Holiest Minimum-Cost Paths and Flows in Surface Graphs*

Jeff Erickson
Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL, USA
jeffe@illinois.edu

Kyle Fox†
Computer Science
The University of Texas at Dallas
Richardson, TX, USA
kyle.fox@utdallas.edu

Luvsandondov Lkhamsuren‡
Airbnb
San Francisco, CA, USA
lkhamsurenl@gmail.com

ABSTRACT

Let G be an edge-weighted directed graph with n vertices embedded on an orientable 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 paths 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 orientable surfaces. Our algorithm runs in $O(g^2 n \log g)$ time in this setting, and it 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.

CCS CONCEPTS

• **Mathematics of computing → Graphs and surfaces**; **Graph algorithms**; *Network flows*; • **Theory of computation → Shortest paths**;

*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/holiest.pdf. The research presented in this paper was partially supported by NSF grants CCF-1408763, IIS-1408846, IIS-1447554, CCF-1513816, CCF-1546392, CCF-1527084, and CCF-1535972; by ARO grant W911NF-15-1-0408, and by grant 2012/229 from the U.S.-Israel Binational Science Foundation.
†Portions of this work were done while the author was a postdoctoral associate at Duke University.
‡Portions of this work were done while this author was a student at the University of Illinois at Urbana-Champaign.

STOC'18, June 25–29, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5559-9/18/06…$15.00
https://doi.org/10.1145/3188745.3188904

KEYWORDS

computational topology, graphs, surfaces, shortest paths, network flows

ACM Reference Format:
Jeff Erickson, Kyle Fox, and Luvsandondov Lkhamsuren. 2018. Holiest Minimum-Cost Paths and Flows in Surface Graphs. In *Proceedings of 50th Annual ACM SIGACT Symposium on the Theory of Computing (STOC'18).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3188745.3188904

1 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, 5, 22, 23] and global minimum cuts [48] in directed planar (genus 0) graphs as well as the computation of minimum cut oracles in planar and more general embedded graphs [3, 6] (see also Wulff-Nilsen [62]).

This assumption is also used in algorithms for the multiple-source shortest paths problem introduced for planar graphs by Klein [43]. In the multiple-source shortest paths 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, 43]. 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, 24, 27, 29, 31], the computation of maximum flows and minimum cuts [5, 12, 14, 25, 27, 38, 45], the computation of exact and approximate distance oracles [10, 41, 49], and even the computation of *single*-source shortest paths [44, 50].

Enforcing uniqueness. 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.* [51] to argue that shortest paths are unique *with high probability*. This method is used directly by Erickson [23], Mozes *et al.* [48], Cabello *et al.* [11], and the numerous papers that rely on the latter result.

As an alternative to using randomness, one can 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 [15] and Dantzig *et al.* [18], and variants of it have been used for computing minimum cut oracles in planar graphs [6, 32, 62]. 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 a $\log n$ factor increase in the running time of their multiple-source shortest paths 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 [35, 56, 57]; 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, 43] and maximum flows in planar graphs [4, 22, 23] rely (at least by some interpretations) on the parametric shortest paths framework introduced by Karp and Orlin [40, 63]. In short, these algorithms 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 [43], 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 the edge lengths to guarantee unique shortest paths. In particular, Eisenstat and Klein [22] give linear-time maximum flow and multiple-source shortest paths algorithms 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 [60] 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, assuming the initial shortest path tree was itself leftmost. We note the leafmost rule bears a strong resemblance to Cunningham's [17] 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 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 paths algorithm of Cabello *et al.* [11] appear to crucially rely on a guarantee that shortest paths really are unique.

1.1 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 3, 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 $z(e)$ for each edge e so that given a cycle γ bounding a subset of faces $F' \subseteq F$, the absolute value of the sum of these integers along γ is equal to the number of faces in F'. This latter assignment of integers is inspired in part by results of Park and Philips [53] and Patel [54] on the minimum quotient and sparsest cut problems in planar graphs. Our perturbation vectors contain both $[e]$ and $z(e)$, 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.* [48].[1]

Our scheme can also be used in the multiple-source shortest paths 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 5. 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. Based on this observation, we generalize the linear-time multiple-source shortest paths 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 6.

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 can obtain linear time algorithms for each of these problems, assuming the genus is a constant. However, our new algorithms improve the running time for computing cuts from $g^{O(g)}n$ to $2^{O(g)}n$, and they improve the running time for the other problems from $2^{O(g)}n$ to $O(\text{poly}(g)n)$. In particular, ours are the first algorithms for the latter problems that simultaneously have polynomial dependency on g and linear dependency on n. We describe these applications in Section 7.

Because of space constraints, we are unable to provide full details for some of our algorithms and lemma proofs in this version of the paper. We refer the reader to the full version of the paper available at https://utdallas.edu/~kyle.fox/publications/holiest.pdf for these details.

[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.

1.2 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 [30]. Several researchers have designed efficient algorithms for specializations of the maximum flow problem in planar graphs using this idea [1, 4, 33, 37, 55, 60, 61]. There is a deep connection between flows in planar graphs and shortest paths in their duals (see, for example, Venkatesan [59]). As far as we are aware, though, Klein [43] was the first to apply the idea of directly computing _____most shortest path trees.

Khuller, Naor, and Klein [42] observed that the set of integral *circulations* in a planar graph form a distributive 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 flow 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 [46] show that the left/right relation on s, t-paths in planar graphs also forms a lattice.

Bourke, Tewari, and Vinodchandran [7] observed that the *reachability* problem for planar directed graphs lies in *unambiguous log-space* (UL). A key aspect of their algorithm is computing a set of lengths for the edges of a grid graph so that shortest paths are unique. Their edge weighting scheme was later extended to arbitrary planar graphs by Tewari and Vinodchandran [58] and graphs embedded on constant genus surfaces by Datta *et al.* [19]. This latter result is similar to ours in that the length of each edge is the linear combination of $O(g)$ separate length functions including parts that encode the topology of paths and one part encoding face containment for topologically trivial cycles. However, our lexicographic perturbation scheme is arguably easier to implement than Datta *et al.*'s scheme in that they (and Tewari and Vinodchandran [58]) must compute a straight-line embedding of a subgraph of the input, while we work directly with the graph's *combinatorial embedding*. Also, they use about twice as many length functions as we use vector components, and it is unclear if their scheme is as directly useful as ours for designing a linear time algorithm for multiple-source shortest paths in embedded graphs with small integer edge lengths.

2 PRELIMINARIES

We begin with an introduction to surface embedded graphs. For more background we refer the reader to books and surveys [16, 21, 34, 47, 52, 64] related to the topic.

Surfaces. A ***surface*** or 2-manifold with boundary Σ 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 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 embeddings 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 an orbit in the involution rev. We refer to one dart in e's orbit 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 orbit 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 orbit. Vertex v is the ***tail*** of these darts' reversals. Orbits of the permutation $rev \circ \pi$ give the *clockwise* ordering of darts around each face of the

[2] Cabello *et al.* [11] describe a reduction for their multiple-source shortest paths algorithm in graphs embedded in non-orientable surfaces to the same problem in graphs embedded in orientable surfaces. We can 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.

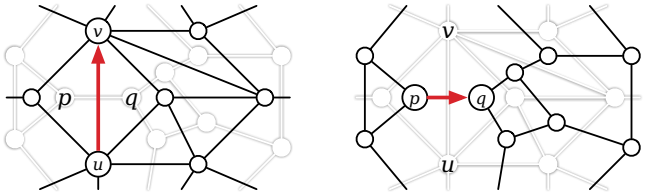

Figure 1. A dart $u \to v$ in the primal graph G and the corresponding dart $p \uparrow q$ in the dual graph G^*.

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. 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 orbits 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. See Figure 1.

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 *antisymmetric* 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] For convenience in our proofs, our formal definitions will require non-negativity only when determining *feasibility* of flows.

A (primal) ***flow*** $f : \vec{E} \to \mathbb{R}$ is an assignment of 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 dart $u \to v = q \uparrow p$, we have $f(u \to v) - f(v \to u) = \alpha(p) - \alpha(q)$. In other words, high potentials to the *right* of darts

[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!

encourage high flow values while high potentials to the left encourage low flow values. 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 any set of flows with identical vertex imbalances that is isomorphic with $\mathbb{R}^{2g}$.

A ***dual flow*** $z : \vec{E} \to \mathbb{R}$ assigns real values to the darts of the *dual* graph G^* such that $z(d) = -z(rev(d))$ for every dart d. Equivalently, we consider a dual flow to be a function on the *edges* of G^* by defining $z(e) = z(\vec{e})$. The ***dual imbalance*** $\partial z : F \to \mathbb{R}$ of dual flow z is the total dual flow going clockwise around each primal face, or equivalently, into each dual vertex. Formally, $\partial z(p) = \sum_{q \uparrow p} z(q \uparrow p)$. 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 $0 \le f(d) \le \mu(d)$ 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 $v \ne s$ 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 $dist_c(s, t)$ denote the distance from s to t according to costs c. Let $\sigma(s, t)$ 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. We may ***root*** T at a vertex s by considering the darts of T oriented *away* from s. Given a root s and vertex $v \ne s$, the ***predecessor*** of v in T is the unique dart $u \to v$ that lies on the path from s to v in T. Given an edge $e \notin T$, the ***fundamental cycle*** of e with T, denoted $cycle(T, e)$ is the unique simple cycle of edges in $T + e$. If $e \in T$, then $cycle(T, e)$ is empty. Given a dart d of an edge e, its fundamental cycle $cycle(T, d)$ with T is the orientation of $cycle(T, e)$ that contains d. A ***spanning cotree*** C of G is a subset of edges that form a spanning tree in the dual graph. We may root C at a dual vertex r by now considering the darts of C oriented *toward* r. The ***successor*** of dual vertex $p \ne r$ is the dart $p \uparrow q$ that lies on the dual path from p to r in C. Dual vertex o is a ***descendant*** of p in C if p is on the dual path from o to r in C.

Given a vector a, let a_i denote the ith component of a. In $O(gn)$ time, we can compute a ***homology signature*** $[e]$ for each edge $e \in E$. 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 [28] and subsequest papers [2, 9, 13, 27]. 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.

Lemma 2.1. *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 $[f_1] = [f_2]$.*

In particular, Lemma 2.1 implies that classes of flows with equivalent homology signatures do not depend upon the particular choice of basis used to define the signatures.

3 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 z where $\partial z(q) < 0$ for all $q \in F \setminus \{r\}$. The definition of a drainage immediately implies $\partial z(r) = -\sum_{q \in F \setminus \{r\}} \partial z(q)$. Because r is the only face with positive dual imbalance with respect to z, we refer to r as the ***sink*** of the drainage z.

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 2. We then compute a drainage z 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 z as if each face $p \in F \setminus \{r\}$ is sending one unit of dual flow along C to r.

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

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 z. More concretely, we define a new dart cost function $c' : \vec{E} \to \mathbb{R} \times \mathbb{N}^{2g+2}$ as follows. Let $+\!\!+$ denote the concatenation of two vectors, and define

$$c'(d) := (c(d), 1) +\!\!+ [d] +\!\!+ (z(d)).$$

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 $\sum_{d \in \vec{E}} f(d) \cdot c'(d)$. Given a cost vector c' for either a single dart or a whole flow, we refer to the components of dart and flow costs determined by homology signatures as the ***homology parts*** of c', denoted $[c']$.

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

The last component is referred to as the ***face part*** and denoted $z(c')$. 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' means first minimizing the original cost according to c, then minimizing the sum of the darts' flow values, then lexicographically minimizing the homology class of the flow, and finally choosing 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 3.1. *For any flow f and $j \in \{1, \ldots, 2g\}$, we have $[f]_j = c'_{j+2}(f)$. In addition, $c'_{2g+3}(f) = \sum_{d \in \vec{E}} f(d) \cdot z(d)$.*

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.

3.1 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. The former observation is a slight generalization of one by Patel [54, Lemma 2.4] who in turn generalized a result for planar graphs by Park and Phillips [53].

Lemma 3.2. *Let z be a drainage with sink r, and let $F' \subseteq F$. We have*

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

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

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 3.3. *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 $0 \leq a \leq 1$, we have

$$\begin{aligned} f_1(d) + a\tilde{f}(d) &= f_1(d) + a(f_2(d) - f_1(d)) \\ &\geq \min\{f_1(d), f_1(d) + f_2(d) - f_1(d)\} \\ &\geq 0 \text{ and} \\ f_1(d) + a\tilde{f}(d) &= f_1(d) + a(f_2(d) - f_1(d)) \\ &\leq \max\{f_1(d), f_1(d) + f_2(d) - f_1(d)\} \\ &\leq \mu(d). \end{aligned}$$

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 3.1 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 2.1 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 $0 < a_d \leq 1$. 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 z be the drainage used to define c'. Lemmas 3.1 and 3.2 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 3.2 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 3.4. *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 3.4 is a directed path from s to t.

Corollary 3.5. *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 ***homologically lexicographic least leftmost*** or ***holiest*** flows and paths.

4 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.* [48] for directed planar graphs. The only change necessary to derandomize their algorithm is to guarantee uniqueness of shortest paths in the *dual graph.*

Corollary 4.1. *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.*

5 MULTIPLE-SOURCE SHORTEST PATHS

Our scheme can be used in a black box fashion in the multiple-source shortest paths algorithm of Cabello *et al.* [11]. However, they depend on another property of the dart costs beyond uniqueness of shortest paths. Our perturbation scheme does guarantee the additional property, but we must first describe their algorithm in order to even explain what that property is. Understanding their algorithm is also a crucial first step in describing our linear-time algorithm for embedded graphs with constant genus and small integer dart costs. In order to more cleanly explain our linear-time algorithm in later sections, we describe a slight variant of Cabello *et al.*'s algorithm. This variant is based on the linear-time multiple-source shortest paths 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 paths 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. To move 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 $-dist_c(u, v)$. 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 $dist_\lambda$ denote $dist_{c_\lambda}$ for simplicity. Define the ***slack*** of dart $x{\to}y$ with regard to λ to be $slack_\lambda(x{\to}y) := dist_\lambda(v, x) + c_\lambda(x{\to}y) - dist_\lambda(v, y) \geq 0$. 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. Note that, with the exception of $v{\to}u$ during the special pivot, slacks do not change during pivots.

The algorithm and analysis of Cabello *et al.* [11] depend upon two genericity assumptions: all shortest paths are unique, and exactly one dart becomes tense at each critical value of λ. Suppose we apply our perturbation scheme and work perturbed costs c'. Observe that λ is now an increasing vector instead of a scalar. Corollary 3.5 guarantees that the first assumption of Cabello *et al.* is now enforced. The second assumption can be shown to hold by treating the selection of pivots as a minimum-cost flow problem, for which our perturbation scheme guarantees a unique solution (see Cabello *et al.* [11, Section 6]).

After applying our perturbation scheme, the time to do basic operations on costs increases by a factor of g. We conclude:

Theorem 5.1. *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 $O(gn \log n)$ space so that the (unperturbed) length of the shortest path from any vertex incident to r to any other vertex can be retrieved in $O(\log n)$ time.*

6 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.

Our algorithm implicitly maintains holiest shortest path trees according to a slight modification of our lexicographic perturbation scheme. First, the drainage z used to define the perturbed costs is required to use r as its sink. 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 z to perturb each dart's cost. Let $c' : \vec{E} \to \mathbb{R} \times \mathbb{N}^{2g+1}$ be the resulting perturbed costs. In order to guarantee shortest paths are still well defined and unique according to c', we must make the small assumption that there are no cycles with zero unperturbed cost.[5] Let T be the shortest path tree maintained while running Section 5's multiple-source shortest paths 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 5.

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 of 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. 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 the unperturbed slack values for every dart in the path, increment the unperturbed 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.

6.1 Preliminary Observations

We now present some useful observations, slightly modifying conventions and terminology from Erickson and Har-Peled [26] 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 [26, 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***. We let π_q denote the (possibly trivial) maximal subpath of $\bar{X}$ with one endpoint on q that contains at most one vertex that is either degree 3 or equal to r. We let ξ_q denote the orientation of π_q that begins with q.

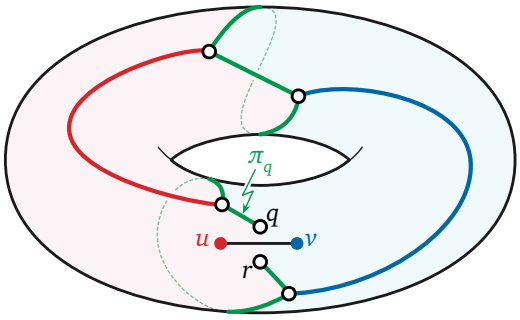

Figure 2. The 2-core $\bar{X}$ of a cut graph X on the torus. Cut paths between red vertices are red, cut paths between blue vertices are blue, and cut paths containing active darts are green.

The 2-core $\bar{X}$ of X is useful, because there is a subset of oriented cut paths containing precisely the set of active darts [11, Section 4.2]. In particular, let $\bar{P}$ denote the ***blue boundary walk***, the clockwise facial walk along dual darts of $\bar{X} \cup \{rq\}$ that includes every primal dart with a blue tail in G *except for* $r{\uparrow}q = v{\to}u$. A dart d is active if and only if d is in $\bar{P}$ but $rev(d)$ *is not*. Either every dart in a given oriented cut path has this property, or none of them do. See Figure 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, the first three of which follow immediately from the symmetry present in the definitions of slack and our perturbation scheme. We again refer the reader to the full version of this paper for proofs of the remaining lemmas.

Lemma 6.1. *Let d be any dart of T. We have* $[slack_\lambda(rev(d))] +\!\!+ z(slack_\lambda(rev(d))) = [slack_\lambda(d)] +\!\!+ z(slack_\lambda(d)) = 0$.

Lemma 6.2. *Let d be an active dart. We have* $[slack_\lambda(d)] = [cycle(T, d)] + [c'(v{\to}u)] - [\lambda]$.

Lemma 6.3. *Let $D = \langle d_1, d_2, \ldots \rangle$ be a sequence of the active darts in increasing lexicographic order of $[slack_\lambda(d_i)]$. Sequence D is in increasing lexicographic order of $[cycle(T, d_i)]$ as well.*

Lemma 6.4. *Suppose a pivot inserts dart d^+ into the holiest path tree T while removing dart d^-. Let $d = x{\to}y$ be any dart where x turns from red to blue while y does not change colors. We have* $[cycle(T - d^- + d^+, d)] = [cycle(T, d)] + [cycle(T, d^+)]$.

Lemma 6.5. *Fix an oriented cut path ξ, and let d_1 and d_2 be darts of ξ. Cycles $\gamma_1 = cycle(T, d_1)$ and $\gamma_2 = cycle(T, d_2)$ are homologous.*

[5] Our assumption does not appear necessary for guaranteeing correctness or efficiency of Eisenstat and Klein's [22] planar graph multiple-source shortest paths algorithm. Unlike our generalization, their algorithm may begin with an *arbitrary* shortest path tree. In this case, they do not maintain a holiest shortest path tree, but their leafmost pivots do provide the weaker but sufficient guarantee that shortest paths to a common endpoint do not cross.

Lemma 6.6. *Let $\vec{\Pi} = \{\xi_{i_1}, \xi_{i_2}, \ldots\}$ be the set of oriented cut paths containing darts that are* active *and whose perturbed slacks have equal homology part. 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. We have D is a subsequence of the blue boundary walk $\bar{P}$. In particular, the darts within any one oriented cut path $\pi \in \Pi$ appear as a consecutive subsequence of D.*

Lemma 6.7. *Suppose $\lambda_0 + [\lambda] = c'(v{\to}u)_0 + [c'(v{\to}u)]$ and $z(\lambda) \leq z(c'(v{\to}u))$. Then, there are no more pivots to perform in the current iteration. In particular, for any dart d with $slack_\lambda(d)_0 = 0$, we have $z(slack_\lambda(d)) > z(c'(v{\to}u)) - z(\lambda)$.*

6.2 Algorithm Outline

Based on the previous observations, we use the following strategy to compute multiple-source shortest paths. Recall, in each *iteration* of the multiple-source shortest paths algorithm, we move the source of T between consecutive vertices u and v on r. Each iteration begins with the special pivot which sets $\lambda := -dist_{c'}(u, v)$. Now, consider continuously increasing λ until it reaches $c'(v{\to}u)$.

We divide the remainder of the iteration into a number of ***rounds***. Over the course of each round, λ_0 remains a static integer while the homology and face parts of λ continuously increase. The round ends when either $\lambda = c'(v{\to}u)$ or the homology and face parts of λ become infinitely positive. If the latter case occurs, we say the round is ***fully completed***. At the end of fully completed rounds, λ_0 increases by 1, and the homology and face parts of λ become infinitely negative. We perform all pivots necessary to increase λ to the end of each round in the order these pivots occur.

An active dart d can pivot into T during the current round only if $slack_\lambda(d)_0 = 0$ which implies $[slack_\lambda(d)] + z(slack_\lambda(d)) \geq 0$; the pivot occurs when λ becomes large enough that the entire slack vector goes to 0. Therefore, we check each active dart d at the moment that $[slack_\lambda(d)] + z(slack_\lambda(d)) = 0$ to see if $slack_\lambda(d)_0 = 0$ as well. Between pivots, and in accordance with Lemmas 6.2–6.6, we do these checks cut path-by-cut path, first in lexicographic order by $[cycle(T, d)]$, and then in the order they appear along the blue boundary walk $\bar{P}$. We check the active darts within each of the oriented cut paths in order. When we detect a pivot must occur, we perform the pivot and then resume checking darts that still have non-negative homology part to their slack. By Lemma 6.1, the first of these checks occurs on the reversal of the dart just removed from T in the previous pivot. We say a dart d has been ***passed*** in the current round when $[slack_\lambda(d)] + z(slack_\lambda(d)) < 0$. Each round of checking and pivoting is further broken into three ***stages*** as follows:

(1) During the first stage, $[\lambda] < [c'(v{\to}u)]$. Following Lemma 6.2, we only check for pivots along cut paths whose darts d have $[cycle(T, d)] < 0$. By Lemma 6.7, the current iteration finishes at the end of this stage if $\lambda_0 = c'(v{\to}u)_0$.
(2) At the beginning of this stage, the homology and face parts of λ are equal to the homology and face parts of $c'(v{\to}u)$. To increase $[\lambda]$, we must check for pivots from active darts d where $[cycle(T, d)] = 0$. By Lemma 6.6, darts in ξ_q must be checked first. The stage ends *immediately* after we verify $slack_\lambda(d)_0 > 0$ for every dart d in ξ_q. This stage requires a bit of care, because we cannot afford to explicitly update π_q after every pivot. This stage most closely resembles how Eisenstat and Klein [22] handle each of the rounds as defined above, because it is the only non-trivial stage when G is a planar graph.
(3) At the beginning of this stage, $[\lambda] = [c'(v{\to}u)]$. We now check for pivots along cut paths other than π_q whose darts d have $[cycle(T, d)] \geq 0$. The end of this stage marks a full completion of the current round.

The rest of this section is organized as follows. In Section 6.3, we go over the data structures used to efficiently implement our algorithm. We then discuss checking for and performing pivots during stages 1 and 3 (Section 6.4) and stage 2 (Section 6.5) separately before discussing the special pivot (Section 6.6) in more detail. We discuss how to efficiently pick which cut path to perform pivot checks on in Section 6.7 before finally analyzing the running time of our algorithm in Section 6.8.

6.3 Data Structures

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 store 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 dual vertex $p \in F$ we store its successor dart $succ(p)$ in an arbitrary dual 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 ξ_q. From the end of stage 2 to immediately before the end of stage 1, $visited(p)$ is True if $succ(p)$ lies on ξ_q. At the end of stage 1 and between iterations, $visited(p)$ is set to False for every dual vertex p. During stage 2, it is only True if $succ(p)$ lies on ξ_q *and* it has been passed in the current round.
- We maintain the unperturbed part of parameter λ as λ_0.
- We maintain a ***reduced cut graph*** $\tilde{X}$, an embedded graph of genus g with a vertex for every vertex of the 2-core $\bar{X}$ and an edge for every cut path. We refer to vertices and edges of $\tilde{X}$ as cut vertices and cut edges, respectively. As in all embedded graphs, each cut edge π_i has two ***cut darts*** representing the two orientations of π_i. We say a cut dart is ***active*** if its oriented cut path contains active darts.
- For each dart $d \in \vec{E}$ we store its *unperturbed* slack $slack_0(d) := slack_\lambda(d)_0$ as an integer. If d's edge lies on a cut path other than π_q, we also store $\xi(d)$, the cut dart for d's oriented cut path; otherwise, we set $\xi(d)$ to Null.
- Lemma 6.5 implies that for each cut dart ξ, there is a unique ***fundamental cycle homology signature*** equal to $[d]$ for any dart d in ξ's oriented cut path. We store this value as $[\xi]$. We also store a boolean $passed(\xi)$ that is True if ξ is active and has been passed in the current round. Finally, if ξ's cut path is not π_q, then we store its darts in a list $darts(\xi)$.

- Finally, we maintain a ***finger*** f which points to a single dual vertex. When searching ξ_q for the next dart to pivot into T we will use f to track our progress.

We begin the algorithm 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.* [36], because the perturbed cost of some darts may be negative. Therefore, we begin by computing *some* shortest path tree rooted at r using the unperturbed costs c in $O(n)$ time. Let $H \subseteq \vec{E}$ be the subset of darts with 0 unperturbed 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. The initial computation of T is the *only* time our algorithm will explicitly refer to the face parts of the darts' perturbed costs. The rest of the data structures can be initialized in $O(gn)$ time. We now discuss how to handle each of the stages as described above. Performing the special pivot is handled last, because the procedure closely resembles pivots performed during these stages.

6.4 Stages 1 and 3

In both stages 1 and 3, we check for and perform pivots using darts on cut paths other than π_q. Suppose we have just performed a pivot or the current stage has just begun. We begin by discussing how to check for pivots.

Checking for pivots. We find the oriented cut path ξ containing the next dart that needs checking for a pivot, assuming one exists. The search can be accomplished by taking a walk along the cut darts of the reduced cut graph that correspond to the blue boundary walk $\bar{P}$. The active cut darts are those whose edges are encountered once during this walk. The oriented cut path we seek corresponds to the first active cut dart ξ in the walk with lexicographically least fundamental cycle homology signature among all unpassed active cut darts. Later, we describe a more efficient way to find ξ.

If $\xi = \xi_q$, then we have completed stage 1. We take a walk in the dual graph from q, following *succ* pointers until we reach a dual vertex p for which $visited(p) \neq \textsc{True}$. We unset the *visited* booleans for each dual vertex encountered during the walk. The *visited* booleans will be fixed to accurately represent ξ_q before the next run of stage 1 or 3. If there is no choice for ξ, because every active dart has been passed, then we have completed stage 3. We increment λ_0 and unset the *passed* variables for every cut dart. Then, for each active dart d, we decrement $slack_0(d)$ and increment $slack_0(rev(d))$.

If the current stage is still active, then we check the darts of $darts(\xi)$ in order, performing a pivot if we find a dart d in $darts(\xi)$ with $slack_0(d) = 0$. Suppose dart d^- was removed from T in the previous pivot, we have not yet completed a round since that pivot, and $rev(d^-)$ is in $darts(\xi)$. In this case, $rev(d^-)$ is the first dart we check.[6] Otherwise, we start with the first member of $darts(\xi)$. If we find no dart d that can be pivoted into T, we set $passed(\xi)$ to $\textsc{True}$ and repeat the search procedure for the next ξ.

[6] As discussed above, we start at $rev(d^-)$, because the homology and face parts of its slack are equal to 0. Our analysis still goes through if we check all the earlier darts of $darts(\xi)$ as well, although we will not find any pivots that use those earlier darts.

Performing a pivot. Suppose we decide to pivot dart $d^+ = x^+ \to y = o^+ \uparrow p^+$ into the holiest tree T while removing dart $d^- = x^- \to y = o^- \uparrow p^-$. Let ξ^+ be the cut dart whose oriented cut path contains d^+. Dart d^+ lies on some cut path $\pi \neq \pi_q$. Cycle $cycle(T - d^- + d^+, d^-) = cycle(T, d^+)$ is non-contractible in the surface $\Sigma - r$. Therefore, d^- will belong to a new cut path other than π_q after the pivot. We walk back through the hair of X from p^- following *succ* darts until we encounter a dual vertex p_1 such that either $\xi(succ(p_1))$ is set or $visited(p_1)$ is set. In the first case, p_1 lies on a cut path other than π_q. In the latter case, either $p_1 = q$ or it lies interior to π_q. We also walk forward through the hair from o^- following *succ* darts until we encounter a dual vertex p_2 lying on some cut path. Note we cannot have both walks end at a *visited* dual vertex. Otherwise, we will create a contractible dual cycle after adding d^-'s edge to X, implying $T + d^- + d^+$ is not connected.

After finding p_1 and p_2, we modify the reduced cut graph $\tilde{X}$ according to the new cut path from p_1 to p_2 that we found. Let π^- be the new cut edge for this cut path. We add π^- to $\tilde{X}$, possibly subdividing existing cut edges according to where we found p_1 and p_2. When we subdivide cut edges other than the one for π_q, we split the old cut edge's cut dart's dart lists. If we subdivide the cut edge for π_q, we unset *visited* for each dual vertex no longer on π_q and build new dart lists from scratch for the two new cut darts that are not orientations of the newly shortened π_q. The fundamental cycle homology signature for these new cut darts are initially set to 0 to continue respecting the fundamental cycles of T. We create new dart lists for both cut darts of π^- by just including every dart we encountered during the walks and their reversals.

We must then *remove* the cut edge π^+ for ξ^+. In a reversal of the above steps, we do a walk from o^+ along darts of $darts(rev(\xi^+))$ until we encounter a dual cut vertex p_3 of $\bar{X}$. Every dual vertex $p \neq p_3$ encountered during the walk becomes a hair of X, so we set $succ(p)$ for each of these dual vertices to follow the walk. If p_3 is an endpoint of π_q, π^+, and exactly one other cut path π, then the walk continues along π until another vertex of $\bar{X}$ is encountered, except now each dual vertex p has both $succ(p)$ set to follow the walk and $visited(p)$ set to $\textsc{True}$ to represent π_q being enlongated to the new endpoint. A similar walk and setting of dual vertex variables is performed from p^+ using darts of $darts(\xi^+)$. We then update $\tilde{X}$ by removing π^+, changing the endpoint of π_q if necessary, and merging any cut edges sharing degree 2 vertices of $\tilde{X}$ as well as their lists. Finally, we add d^+ to T by setting $pred(y) := d^+$.

Call each vertex turning from red to blue during the current pivot ***purple***. We must now compute new fundamental cycle homology signatures for cut darts whose darts in G have one purple endpoint. Observe, $[cycle(T, d^+)] = [\xi^+]$. Following Lemma 6.4, we reassign $[\xi] := [\xi] + [cycle(T, d^+)]$ for each cut dart ξ whose oriented cut edge contains darts with purple tails but not purple heads. The reversals of these cut darts are assigned the opposite fundamental cycle homology signatures.

Finally, we need to figure out which cut darts have been passed in the current round. Cut dart $rev(d^-)$ has now become active, because its tail is purple but not its head. Let ξ^- be the orientation of π^- whose oriented cut path contains d^-. Similar to above, we walk along the $O(g)$ cut darts that correspond to the blue boundary walk $\bar{P}$. By Lemma 6.1, $[slack_\lambda(rev(d))] + z(slack_\lambda(rev(d))) = 0$.

Following Lemmas 6.3 and 6.6, we set $passed(\xi) :=$ True for each cut dart ξ such that either $[\xi] < [rev(\xi^-)]$ or $[\xi] = [rev(\xi^-)]$ and ξ appears earlier in the walk than $rev(\xi^-)$.

6.5 Stage 2

In stage 2, we check for and performs pivots using darts on ξ_q. We begin by setting the finger $f := q$ and then start checking for pivots.

Checking for pivots. We do the following iterative procedure to discover the darts of ξ_q and look for pivots. If $\xi(succ(f))$ is set, we have discovered the dual vertex f where π_q is incident to other cut paths as well as which of the currently stored cut edges contains f. We update the reduced cut graph by moving π_q's endpoint to this newly discovered intersection, subdividing or merging cut edges and cut darts' dart lists as discussed above. Afterward, the current stage is over.

Suppose $\xi(succ(f))$ is not set. Then, we set $visited(f)$ to true, because $succ(f)$ must lie on ξ_q. We then check if $slack_0(succ(f)) = 0$. If so, we pivot $succ(f)$ into T as discussed below. If not, we set f to the head of $succ(f)$ to continue the search for the next pivot.

Performing a pivot. Suppose we decide to pivot dart $d^+ = x^+ \to y = o^+ \uparrow p^+$ into the holiest tree T while removing dart $d^- = x^- \to y = o^- \uparrow p^-$. We need to reassign *succ* pointers and unset some *visited* booleans based on the changing cut path π_q. Dart $rev(d^-)$ will lie on ξ_q, because otherwise there will be no path from o^+ to r in the cut graph. We add d^-'s edge to X by setting $succ(p^-) := rev(d^-)$. Then, we walk from p^-, following *succ* darts until we encounter o^+. We flip the *visited* boolean for every dart head encountered on this walk and reverse each dart we walk along to reflect the new route for ξ_q through $rev(d^-)$. Finally, we add d^+ to T by setting $pred(y) := d^+$. We set $f := p^-$ so that $rev(d^-)$ will be the next dart checked for a pivot.[7]

6.6 The Special Pivot

At the beginning of the iteration, we pivot dart $v \to u = r \uparrow q$ into the holiest tree T while removing dart $d^- = x^- \to v = o^- \uparrow p^-$. We consider two cases for how to handle the special pivot depending on whether it more closely resembles a normal pivot during stages 1 or 3 or a normal pivot during stage 2. We first remark that π_q may not be known before the special pivot occurs, because all the *visited* booleans are unset and q was just reassigned for the current iteration. However, $r \uparrow q$ lies on a cut path other than π_q if and only if it did so immediately before reassigning q and performing the special pivot.

Suppose $r \uparrow q$ lies on a cut path π^+ other than π_q. In this case, π_q is actually trivial, because q lies on π^+ as well. In particular, the *visited* booleans are accurately set to be False everywhere. We update the reduced cut graph by subdividing π^+ and its dart lists to represent the location of q. We then follow the same strategy as when we perform a pivot in stage 1 or 3, except we interpret q as o^+ and r as p^+ (with this interpretation, $o^+ \uparrow p^+$ has the same primal head as d^-). Of course, we actually set $pred(u) := v$ at the end of the process instead of the other way around. As before, the process ends with d^- belonging to a cut path other than π_q. By Lemma 6.1, we have $[slack_\lambda(rev(d^-))] \dotplus z(slack_\lambda(rev(d^-))) = 0$. Together with Lemma 6.7, we see the algorithm is now in the middle of stage 1 or stage 3.

Now, suppose $r \uparrow q$ is not on a cut path other than π_q. Similar to the strategy for stage 2, we set $succ(p^-) := rev(d^-)$ and then we follow *succ* darts starting at p^- until we reach q. We reverse all the *succ* pointers along the way and set *visited* to true for each dual vertex at the tail of the new *succ* pointers to represent the new route ξ_q must take through $rev(d^-)$. We set $pred(u) := v$. By Lemmas 6.1 and 6.7, the algorithm must be in the middle of stage 2, so we set $f := p^-$.

6.7 Searching for Cut Darts

At this point, we have a functioning algorithm that, as argued below, runs in linear time assuming g is a constant. However, the algorithm spends $O(g^2)$ time every time it starts searching a new dart list of a cut dart. To improve the running time, we maintain an ordered dictionary $\mathcal{A}$ containing active cut darts. The cut darts within $\mathcal{A}$ are sorted first in lexicographic order by their fundamental cycle homology signatures and then in the order their darts of G^* appear in the blue boundary walk $\bar{P}$. Dictionary $\mathcal{A}$ will contain at most $O(g)$ cut darts at any one time. It should support insertion and deletion in $O(g \log g)$ time each, finding the first entry in constant time, and finding the successor of *any of its current entries* in constant time.

Observe that a prefix of the cut darts in $\mathcal{A}$ have been passed. We maintain a pointer to the first member of $\mathcal{A}$ that has not been passed. Therefore, we can select a cut dart along which to do pivot checks in constant time. If no pivots are found, then the pointer is moved to the next member of the dictionary. If there is no next member, then a round has been fully completed. Slack changes are performed in time linear in the number of active darts by touching only darts and their reversals from ξ_q and cut darts in $\mathcal{A}$. Afterward, the pointer is moved to the first member of the dictionary. The dictionary is updated after a special pivot that puts the algorithm in stage 1 or 3, after completing a stage 2 that starts immediately after the special pivot, after each pivot in stages 1 and 3, and after any other stage 2 that performs a pivot.

For updates immediately following the special pivot or a stage 2 immediately following it, we remove every cut dart from $\mathcal{A}$, perform the walk in the reduced cut graph corresponding to the blue boundary walk, and add each active cut dart we encounter. The other updates occur when the reduced cut graph is changing. Each cut dart leaving the reduced cut graph is removed from the dictionary $\mathcal{A}$. Then, a walk is done in the new reduced cut graph adding each new cut dart encountered to $\mathcal{A}$. The pointer is updated to the cut dart containing the reverse of the dart of G just pivoted out of T.

6.8 Running Time Analysis

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 as well as the time spent working with the reduced cut graph and cut dart dictionary $\mathcal{A}$.

[7] As in the previous footnote, we can afford to check each new dart up to $rev(d^-)$ on ξ_q for a pivot as well, but doing so is unnecessary.

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 forward or backward along the path $u_1{\to}u_2{\to}\ldots{\to}u_{k-1}$.

Lemma 6.8. *The shortest path to x in T changes restricted homotopy class with respect to $x{\to}y$ $O(g)$ times.*

The general strategy for bounding the running time of our algorithm is to charge various interactions with darts to changes in the restricted homotopy class of their endpoints. We begin by discussing pivots from stages 1 and 3. The argument for the first case of the special pivot is similar. Suppose we decide to pivot dart $d^+ = x^+{\to}y = o^+{\uparrow}p^+$ into the holiest tree T while removing dart $d^- = x^-{\to}y = o^-{\uparrow}p^-$. Recall, a vertex x is purple if it turns from red to blue during a pivot. We may interact with every dart $x{\to}y$ where x is purple and y is not and their reversals. Cycle $cycle(T, d^+)$ 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 we spend $O(gn)$ time total interacting with individual darts during pivots in stages 1 and 3.

We now bound the time spent checking darts for pivots during stages 1 and 3. Suppose we check a dart $d = x{\to}y$. After the check, dart d has been passed. It will either continue to be passed until the algorithm terminates, it is no longer active, or its unperturbed slack decreases at the end of a round. However, we can only decrease dart $x{\to}y$'s unperturbed slack at most $c(x{\to}y) + c(y{\to}x)$ times before we have to deactivate it and start decreasing the slack of its reversal $y{\to}x$. Deactivating d requires x being purple, but not y, during a pivot, which requires a change in the restricted homotopy class of x relative to $x{\to}y$ as described above. Overall, we spend at most $O(gL)$ time doing pivot checks during stages 1 and 3.

All other interactions with individual darts occur when preparing for or executing stage 2 of a round, and in every case the *visited* boolean is switched for one of the darts' endpoints. We will bound the number of times the *visited* booleans are set to True at any point in the algorithm. Suppose we set $visited(p)$ to True and let $succ(p) = x{\to}y = o{\uparrow}p$. If the setting occurs during stage 1 or stage 3, then the restricted homotopy class of x relative to $x{\to}y$ changes as discussed above. Now suppose the setting occurs during the special pivot or during a pivot in stage 2 where dart $d^+ = x^+{\to}y = o^+{\uparrow}p^+$ pivots into holiest tree T while dart $d^- = x^-{\to}y = o^-{\uparrow}p^-$ leaves the tree. Recall, p lies on a walk in the cut graph from o^+ to p^-. Every face along this walk, including p, is enclosed by $cycle(T, d^+)$. 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. Finally, suppose $visited(p)$ is set to True during stage 2 while checking for pivots along ξ_q. Similar to above, $visited(p)$ cannot be set again until $succ(p)$ is deactivated during stage 2, $succ(p)$ is deactivated during stage 1 or 3, or the slack of $succ(p)$ is decremented at the end of the round so $visited(p)$ can be unset at the end of stage 1. The first case requires a change of restricted homotopy class as described for when $visited(p)$ is set during a pivot, the second case requires a change of restricted homotopy class as described for darts touched during stages 1 and 3, and the third case can only occur $c(x{\to}y) + c(y{\to}x)$ times before a deactivation has to occur anyway. The *visited* boolean settings, and therefore all individual dart interactions, occur $O(gL)$ times total.

Finally, we must account for the time spent interacting with the reduced cut graph and the ordered dictionary $\mathcal{A}$. There is one $O(g)$-time walk around the reduced cut graph per pivot and $O(gn)$ pivots total, so we spend $O(g^2n)$ time doing walks around the reduced cut graph. There are $O(n)$ special pivots, and we can build a fresh copy of $\mathcal{A}$ in $O(g^2 \log g)$ time after each of them for $O(g^2 n \log g)$ time total building these fresh copies. A total of $O(gn)$ cut darts enter and leave the reduced cut graph [11, Section 4.2], and each deletion or insertion from $\mathcal{A}$ takes $O(g \log g)$ time, so the individual insertions and deletions during pivots take $O(g^2 n \log g)$ time total as well.

The overall running time of our algorithm is $O(g(gn \log g + L))$. We have the following theorem.

Theorem 6.9. *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.*

7 APPLICATIONS OF LINEAR-TIME ALGORITHM

We now turn to applications of the linear-time multiple-source shortest paths algorithm described in the previous section.

7.1 Shortest Path Distances

The above algorithm successfully computes the pivots for multiple-source shortest paths around r in the order that they occur. 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 $j' \geq j$. Given a monotone correspondence $\mathcal{C}$, we can 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 $\mathcal{C}$ 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 can compute the unperturbed distance from u_i to v_{j+1} as as $dist + c(v_j{\to}v_{j+1}) - slack_0(v_j{\to}v_{j+1})$ 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 fully completed round 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 λ_0, 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 $u_i \to u_{i+1}$.

Theorem 7.1. *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.*

7.2 The Applications

We can use the algorithm of Theorem 7.1 to easily derive deterministic linear-time algorithms for a variety of problems on unweighted undirected embedded graphs of constant genus. We discuss a few of these problems in this section. For each problem, there is a published algorithm that runs in near-linear time that can use the multiple-source shortest paths algorithm of Cabello *et al.* [11] in a black box fashion. While Cabello *et al.* require uniqueness of shortest paths, it is otherwise unnecessary in these algorithms. In every case, the set of shortest path distances required by the algorithm have the form required by Theorem 7.1 with $k_2 = O(n)$.

We note our multiple-source shortest paths algorithm is not necessary for $g^{O(g)}n$ or (in some cases) $2^{O(g)}n$ time algorithms for these problems, assuming that the input graph is unweighted and undirected. However, we are unaware of any publications stating this observation explicitly. In every case, the use of our linear-time multiple-source shortest paths algorithm results in a substantial decrease in the dependency on g. We now give our improvements.

Given an undirected possibly edge weighted graph G and two vertices s and t, an s,t-cut is a bipartition (S, T) of the vertices such that $s \in S$ and $t \in T$. The ***capacity*** of an s,t-cut (S, T) is the total number/total weight of all edges with exactly one endpoint in S. The ***minimum s,t-cut*** is the s,t-cut of minimum capacity. Plugging our multiple-source shortest paths procedure into an algorithm of Erickson and Nayyeri [27], we derive the following result.

Theorem 7.2. *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.*

A ***global minimum cut*** is an s,t-cut of minimum capacity across all choices of distinct vertices s and t. We combine our algorithm with algorithms by Erickson *et al.* [25], Erickson and Nayyeri [27], and Chang and Lu [14].

Theorem 7.3. *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.*

Let graph G be embedded in surface Σ. A ***non-separating cycle*** γ in G is one for which $\Sigma - \gamma$ is connected. A shortest non-separating or non-contractible cycle is a non-separating or non-contractible cycle with a minimum number of edges or total cost if the edges have costs. For these two problems, there was no linear time algorithm known for directed graphs with small integer dart costs. However, there were $2^{O(g)}n$ time algorithms for the undirected case as mentioned above. We combine our algorithm with algorithms by Erickson [24] and Fox [31].

Theorem 7.4. *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 can 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, a ***homology basis*** is a maximal collection of cycles belonging to linearly independent homology classes with coefficients in $\mathbb{Z}_2$. A ***shortest homology basis*** is one in which the total number of edges or edge costs is minimized. We combine our algorithm with one by Borradaile *et al.* [2].

Theorem 7.5. *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.*

ACKNOWLEDGMENTS

The authors would like to thank Sergio Cabello, Erin W. Chambers, and Shay Mozes for many helpful discussions. They would also like to thank the anonymous reviewers for their helpful comments on the writing.

REFERENCES

[1] Claude Berge and Alain Ghouilla-Houri. 1965. *Programming, Games, and Transportation Networks.* Methuen & Co.
[2] Glencora Borradaile, Erin Wolf Chambers, Kyle Fox, and Amir Nayyeri. 2017. Minimum cycle and homology bases of surface-embedded graphs. *J. Comput. Geom.* 8, 2 (2017), 58–79.
[3] Glencora Borradaile, David Eppstein, Amir Nayyeri, and Christian Wulff-Nilsen. 2016. All-Pairs Minimum Cuts in Near-Linear Time for Surface-Embedded Graphs. In *Proc. 32nd Intern. Symp. Comput. Geom.* 22:1–22:16.
[4] Glencora Borradaile and Philip Klein. 2009. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM* 56, 2 (2009), 9:1–30.
[5] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. 2017. Multiple-Source Multiple-Sink Maximum Flow in Directed Planar Graphs in Near-Linear Time. *SIAM J. Comput.* 46, 4 (2017), 1280–1303.
[6] Glencora Borradaile, Piotr Sankowski, and Christian Wulff-Nilsen. 2015. Min st-Cut Oracle for Planar Graphs with Near-Linear Preprocessing Time. *ACM Trans. Algorithms* 11, 3 (2015), 16:1–16:29.
[7] Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. 2009. Directed Planar Reachability Is in Unambiguous Log-Space. *ACM Trans. Comput. Theory* 1, 1 (2009), 4:1–4:17.
[8] Ulrik Brandes and Dorothea Wagner. 2000. A Linear Time Algorithm for the Arc Disjoint Menger Problem in Planar Directed Graphs. *Algorithmica* 28, 1 (2000), 16–28.
[9] Oleksiy Busaryev, Sergio Cabello, Chao Chen, Tamal K. Dey, and Yusu Wang. 2012. Annotating Simplices with a Homology Basis and Its Applications. In *Proc. 13th Scand. Workshop Algorithm Theory.* 189–200.
[10] Sergio Cabello. 2010. Many Distances in Planar Graphs. *Algorithmica* 62, 1–2 (2010), 361–381.

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

[12] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. 2012. Homology flows, cohomology cuts. *SIAM J. Comput.* 41, 6 (2012), 1605–1634.

[13] Erin W. Chambers, Kyle Fox, and Amir Nayyeri. 2014. Counting and Sampling Minimum Cuts in Genus g Graphs. *Discrete Comput. Geom.* 52, 3 (2014), 450–475.

[14] Hsien-Chih Chang and Hsueh-I Lu. 2013. Computing the Girth of a Planar Graph in Linear Time. *SIAM J. Comput.* 42, 3 (2013), 1077–1094.

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

[16] Éric Colin de Verdière. 2012. Topological algorithms for graphs on surfaces. (May 2012). Habilitation thesis.

[17] William H. Cunningham. 1976. A network simplex method. *Math. Program.* 11 (1976), 105–116.

[18] George P. Dantzig, Alex Orden, and Philip Wolfe. 1955. The generalized simplex method for minimizing a linear form under linear inequality constraints. *Pacific J. Math.* 5 (1955), 183–195.

[19] Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. 2012. Space complexity of perfect matching in bounded genus bipartite graphs. *J. Comput. Syst. Sci.* 78, 3 (2012), 765–779.

[20] Edsger W. Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs. *Numer. Math.* 1 (1959), 269–271.

[21] Herbert Edelsbrunner and John L. Harer. 2010. *Computational Topology: An Introduction.* Amer. Math. Soc.

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

[23] Jeff Erickson. 2010. Maximum Flows and Parametric Shortest Paths in Planar Graphs. In *Proc. 21st Ann. ACM-SIAM Symp. Discrete Algorithms.* 794–804.

[24] Jeff Erickson. 2011. Shortest Non-trivial Cycles in Directed Surface Graphs. In *Proc. 27th Ann. Symp. Comput. Geom.* 236–243.

[25] Jeff Erickson, Kyle Fox, and Amir Nayyeri. 2012. Global minimum cuts in surface embedded graphs. In *Proc. 23rd Ann. ACM-SIAM Symp. Discrete Algorithms.* 1309–1318.

[26] Jeff Erickson and Sariel Har-Peled. 2004. Optimally Cutting a Surface into a Disk. *Discrete Comput. Geom.* 31, 1 (2004), 37–59.

[27] Jeff Erickson and Amir Nayyeri. 2011. Minimum cuts and shortest non-separating cycles via homology covers. In *Proc. 22nd Ann. ACM-SIAM Symp. Discrete Algorithms.* 1166–1176.

[28] Jeff Erickson and Kim Whittlesey. 2005. Greedy optimal homotopy and homology generators. In *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algorithms.* 1038–1046.

[29] Jeff Erickson and Pratik Worah. 2010. Computing the shortest essential cycle. *Discrete Comput. Geom.* 44, 4 (2010), 912–930.

[30] Lester R. Ford and Delbert R. Fulkerson. 1956. 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.

[31] Kyle Fox. 2013. Shortest Non-trivial Cycles in Directed and Undirected Surface Graphs. In *Proc. 24th Ann. ACM-SIAM Symp. Discrete Algorithms.* 352–364.

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

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

[34] Allen Hatcher. 2002. *Algebraic Topology.* Cambridge Univ. Press. http://www.math.cornell.edu/~hatcher/AT/ATpage.html

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

[36] Monika R. Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. 1997. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.* 55, 1 (1997), 3–23.

[37] Alon Itai and Yossi Shiloach. 1979. Maximum flow in planar networks. *SIAM J. Comput.* 8 (1979), 135–150.

[38] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. 2011. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proc. 43rd Ann. ACM Symp. Theory Comput.* 313–322.

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

[40] Richard M. Karp and James B. Orlin. 1981. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Appl. Math.* 3 (1981), 37–45.

[41] Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. 2011. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Proc. 38th Int. Colloq. Automata Lang. Prog. (Lecture Notes Comput. Sci.)*, Vol. 6755. Springer-Verlag, 135–146.

[42] Samir Khuller, Joseph Naor, and Philip Klein. 1993. The lattice structure of flow in planar graphs. *SIAM J. Discrete Math.* (1993), 477–490.

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

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

[45] Jakub Łącki, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. 2012. Single Source - All Sinks Max Flows in Planar Digraphs. In *Proc. 53rd IEEE Symp. Found. Comput. Sci.* 599–608.

[46] Jannik Matuschke and Britta Peis. 2010. Lattices and Maximum Flow Algorithms in Planar Graphs. In *Proc. 36th Int. Workshop Graph Theor. Concepts Comput. Sci. (Lecture Notes Comput. Sci.)*, Dimitrios M. Thilikos (Ed.). Springer-Verlag, 324–335.

[47] Bojan Mohar and Carsten Thomassen. 2001. *Graphs on Surfaces.* Johns Hopkins Univ. Press.

[48] Shay Mozes, Cyril Nikolaev, Yahav Nussbaum, and Oren Weimann. 2018. Minimum Cut of Directed Planar Graphs in $O(n \log\log n)$ Time. In *Proc. 29th Ann. ACM-SIAM Symp. Disc. Algo.* 477–494.

[49] Shay Mozes and Christian Sommer. 2012. Exact distance oracles for planar graphs. In *Proc. 23rd Ann. ACM-SIAM Symp. Discrete Algorithms.* 209–222.

[50] Shay Mozes and Christian Wulff-Nilsen. 2010. 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 (Lecture Notes Comput. Sci.)*. Springer-Verlag, 206–217.

[51] Ketan Mulmuley, Umesh Vazirani, and Vijay Vazirani. 1987. Matching is as easy as matrix inversion. *Combinatorica* 7 (1987), 105–113.

[52] James R. Munkres. 2000. *Topology* (2nd ed.). Prentice-Hall.

[53] James K. Park and Cynthia A. Philips. 1993. Finding minimum-quotient cuts in planar graphs. In *Proc. 25th Ann. Symp. Theory Comput.* 766–775.

[54] Viresh Patel. 2013. Determining Edge Expansion and Other Connectivity Measures of Graphs of Bounded Genus. *SIAM J. Comput.* 42, 3 (2013), 1113–1131.

[55] Heike Ripphausen-Lipa, Dorothea Wagner, and Karsten Weihe. 1997. The Vertex-Disjoint Menger Problem in Planar Graphs. *SIAM J. Comput.* 26, 2 (1997), 331–349.

[56] Robert Endre Tarjan. 1997. Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. *Math. Program.* 77 (1997), 169–177.

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

[58] Raghunath Tewari and N. V. Vinodchandran. 2012. Green's theorem and isolation in planar graphs. *Inf. Comput.* 215 (2012), 1–7.

[59] Shankar M. Venkatesan. 1983. *Algorithms for network flows.* Ph.D. thesis. The Pennsylvania State University. Cited in [39].

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

[61] Karsten Weihe. 1997. Maximum (s, t)-Flows in Planar Networks in $O(|V| \log |V|)$-Time. *J. Comput. Syst. Sci.* 55, 3 (1997), 454–476.

[62] Christian Wulff-Nilsen. 2009. Minimum Cycle Basis and All-Pairs Min Cut of a Planar Graph in Subquadratic Time. Preprint. (December 2009).

[63] Neal E. Young, Robert E. Tarjan, and James B. Orlin. 1991. Faster parametric shortest path and minimum balance algorithms. *Networks* 21, 2 (1991), 205–221.

[64] Afra Zomorodian. 2005. *Topology for Computing.* Cambridge Univ. Press.