparrow}q$. If $\tilde{d}^*$ is unset, then that represents us being at the beginning of some subiteration other than the zeroth one. We see if p^* is set, and if not, set it to q. Then we have p^* (continue to) follow the path given by $succ(\cdot)$ and set $visited(p)$ for each dual vertex p we visit. Suppose we encounter a dart d where $uslack(d) = 0$ during our traversal. Dart d must be the active dart of least perturbed slack, because we have already confirmed that the unperturbed slack is greater than zero for all active darts whose perturbed slack has lessor homology and face parts. We pivot d into T using the procedure described below.

We continue our search until p^* reaches some dual vertex p_q lying on a cut path other than π_q. This vertex may be r. We have just discovered the new location where π_q reaches a branching vertex or r, and we may need to repair the reduced cut graph and the dart lists for the cut edges based on our newfound knowledge.

If we just finished the first walk along π_q since the last special pivot, we move the endpoint of the cut edge $\tilde{e}^*$ for cut path π^* according to where we found q_p. If we just created a degree 2 vertex in the reduced cut graph $\tilde{X}$, then we merge the two cut edges and their cut darts' dart lists. If we just subdivided a cut edge of $\tilde{X}$, then we split the old cut edge's cut dart's dart lists at p_q. We then take a walk in the *reduced cut graph* clockwise around the cut darts whose paths surround the blue side of T, beginning our walk at the cut vertex representing p_q and ending at the cut vertex representing r. For each cut edge $\tilde{e}$ we encounter *once* in order we encounter them, we add its cut dart oriented in the direction of the walk to the appropriate list of$\mathcal{A}$ so we know where to begin subsequent searches for pivots in the current subiteration.

However, if we've already done such a walk since the last special pivot, we can save a bit of time. We *add* an additional cut edge from the cut vertex representing q to the appropriate location based on q_p as the new π_q. Call each vertex that turned from red to blue since we began exploring π_q ***purple***. We walk clockwise around the cut darts of $\tilde{X}$ whose cut paths surround the purple vertices. The walk includes ***old*** cut darts whose reversals' oriented cut paths contain darts about to become inactive, cut darts and their reversals whose cut paths contain darts that will never be active in the current iteration, and ***new*** cut darts whose oriented cut paths will contain new active darts [11, Section 4.2]. We remove old cut darts from their lists in $\mathcal{A}$, deleting nodes whose lists become empty. We then add every new cut dart to a list in $\mathcal{A}$, adding new lists as necessary. Next, we remove the old cut edge representing π_q. We then perform the walk described above to figure out the correct order of cut darts within $\mathcal{A}$'s lists. Doing this walk *after* figuring out the nodes of $\mathcal{A}$ means we can use the $treenode$ values to fix the lists in only $O(g)$ additional time.

We then set p^* to NULL, and set $\tilde{d}^*$ to the first cut dart of the first list in $\mathcal{A}$. If we have not performed any pivots since the last time we did this walk, we may skip the above steps involving the reduced cut graph and just use the unmodified $\mathcal{A}$ from the previous subiteration.

Now, suppose $\tilde{d}^*$ is set. We check its darts in order along $darts(\tilde{d}^*)$. In the zeroth subiteration, we only search darts d where $cyclesig(d)$ is negative; we will guarantee $\tilde{d}^*$'s real darts have this property. We look for the first dart d with $uslack(d) = 0$ as it is blocking us from increasing the homology part of λ to the homology part of $c'(v{\to}u)$. In subsequent subiterations, we want to increase λ by a whole integer. We look for the first dart d with $uslack(d) = 0$ if $cyclesig(d)$ is non-negative or $uslack(d) = 1$ otherwise. If we find a dart d in either situation described above, we pivot it into T using the procedure described below. If not, we try to move $\tilde{d}^*$ to the next position in its current list $\mathcal{A}$. If that list is empty,

we try to move $\tilde{d}^*$ to the first position in the next list.

If we cannot move $\tilde{d}^*$ or initially set it because there are no more positions in $\mathcal{A}$, then we have reached the end of a subiteration. If we are finishing any subiteration other than the zeroth one, then we have successfully completed all the pivots necessary to increase λ by 1. We increase $uparameter$ by 1, and for each active dart d, we decrement $uslack(d)$ and increment $uslack(rev(d))$. We also unset $visited(p)$ for every dual vertex along π_q. After *any* subiteration, we then do the following. We check if $uparameter = c(v \to u)$. If so, then we have finished the full iteration moving the source of T to vertex v. Otherwise, we set $p^* := q$ and begin searching π_q as described above.

Performing a pivot Suppose the algorithm decides to pivot dart $d^+ = x^+ \to y = o^+ \uparrow p^+$ into the holiest tree T while removing dart $d^- = x^- \to y = o^- \uparrow p^-$. The steps for performing a regular or special pivot are largely the same.

First, suppose d^+ lies on π_q. We need to reroute π_q through d^-'s edge. We add d^-'s edge to X by setting $succ(p^-) := rev(d^-)$. Then, we walk back through the hair of X from p^- following $succ$ darts until we encounter o^+. The search must end there before reaching a cut path other than π_q, because otherwise there will be no route from o^+ to the other cut paths or r in our newly repaired cut graph. Let P_{p^-} be the dual path we just followed. As we walk along P_{p^-}, we reverse its darts so they point toward $rev(d^-)$ by resetting successors of P_{p^-}'s dual vertices.

Suppose we are performing a regular pivot. If we encounter any dual vertices p with $visited(p)$ set, then we place p^* at the first one encountered and unset $visited(p)$ for every dual vertex we encounter. We do these steps to represent the fact that p^* now lies at the dual tail of the first active dart in the modified cut path we are creating and the previously visited faces no longer lie on that cut path. If we are performing the special pivot, we will not encounter any visited dual vertices and we leave p^* set to NULL. Finally, we add d^+ to T by setting $pred(y) = d^+$.

Now, suppose d^+ lies on any cut path other than π_q. The fundamental cycle of d^+ in T is non-contractible in the surface $\Sigma - r$. Therefore, d^- has to belong on a new cut path other than π_q. We walk back through the hair of X from p^- following $succ$ darts until we encounter a dual vertex p_1 such that either p_1 is on some cut path other than π_q or $visited(p_1)$ is set. The latter case represents us running into π_q during our search. We *also* walk forward through the hair from o^- following $succ$ darts until we encounter a dual vertex p_2 such that p_2 is on some cut path. Note that this second walk cannot encounter π_q. Otherwise, the first walk encounters it as well and we create a contractible cycle in the dual after adding d^-'s edge to X or the first walk encounters another cut path, meaning there is a walk through the cut graph to π_q from that location and π_q ends at an earlier vertex than we are currently assuming.

After finding p_1 and p_2, we modify the reduced cut graph $\tilde{X}$ according to the new cut path we found. Let $\tilde{e}$ be the new cut edge for this cut path. We add $\tilde{e}$ to $\tilde{X}$, possibly subdividing existing cut edges according to where we found p_1 and p_2. When we subdivide cut edges, we split their old cut edge's cut dart's dart lists. We create the new dart lists for both cut darts of $\tilde{e}$ by just including every dart we encountered during our walks and their reversals.

Similar to earlier, call each vertex turning from red to blue during the current pivot ***purple***. We walk clockwise around the cut darts of $\tilde{X}$ whose cut paths surround the purple vertices. The $cyclesigs$ for the cut darts encountered on our walk may be inaccurate due to the change in T. In particular, if dart d has a purple tail, then its fundamental cycle homology signature changes precisely by the fundamental cycle homology signature of d^+. We add the fundamental cycle homology signature of d^+ to $cyclesig(\tilde{d})$ for each cut dart $\tilde{d}$ encountered on our walk and subtract it for each of these cut darts' reversals. As before, we remove old cut darts from their lists in $\mathcal{A}$, deleting nodes whose lists become empty. We then add every new cut dart to a list in $\mathcal{A}$, adding new lists as necessary. The lists in $\mathcal{A}$ may be in the incorrect order after doing this pass, but we will fix their order shortly.

We add d^+ to T by setting $pred(y) := d^+$ and update $\tilde{X}$ by removing the cut path containing d^+ and all the darts of its representative cut darts. Finally, we walk clockwise round the cut darts whose cut paths surround all (new and old) blue vertices, using cut darts' $treenode$ pointers to quickly restore the order of cut darts within each nodes' list.

Whether or not d^+ lies on π_q, we have one more step to perform if we just did the special pivot to move the source from u to v. The unperturbed distance from u to v is $uslack(v{\to}u) - c(v{\to}u)$, so we set $uparamter := c(v{\to}u) - uslack(v{\to}u)$ and set $uslack(v{\to}u) := 0$.

G.2 Running time

Having described how our algorithm is implemented, we turn to bounding its running time. From prior work, we already know there are at most $O(gn)$ pivots [11]. However, we still need a bound on the time spent interacting with individual darts in X that do not get pivoted into T.

Fix a dart $x{\to}y = o{\uparrow}p$. Let $\langle u_1, u_2, \ldots, u_k \rangle$ be the sequence of sources for the shortest path tree T in order around r. We say two paths P_1 and P_2 from a vertex on r to x are ***restricted homotopic*** with respect to $x{\to}y$ if there is a homotopy from P_1 to P_2 in $\Sigma_o = \Sigma - \{o, r\}$ where the first endpoint of the path may move anywhere along the embedding of r's boundary except for the embedding of dart $u_k{\to}u_1$. We have the following lemma.

Lemma G.6. *The shortest path to x in T changes restricted homotopy class with respect to $x{\to}y$ $O(g)$ times.*

Proof: Let $A = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ be the sequence of shortest paths to x. Suppose σ_i and σ_j are restricted homotopic and $j > i$. Let $r[u_i, u_j]$ denote the subpath around r from u_i to u_j. We see $r[u_i, u_j] \circ \sigma_j \circ rev(\sigma_i)$ forms a disk in Σ_o. Because the shortest path from each vertex u_i to x is unique, the shortest paths in this set are pairwise non-crossing. In particular, no shortest path between σ_i and σ_j can lie in another restricted homotopy class, and each restricted homotopy class is represented by a contiguous subsequence of A. Let A' be the subsequence containing the first example of each homotopy class. Extend the beginning of each path in A' backward along r to u_1; these extended paths still do not cross. The maximum number of pairwise non-crossing, non-homotopic paths in Σ_o is $O(g)$ (see Chambers *et al.* [12, Lemma 2.1]), and this same bound applies to the size of A'. □

The general strategy for bounding the running time of our algorithm is to charge various interactions with darts to changes in the restricted homotopy of their endpoints. We begin by discussing pivots. Suppose the algorithm decides to pivot dart $d^+ = x^+{\to}y = o^+{\uparrow}p^+$ into the holiest tree T while removing dart $d^- = x^-{\to}y = o^-{\uparrow}p^-$.

If d^+ lies on π_q, then our algorithm interacts with several darts in the dual graph between o^+ and p^- along with their reversals. However, the fundamental cycle through d^+ before the pivot encloses every face along this walk. For each dart $x{\to}y = o{\uparrow}p$ along the walk from o^+ to p^-, we see the restricted homotopy class of x relative to $x{\to}y$ changes, because the difference between the old shortest path to x and the new shortest path to x encloses o but not r.

If d^+ lies on any other cut path, then the algorithm may interact with every dart $x{\to}y$ where x is purple and y is not and their reversals. The fundamental cycle through d^+ with T is non-contractible in the surface $\Sigma - r$. Therefore, for each of these darts $x{\to}y$, the restricted homotopy class of x relative to $x{\to}y$ changes. We conclude that the algorithm spends $O(gn)$ time total interacting with individual darts during pivots.

When searching for pivots, the algorithm again interacts with several darts. If the algorithm is in the zeroth iteration, then these darts all lie on cut paths other than π_q. We may charge to the interactions from the *earlier* pivot in the same iteration that made those darts active, because that earlier pivot must

have added a dart d^+ to T that lied on a cut path other than π_q. Note that this earlier pivot may have been the special pivot. In subsequent subiterations, we charge dart interactions to the *future* pivot within that subiteration that makes the dart inactive or to the moment when the algorithm decreases that dart's unperturbed slack. Observe that we can only decrease dart $x \to y$'s slack at most $c(x \to y) + c(y \to x)$ times before we start decreasing the slack of its reversal $y \to x$. However $y \to x$ cannot become active unless it lies on a cut path other than π_q and we interact with it while performing a pivot with darts off π_q as described above or we change $succ(y)$ to be $y \to x$ so $y \to x$ can join π_q. Both interactions mean changing restricted homology classes as described above. We conclude the algorithm spends $O(gL)$ time total interacting with individual darts while searching for pivots and changing unperturbed slack values.

Finally, we must account for the time spent interacting with the reduced cut graph and binary search tree $\mathcal{A}$. We encounter $O(gn)$ cut darts across the whole algorithm as we perform walks in the reduced cut graph around cut darts whose cut paths surround purple vertices [11, Section 4.2]. Performing insertions or deletion in $\mathcal{A}$ takes $O(g \log g)$ time, because comparing homology signatures takes $O(g)$ time, so we spend $O(g^2 n \log g)$ time total doing the insertions and deletions based on these walks. The algorithm sometimes walks along cut darts whose cut paths surround all blue vertices. Fortunately, there's only one walk of this kind modifying the nodes of $\mathcal{A}$ per each of the $O(n)$ main iterations of the algorithm. The other walks of this kind take only constant time per cut dart, and they occur at most once per pivot. Each of these walks may encounter $O(g)$ cut darts, meaning these walks take $O(g^2 n \log g)$ time total. The overall running time of our algorithm is $O(g(gn \log g + L))$.

Theorem G.7. *Let $G = (V, E, F)$ be a graph of size n and genus g, let $c : \vec{E} \to \mathbb{N}^+$ be a non-negative, integral dart cost function with dart costs summing to L, and let $r \in F$ be any face of G. We can compute every dart entering and leaving the shortest path trees from each vertex incident to r in order in $O(g(gn \log g + L))$ time.*

G.3 Computing distances

The above algorithm successfully computes the pivots for multiple-source shortest paths around r in order. However, most applications of multiple-source shortest paths are actually concerned with at least a subset of the shortest path distances. Fortunately, this subset of distances is usually structured in a convenient way.

Let $A = \langle u_1, u_2, \ldots, u_{k_1} \rangle$ be the sequence of vertices around r, and let $B = \langle v_1, v_2, \ldots, v_{k_2} \rangle$ be the sequence of vertices in an arbitrary walk through G. A ***monotone correspondence*** $\mathcal{C}$ between A and B is a set of pairs (u_i, v_j) where for each $(u_i, v_j), (u_{i'}, v_{j'}) \in \mathcal{C}$ with $i' \geq i$, we have $j' \geq j$. We may easily modify our linear-time algorithm to compute the unperturbed distance from u_i to v_j for every pair (u_i, v_j) appearing in a monotone correspondence $\mathcal{C}$ with only an $O(k_2)$ *additive* increase in the running time. This observation is a generalization of one by Eisenstat and Klein [22, Theorem 4.3] for planar graphs.

We store a variable $dist$ that is initially the unperturbed distance from u_1 to v_1. The initial value for $dist$ can be computed in $O(n)$ time after computing the initial holiest tree T. As the algorithm runs, we will update $dist$ with the shortest path distance between some u_i and v_j. Suppose an iteration of the algorithm has just ended and we are storing the u_i to v_j distance. We may compute the unperturbed distance from u_i to v_{j+1} as as $dist + c(v_j \to v_{j+1}) - uslack(v_j \to v_{j+1})$ and reassign $dist$ to that value. We repeat this step until we have computed distances for every pair containing u_i.

Now, suppose we have just performed the special pivot to move the source of T from u_i to u_{i+1}. After the special pivot, the distance to every vertex in G from the source of T has decreased by the distance from u_i to u_{i+1}. We decrease $dist$ by that amount. Now, the unperturbed distance from u_{i+1} to some vertex v_j increases by 1 at the end of each subiteration (other than the zeroth one) where v_j is red. To easily track if v_j is red, we maintain marks on edges appearing an odd number of times along an

arbitrary walk from u_{i+1} to v_j. If there are an odd number of marked edges containing active darts when we increase $uparameter$, then the unperturbed distance to v_j increases by 1 and we increment $dist$. Otherwise, $dist$ remains unchanged. To maintain these marks for any pair (u_i, v_j) we compute an arbitrary (u_1, v_1) walk at the beginning of the algorithm. Every time we consider distances to the next vertex along B, we flip the mark on the next edge used in B's walk. Every time we perform a special pivot, we flip the mark on the edge for $u_i \rightarrow u_{i+1}$.

Theorem G.8. *Let $G = (V, E, F)$ be a graph of size n and genus g, let $c : \vec{E} \to \mathbb{N}^+$ be a non-negative, integral dart cost function with dart costs summing to L. Let $r \in F$ be any face of G incident to vertices $A = \langle u_1, u_2, \ldots, u_{k_1} \rangle$ in order, and let $B = \langle v_1, v_2, \ldots, v_{k_2} \rangle$ be the sequence of vertices along an arbitrary walk in G. Let $\mathcal{C}$ be an arbitrary monotone correspondence between A and B. We can compute the distance from u_i to v_j for every pair $(u_i, v_j) \in \mathcal{C}$ in $O(g(gn \log g + L) + k_2)$ time.*

Applications We may use the algorithm of Theorem G.8 to easily derive deterministic linear-time algorithms for a variety of problems on unweighted undirected embedded graphs of constant genus. We give a few examples here. In every case, one could already achieve an algorithm running in $g^{O(g)}n$ time, at least for unweighted undirected versions of the following problems, by plugging in the linear time minimum cut algorithm of Weihe [64] into some published algorithm for the various problems. However, there are published algorithms for all of these problems that run in near-linear time and can use the multiple-source shortest path algorithm of Cabello *et al.* [11] as a black box result. While Cabello *et al.* require uniqueness of shortest paths, it is otherwise unnecessary in these applications. In every case, the set of shortest path distances required by the algorithm have the form required by Theorem G.8 with $k_2 = O(n)$.

We may compute minimum s, t-cuts in a genus g surface embedded graph using an algorithm of Erickson and Nayyeri [28] that runs multiple-source shortest paths in a derived graph of genus $2^{O(g)}$.

Theorem G.9. *Let $G = (V, E, F)$ be an unweighted undirected graph of size n and genus g, and let $s, t \in V$. There exists a deterministic algorithm that computes a minimum s, t-cut of G in $2^{O(g)}n$ time.*

We may compute global minimum cuts in linear time as well using an algorithm of Erickson *et al.* [26] with some slight modifications. We replace their use of an algorithm by Chambers *et al.* [29] for computing minimum weight homologous subgraphs with the algorithm of Erickson and Nayyeri [28] for the same problem, plugging in our multiple-source shortest path algorithm when needed by Erickson and Nayyeri. Also, we replace their use of an algorithm by Łącki and Sankowski [48] for global minimum cut in planar graphs with a linear time algorithm for the same problem in unweighted planar graphs by Chang and Lu [15].

Theorem G.10. *Let $G = (V, E, F)$ be an unweighted undirected graph of size n and genus g. There exists a deterministic algorithm that computes a global minimum cut of G in $2^{O(g)}n$ time.*

We may compute shortest non-separating and non-contractible cycles by simply substituting our multiple-source shortest path algorithm into the algorithms of Erickson [25] and Fox [33]. Erickson uses $O(g)$ instances of multiple-source shortest paths to compute a shortest non-separating cycle relative to G's embedding. Fox uses $O(g^2 + b)$ instances to compute a shortest non-contractible cycle when the embedding has b boundary components. Note that these algorithms are actually designed for directed graphs with dart costs, but the undirected case follows from the obvious reduction. In this case, there was *no* known linear time algorithm for directed graphs with small integer dart costs, although the aforementioned running time of $g^{O(g)}n$ was achievable in unweighted undirected graphs.

Theorem G.11. *Let $G = (V, E, F)$ be a graph of size n and genus g, embedded in a surface with b boundary components, and let $c : \vec{E} \to \mathbb{N}^+$ be a positive, integral dart cost function with dart costs summing to L. We may compute a shortest non-separating cycle in G in $O(g^2(gn \log g + L))$ time and a shortest non-contractible cycle in G in $O((g^2 + b)g(gn \log g + L))$ time.*

Finally, we may compute a shortest *homology basis* in an unweighted undirected graph by plugging our multiple-source shortest path algorithm into the algorithm of Borradaile *et al.* [2]. Their algorithm uses $O((g + b)^2)$ instances of multiple-source shortest paths in a derived graph of genus $O(g + b)$ when the embedding has b boundary faces.

Theorem G.12. *Let $G = (V, E, F)$ be an unweighted undirected graph of size n and genus g, embedded in a surface with b boundary components. There exists a deterministic algorithm that computes a minimum homology basis of G in $O((g + b)^4 n \log(g + b))$ time.*

Acknowledgements. The authors would like to thank Serigo Cabello, Erin W. Chambers, and Shay Mozes for many helpful discussions.

References

[1] C. Berge and A. Ghouilla-Houri. *Programming, Games, and Transportation Networks*. Methuen & Co, 1965.

[2] G. Borradaile, E. W. Chambers, K. Fox, and A. Nayyeri. Minimum cycle and homology bases of surface-embedded graphs. *J. Comput. Geom.*, 8(2):58–79, 2017.

[3] G. Borradaile, D. Eppstein, A. Nayyeri, and C. Wulff-Nilsen. All-pairs minimum cuts in near-linear time for surface-embedded graphs. In *Proc. 32nd Intern. Symp. Comput. Geom.*, pages 22:1–22:16, 2016.

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

[5] G. Borradaile, P. N. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In *Proc. 52nd IEEE Symp. Found. Comput. Sci.*, pages 170–179, 2011.

[6] G. Borradaile, P. N. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017.

[7] G. Borradaile, P. Sankowski, and C. Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 11(3):16:1–16:29, 2015.

[8] U. Brandes and D. Wagner. A linear time algorithm for the arc disjoint Menger problem in planar directed graphs. *Algorithmica*, 28(1):16–28, 2000.

[9] O. Busaryev, S. Cabello, C. Chen, T. K. Dey, and Y. Wang. Annotating simplices with a homology basis and its applications. In *Proc. 13th Scand. Workshop Algorithm Theory*, pages 189–200, 2012.

[10] S. Cabello. Many distances in planar graphs. *Algorithmica*, 62(1–2):361–381, 2010.

[11] S. Cabello, E. W. Chambers, and J. Erickson. Multiple-source shortest paths in embedded graphs. *SIAM J. Comput.*, 42(4):1542–1571, 2013.

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

[13] E. W. Chambers, J. Erickson, and A. Nayyeri. Homology flows, cohomology cuts. *SIAM J. Comput.*, 41(6):1605–1634, 2012.

[14] E. W. Chambers, K. Fox, and A. Nayyeri. Counting and sampling minimum cuts in genus g graphs. *Discrete Comput. Geom.*, 52(3):450–475, 2014.

[15] H. Chang and H. Lu. Computing the girth of a planar graph in linear time. *SIAM J. Comput.*, 42(3):1077–1094, 2013.

[16] A. Charnes. Optimality and degeneracy in linear programming. *Econometrica*, 20(2):160–170, 1952.

[17] É. Colin de Verdière. Topological algorithms for graphs on surfaces. Habilitation thesis, May 2012.

[18] W. H. Cunningham. A network simplex method. *Math. Program.*, 11:105–116, 1976.

[19] G. P. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality constraints. *Pacific J. Math.*, 5:183–195, 1955.

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

[21] H. Edelsbrunner and J. L. Harer. *Computational Topology: An Introduction*. Amer. Math. Soc., 2010.

[22] D. Eisenstat and P. N. Klein. Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs. In *Proc. 45th Ann. ACM Symp. Theory Comput.*, pages 735–744, 2013.

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

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

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

[26] J. Erickson, K. Fox, and A. Nayyeri. Global minimum cuts in surface embedded graphs. In *Proc. 23rd Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 1309–1318, 2012.

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

[28] J. Erickson and A. Nayyeri. Minimum cuts and shortest non-separating cycles via homology covers. In *Proc. 22nd Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 1166–1176, 2011.

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

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

[31] J. Erickson and P. Worah. Computing the shortest essential cycle. *Discrete Comput. Geom.*, 44(4):912–930, 2010.

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

[33] K. Fox. Shortest non-trivial cycles in directed and undirected surface graphs. In *Proc. 24th Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 352–364, 2013.

[34] D. Hartvigsen and R. Mardon. The all-pairs cut problem and the minimum cycle basis problem on planar graphs. *SIAM J. Discrete Math.*, 7(3):403–418, 1994.

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

[36] A. Hatcher. *Algebraic Topology*. Cambridge Univ. Press, 2002.

[37] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.

[38] 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.

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

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

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

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

[43] K. Kawarabayashi, P. N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Proc. 38th Int. Colloq. Automata Lang. Prog.*, volume 6755 of *Lecture Notes Comput. Sci.*, pages 135–146. Springer-Verlag, 2011.

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

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

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

[47] J. Łącki, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Single source - all sinks max flows in planar digraphs. In *Proc. 53rd IEEE Symp. Found. Comput. Sci.*, pages 599–608, 2012.

[48] J. Łącki and P. Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log\log n)$ time. In C. Demetrescu and M. M. Halldórsson, editors, *Proc. 19th Ann. Europ. Symp. Algorithms*, volume 6942 of *Lecture Notes Comput. Sci.*, pages 155–166. Springer, 2011.

[49] J. Matuschke and B. Peis. Lattices and maximum flow algorithms in planar graphs. In D. M. Thilikos, editor, *Proc. 36th Int. Workshop Graph Theor. Concepts Comput. Sci.*, number 6410 in Lecture Notes Comput. Sci., pages 324–335. Springer-Verlag, 2010.

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

[51] S. Mozes, C. Nikolaev, Y. Nussbaum, and O. Weimann. Minimum cut of directed planar graphs in $O(n \log\log n)$ time. *CoRR*, abs/1512.02068, 2015.

[52] S. Mozes, C. Nikolaev, Y. Nussbaum, and O. Weimann. Minimum cut of directed planar graphs in $O(n \log\log n)$ time. In *Proc. 29th Ann. ACM-SIAM Symp. Disc. Algo.*, 2018.

[53] S. Mozes and C. Sommer. Exact distance oracles for planar graphs. In *Proc. 23rd Ann. ACM-SIAM Symp. Discrete Algorithms*, pages 209–222, 2012.

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

[55] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[56] J. R. Munkres. *Topology*. Prentice-Hall, 2nd edition, 2000.

[57] J. K. Park and C. A. Philips. Finding minimum-quotient cuts in planar graphs. In *Proc. 25th Ann. Symp. Theory Comput.*, pages 766–775, 1993.

[58] V. Patel. Determining edge expansion and other connectivity measures of graphs of bounded genus. *SIAM J. Comput.*, 42(3):1113–1131, 2013.

[59] H. Ripphausen-Lipa, D. Wagner, and K. Weihe. The vertex-disjoint Menger problem in planar graphs. *SIAM J. Comput.*, 26(2):331–349, 1997.

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

[61] R. E. Tarjan. Dynamic trees as search trees via euler tours, applied to the network simplex algorithm. *Math. Program.*, 77:169–177, 1997.

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

[63] S. M. Venkatesan. *Algorithms for network flows*. Ph.D. thesis, The Pennsylvania State University, 1983. Cited in [41].

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

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

[66] C. Wulff-Nilsen. Minimum cycle basis and all-pairs min cut of a planar graph in subquadratic time. Preprint, December 2009.

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

[68] A. Zomorodian. *Topology for Computing*. Cambridge Univ. Press, 2005.