All-Pairs Minimum Cuts in Near-Linear Time for Surface-Embedded Graphs[*][†]

Glencora Borradaile[1], David Eppstein[2], Amir Nayyeri[3], and Christian Wulff-Nilsen[4]

1   School of Electrical Engineering and Computer Science, Oregon State
     University, Corvallis, USA
     `glencora@eecs.oregonstate.edu`
2   Computer Science Department, Donald Bren School of Information and
     Computer Sciences, University of California, Irvine, USA
     `eppstein@uci.edu`
3   School of Electrical Engineering and Computer Science, Oregon State
     University, Corvallis, USA
     `nayyeria@eecs.oregonstate.edu`
4   Department of Computer Science, University of Copenhagen (DIKU),
     Copenhagen, Denmark
     `koolooz@diku.dk`

Abstract

For an undirected n-vertex graph G with non-negative edge-weights, we consider the following
type of query: given two vertices s and t in G, what is the weight of a minimum st-cut in G? We
solve this problem in preprocessing time $O(n \log^3 n)$ for graphs of bounded genus, giving the first
sub-quadratic time algorithm for this class of graphs. Our result also improves by a logarithmic
factor a previous algorithm by Borradaile, Sankowski and Wulff-Nilsen (FOCS 2010) that applied
only to planar graphs. Our algorithm constructs a Gomory–Hu tree for the given graph, providing
a data structure with space $O(n)$ that can answer minimum-cut queries in constant time. The
dependence on the genus of the input graph in our preprocessing time is $2^{O(g^2)}$.

1   Introduction

In recent years, problems of speeding up graph algorithms by taking advantage of low-genus
embeddings onto topological surfaces have become an important subtopic of computational
topology [11, 12, 7, 8, 6, 13, 10, 5]; in this paper, we do so for the all-pairs minimum cut
problem. In the *all-pairs minimum cut problem* we seek the minimum st-cut for every pair
$\{s, t\}$ of vertices in an edge-weighted, undirected graph G. Gomory and Hu [15] showed that
these minimum cuts can be represented by a single edge-weighted tree such that:

- the nodes of the tree correspond one-to-one with the vertices of G,

---

- for any distinct vertices s and t, the minimum-weight edge on the unique s-to-t path in the tree has weight equal to the min st-cut in G, and
- removing this minimum-weight edge from the tree creates a partition of the nodes into two sets corresponding to a min st-cut in G.

We call such a tree a minimum cut tree; it is also known as a Gomory–Hu tree or cut-equivalent tree. Gomory and Hu showed how to find this tree with $n-1$ calls to a minimum cut algorithm by building up a collection of nested cuts, in each step adding a minimum st-cut that separates a previously-unseparated pair of vertices.

New results.   We provide the first subquadratic algorithm for all-pairs minimum cuts in bounded-genus graphs. We can find the Gomory–Hu tree of a graph of genus g in time $2^{O(g^2)} n \log^3 n$, giving a data structure of size $O(n)$ which can answer minimum-cut queries in constant time. The best previous method for this class of graphs uses the standard Gomory–Hu algorithm and has a running time of $O(g^8 n^2 \log^2 n \log^2 C)$ (for integer edge weights summing to C) using the best maximum-flow algorithm to find minimum cuts [8] or $2^{O(g)} n^2 \log n$ using the best minimum-cut algorithm [12] for graphs of bounded genus. Our result hinges in part on an improvement in the time for constructing Gomory–Hu trees in planar graphs. Borradaile, Sankowski and Wulff-Nilsen showed how to solve this problem in $O(n \log^4 n)$ time [4]. In this paper, we improve this running time to $O(n \log^3 n)$ time (Section 4). Due to space constraints, some of the proofs are deferred to the full version.

From planar to bounded genus.   We reduce the problem of computing the minimum-cut tree in a graph of genus g to the same problem in a set of $2^{O(g^2)}$ planar graphs. The minimum cut, viewed as an even subgraph of the dual graph, is a collection of cycles belonging to one of 2^{2g} homology classes. Our main observation is that one can reduce the problem of finding a minimum cut that is composed of dual cycles in certain homology classes to a planar problem *before* taking into account the vertices that should be separated. This allows us to find a cut tree whose cuts are composed of dual cycles in certain homology classes. We describe this reduction in Section 3. Through this reduction, we compute minimum cut trees in $2^{O(g^2)}$ different planar graphs such that the minimum st-cut in the original graph is represented in at least one of these $2^{O(g^2)}$ cut trees. Although this would already solve the minimum cut query problem, we also show in the full version of this paper how to produce a single minimum cut tree for the original graph, by merging these cut trees in a way that preserves minimum cuts in time $O(kn \log^2 n)$ where k is the number of cut trees to merge. This tree-merging algorithm does not rely on the surface embedding. Note that one could use the Gomory-Hu algorithm of Gusfield to merge these trees, but Gusfield's algorithm has an $O(n^2)$ overhead independent of the method for finding minimum cuts [16].

Planar speed-up.   The algorithm of Borradaile et al. [4] for planar graphs implements the Gomory–Hu algorithm by using Miller's recursive cycle separator decomposition [20] to guide the selection of pairs of vertices to separate. Starting with the leaf-most pieces of the decomposition, the algorithm separates all pairs of vertices in a piece that are not yet separated. We refer to this algorithm as the *cycle-based* algorithm as it works in the dual graph finding minimum separating cycles of pairs of faces.

We improve the running time of this algorithm by addressing two bottlenecks. The first bottleneck is in finding a separating cycle. The cycle-based algorithm incurs an $O(\log^3 n)$ factor per cycle; multiplied by the depth of the recursive decomposition of the planar graph this results in an $O(\log^4 n)$ factor in the overall runtime. Instead, we find a minimum cut by

way of first computing a maximum flow, using the recursive flow techniques [2, 19], which, surprisingly (because max flow computations usually dominate minimum cut computations), reduces the overhead per cycle to $O(\log^2 n)$. We refer to this algorithm as the *flow-based* algorithm (Section 4.4). The second bottleneck is in adding a cut to the collection; we improve the overhead from an amortized $O(\log^4 n)$ per cut to an amortized $O(\log^3 n)$ per cut (Section 4.5).

Minimum cycle bases. By duality [17], our algorithm also finds the cycle weights in a minimum cycle basis of a planar graph in the same time bound; in fact, the planar all pairs min cut algorithm is really a minimum cycle basis algorithm in the dual graph. However, in graphs of higher genus and even in toroidal graphs, the minimum cycle basis appears to be mostly unrelated to the dual of minimum cuts. Borradaile et al. [1] study computing the minimum cycle bases and the minimum homology basis on surface embedded graphs.

2   Preliminaries

We consider a graph G with n vertices with a cellular embedding on an orientable surface of genus g.[1] In this paper, we use cycle in its topological sense, that is a closed curve; graph theoretically, a cycle in this sense corresponds to a closed walk. We specify simple cycle when we refer to a closed curve that visits each point (vertex, edge) at most once.

Duality. For every connected, surface embedded graph G (the *primal*) there is another connected graph embedded on the same surface, the *dual* G^*. The faces of G are the vertices of G^* and vice versa. The edges of G correspond one-for-one with the edges of G^*: for each edge e in G, there is an edge e^* in G^* whose endpoints correspond to the faces of G incident to e. Dual edges inherit the weight of the corresponding primal edges; namely, $w(e^*) = w(e)$.

Surgery. For a cycle C we define the operation of *cutting* along C in G and denote it $G \curlywedge C$. $G \curlywedge C$ is the graph obtained by cutting along C in the drawing of G on the surface, creating two copies of every edge in C. The edges in the copies of C inherit the weights of the original edges. We view $G \curlywedge C$ as being embedded on a surface with two punctures, corresponding to the two resulting copies of C. Similarly, for a path P with endpoints on different boundary components of G, we define cutting along P in G and denote it $G \curlywedge P$. We view $G \curlywedge P$ as being embedded on a surface with the boundary components containing P's endpoints unified.

$\mathbb{Z}_2$-homology. $\mathbb{Z}_2$ homology as we use it in this paper is described by Erickson and Nayyeri [12]; we refer the reader to their paper for formal definitions of the following. For further background on surface topology and homology we refer the reader to Hatcher [18]. Here, when we talk about homology we mean homology with $\mathbb{Z}_2$ coefficients.

A subgraph is called *even* if it has even degree at every vertex, or equivalently if it is the edge-disjoint union of simple cycles. An even subgraph is *null-homologous* if it is the boundary of a union of faces in G. Two even subgraphs are homologous if their symmetric difference is null-homologous. A *homology basis* is a set $\{C_1, C_2, \ldots, C_{2g}\}$ of simple cycles in G^* that generates the homology class of all cycles of G^*; it can be constructed in linear

---

[1] An embedding is cellular if every face is a topological disk. If G has any embedding on a surface of genus g, then the rotation system of the embedding gives a cellular embedding of G on a surface of genus at most g.

time [12]. The *signature* of an edge $[e]$ is defined as a $2g$-bit vector, whose ith bit is 1 if and only if $e^* \in C_i$. Given two faces a and b and an a^*-to-b^* path P in G^*, $[e]^{ab}$ is the extended $(2g+1)$-bit vector whose first bit is 1 if and only if $e^* \in P$ and whose second to $(2g+1)$st bits are $[e]$. Note that $[e]^{ab}$ depends on the choice of P. For any subgraph $X \subseteq G$ we define $[X] = \bigoplus_{e \in X} [e]$ and $[X]^{ab} = \bigoplus_{e \in X} [e]^{ab}$. Two even subgraphs X and X' are homologous if and only if $[X \oplus X'] = [X] \oplus [X'] = 0$. We use $\oplus$ to denote the symmetric difference of sets and the exclusive or of binary numbers.

Minimum cuts and minimum separating subgraphs.   The dual of a minimum st-cut is the minimum even subgraph X that is null-homologous and such that $[X]_0^{s^*t^*} = 1$, i.e. $[X]^{s^*t^*} = [1\,0\cdots0]$ (Lemma 3.1 [7]). We will call this the minimum s^*t^*-separating subgraph. In a surface of genus g, a minimum separating subgraph is composed of at most $g+1$ simple cycles. In particular, a minimum separating subgraph in a planar graph is a simple cycle. The all-pairs minimum cut problem is equivalent to the all-pairs minimum separating subgraph problem in the dual.

Faces and boundaries.   For a set of faces F, we define ∂F to be the boundary of F, that is, the set of edges that bound faces of F and $\bar{F}$. Finally, for any graph G, we use $F(G)$ to denote the set of faces of G. We additionally consider a special type of faces, *boundary* faces, which correspond to the boundary of punctures in the surface and are introduced over the course of our algorithm from the cutting along operation. If $H = G \nleftarrow X$ for some set of edges X, then $F(H)$ is $F(G)$ plus a set of additional boundary faces, bounded by edges of X.

▶ **Claim 1.** Let X be a set of edges such that $G \nleftarrow X$ has one more boundary face than G. Let S be an ab-separating subgraph in $G \nleftarrow X$ for non-boundary faces $a, b \in F(G \nleftarrow X)$. Then S is ab-separating in G.

Tight cycles and paths.   We say that a cycle C is *tight* if it is the shortest cycle with $\mathbb{Z}_2$-homology signature $[C]$. Tight cycles in all homology classes can be found in time $2^{O(g)} n \log n$ time [12]. We say that an x-to-y path P is tight if $P \cup xy$ is the shortest cycle with signature $[P \cup xy]$ in the graph $G \cup xy$ where xy is embedded on a handle added to the surface connecting a face incident with x to a face incident with y. Note that the faces may coincide, and we may need to add extra edges (with large weights to ensure they are not part of any tight cycle) to make the embedding cellular. In this way, a tight path can be found in the same time bound as a tight cycle.

Crossing or non-crossing.   Let H_1 and H_2 be two subsets of edges. We say that H_1 *crosses* H_2 if there is a subset of edges S of $H_1 \cap H_2$ such that contracting S results in a vertex s such that the edges e_1, e_2, e'_1, e'_2 are incident to s and are in this clockwise order around s with $e_1, e'_1 \in H_1$ and $e_2, e'_2 \in H_2$. Otherwise, H_1 and H_2 do not cross.

A cycle C is *non-self-crossing* if no two subpaths of C cross. A cycle is *weakly simple* if it is non-self-crossing and traverses each edge at most once. A *cycle decomposition* of an even subgraph H is a partition $\mathcal{C} = \{C_1, C_2, \ldots\}$ of the edges of H such that each cycle in $\mathcal{C}$ is simple and no two cycles in $\mathcal{C}$ cross. Chambers, Erickson and Nayyeri (Lemma 3.2 [7]) prove that every even subgraph of a surface-embedded graph has a cycle decomposition.

Cuts.   By abuse of notation, we use an xy-cut to refer to two notions: it can be either a subset C of edges whose removal from the graph separates x and y, or a bipartition (X, Y) of the vertices such that $x \in X$ and $y \in Y$. The edge subset C is the set of edges with one

endpoint in X and the other in Y. We say that a cut (A, B) *crosses* a cut (X, Y) if neither $A \subseteq X$ nor $A \subseteq Y$.

Uniqueness of minimum cuts and shortest paths. We assume that minimum cuts do not cross. That is, if (X, Y) is a minimum xy-cut and (A, B) is a minimum ab-cut, then (X, Y) and (A, B) do not cross. This is automatically true when all minimum cuts are unique, which, in turn can be assumed to be true with high probability by randomly perturbing the edge weights slightly [4, 21]. This perturbation also allows us to assume that shortest paths in the dual graph are unique, which is required for both the cycle- and flow-based planar algorithms.

Cut trees. For a graph G, an edge-weighted tree T on the same vertex set as G is a cut-tree for G if, for every edge e in T, the weight of the cut in G corresponding to the bipartition of the vertices given by $T \setminus \{e\}$ is $w(e)$. T is a *minimum* cut-tree of G if for every pair of vertices x, y the minimum xy-cut in T is the same (in value and bipartition of vertices) as the minimum xy-cut in G.

Region trees. Within our algorithms, it is convenient to associate with a cut tree T a *region tree R* [4]. R is the unique tree obtained from T by adding a leaf vertex to every node of T. Thus, R has one leaf and one internal node for each of the n vertices of T. The internal edges of R are exactly the edges of T. Such a region tree is called a *complete* region tree. The cut tree T can be recovered from R by contracting all the edges incident to leaves.

A *partial* region tree is any tree that can be obtained from a region tree by contracting a subset of internal edges. A region tree may refer either to a complete region tree or to a partial region tree, and should be clear from context. Contracting the leaf edges of a partial region tree and mapping subsets of V to the resulting nodes gives a representation of a *partial* cut tree as maintained throughout the standard Gomory–Hu algorithm.

If a region tree R is rooted at an arbitrary internal node, then each of its internal nodes u represents a *region* which is the graph obtained from G by contracting, for each non-leaf child of u in T the leaves below it to a single vertex, and similarly contracting all leaves of T not below u to a single vertex (this definition is the same as in [4] except that we define a region in the primal rather than the dual graph). For a non-root, internal node v of R, we define S_v to be the set of leaf descendents of v; S_v is one side of the cut $T \setminus \{e\}$ where e is the parent edge of v. The root of R is the region that corresponds to the entire graph G.

Cartesian trees. Our query data structure is based on a *Cartesian tree* for an edge-weighted tree T. This is a binary tree in which the interior nodes represent edges of T and the leaves represent vertices of T. The root node represents the lightest edge of T and its two children are constructed recursively from the two subtrees formed from T by removing this lightest edge. The minimum cut between any two vertices in T can then be found by answering a lowest common ancestor query between the corresponding two leaves of the Cartesian tree. Given an edge-weighted tree T whose edges have been sorted by weight, the Cartesian tree for T can be constructed and processed for constant-time lowest common ancestor queries in time and space $O(n)$ [9].

3 Reduction from bounded genus to planar

We show how to reduce the all-pairs minimum cut problem for a surface-embedded graph G to the planar case. We do so by recursively cutting along (i) a tight cycle that belongs to

some minimum cut or (ii) a tight cycle and a tight path connecting sides of the tight cycle that some minimum cut does not cross.

3.1   Reducing the genus

If S is a minimum ab-separating subgraph in graph G and C is a cycle in a cycle decomposition of S then C is a minimum ab-separating cycle in $G \nmid (S \setminus C)$ for otherwise there would be a cheaper minimum ab-separating subgraph G. The following lemma allows us to reduce the problem of finding S to that of finding C. That is, it allows us to find $S \setminus C$, and, in particular to do so without specifying the faces we wish to separate.

▶ **Lemma 2.** *Let S be the minimum ab-separating subgraph and let $C_1, C_2, \ldots, C_t$ be a cycle decomposition of S with cycles ordered by increasing cost. Then C_i is the cheapest cycle having $\mathbb{Z}_2$-homology signature $[C_i]$ for $i = 1, \ldots, t-1$. Moreover, for any $1 \leq h \leq t$, $\bigcup_{j=h}^{t} C_j$ is a minimum ab-separating subgraph in $G \nmid \bigcup_{i=1}^{h-1} C_i$.*

Proof. If $t = 1$, the lemma is trivially true. Herein, assume $t \geq 2$. For a contradiction, let C_i be the first cycle such that C_i is not the cheapest cycle having its homology signature (with $i < t$). Let C_i' be the cheapest cycle such that $[C_i'] = [C_i]$. We have:

$$w(C_i') < w(C_i) \leq w(C_{i+1}) \leq w(S \setminus C_i). \tag{1}$$

Since S is null-homologous, $[C_i'] = [C_i] = [S \setminus C_i]$. So, both $C_i' \oplus C_i$ and $C_i' \oplus (S \setminus C_i)$ are separating. Since $[S]_0^{ab} = 1 = [C_i]_0^{ab} \oplus [S \setminus C_i]_0^{ab}$ exactly one of $[C_i]_0^{ab}$ or $[S \setminus C_i]_0^{ab} = 1$. That is, exactly one of $[C_i']^{ab} \oplus [C_i]_0^{ab}$ or $[C_i']^{ab} \oplus [S \setminus C_i]_0^{ab} = 1$. Thus, either $C_i' \oplus C_i$ or $C_i' \oplus (S \setminus C_i)$ is ab-separating. By Equation 1, both $w(C_i' \oplus C_i) < w(S)$ and $w(C_i' \oplus (S \setminus C_i)) < w(S)$. This contradicts that S is the minimum ab-separating subgraph.

Since S is ab-separating in G, $H = \bigcup_{j=h}^{t} C_j$ is ab-separating in $G' = G \nmid \bigcup_{i=1}^{h-1} C_i$. For a contradiction, suppose that $H' \neq H$ is the minimum ab-separating cycle in G', that is $w(H') < w(H)$. It follows that $S' = (\bigcup_{i=1}^{h-1} C_i) \cup H'$ is ab-separating in G, and $w(S') < w(S)$, contradicting that S is the minimum ab-separating subgraph in G. ◀

The following lemma is stronger than Lemma 6.1 of Erickson et al. [11], but the proof technique is similar.

▶ **Lemma 3.** *Let S be the minimum ab-separating subgraph. Let A be any subgraph that does not cross S, and let H be the minimum subgraph such that $A \oplus H$ is null-homologous. Then H does not cross S.*

Proof. The even subgraph S separates the faces of G into two sets F_a and F_b. Without loss of generality, we assume that A and a are on the same side of S (i.e. contained in the same piece of $G \nmid S$), $a \in F_a$ and $b \in F_b$.

$A \oplus H$ is a null-homologous even subgraph, so it separates the faces of G into subsets J_b and J_b'. Assume, without loss of generality, that $b \in J_b$. Let $F = F_b \cap J_b$ and note that $a \notin F$ and $b \in F$, so $\partial(F)$ is ab-separating. Also, each edge of $\partial(F)$ is on the boundary of a face of F_b, so it does not cross S. Finally, $b \in F$, therefore, $\partial(F)$ and b are on the same side of S. In the rest of the proof, we show that $\partial(F)$ must be identical to S, thus H does not cross S.

Since S is the minimum ab-separating subgraph,

$$w(\partial F) \geq w(S). \tag{2}$$

Each boundary edge of $F = F_b \cap J_b$ should be a boundary edge in at least one of F_b and J_b. Additionally, no boundary edge of F belongs to $A \backslash S$ as A and b (so A and $\partial(F)$) are on different sides of S. Thus, $\partial F \subseteq H \cup S$.

Let $H' = H \oplus \partial F \oplus S$. Note that $H' \subset H \cup S$ (since $\partial F \subseteq H \cup S$). Because ∂F and S are boundaries of sets of faces, they are both null-homologous, so $[H'] = [H]$. Since H is minimum,

$$w(H') \geq w(H). \tag{3}$$

Finally, we show the following inequality by bounding the contribution of each edge $e \in H \cup S$ to the sides of the inequality.

$$w(\partial F) + w(H') \leq w(S) + w(H). \tag{4}$$

- If $e \in S \cap H$, then e contributes $2w(e)$ to the right side. Since, by the construction, at most one copy of each edge is included in each of ∂F and H', e can contribute at most $2w(e)$ to the left side.
- If $e \in S \oplus H$, then e is in exactly one of ∂F and H' by the definition of H' ($H' = \partial F \oplus H \oplus S$). In this case, e contributes exactly $w(e)$ to both sides of the inequality.

Therefore, all Inequalities (2), (3) and (4) must be equalities. In particular, $w(\partial F) = w(S)$. Thus, the uniqueness of the minimum cut (in the dual graph) implies that ∂F and S are identical, which in turn implies that H does not cross S. ◀

▶ **Lemma 4.** *Let S be a minimum ab-separating subgraph. Let P be an x-to-y path that does not cross S. Let H be the minimum weight subgraph such that $H \oplus P$ is a null-homologous even subgraph. H contains a tight x-to-y path P' that does not cross S.*

Proof. For $H \oplus P$ to be an even subgraph, all vertices of H except x and y must be even degree and x and y must have odd degree. Therefore, x and y must be in the same connected component C_{xy} of $H \oplus P$. Further, C_{xy} has an Eulerian x-to-y walk (i.e., a walk that uses every edge exactly once), and so C_{xy} can be decomposed into an x-to-y path P' and a set $\mathcal{C}_{xy}$ of cycles by iteratively removing cycles from the walk.

Since H is the minimum weight subgraph such that $H \oplus P$ is null-homologous, P' must be a tight path and all the cycles in $\mathcal{C}_{xy}$ must be tight, for otherwise, one could replace P' or a cycle in $\mathcal{C}_{xy}$ with a cheaper path or cycle while not changing $[H \oplus P]$.

By Lemma 3, H does not cross S, and so in particular, P' does not cross S, giving this lemma. ◀

Although cutting along a cycle C that does not cross a minimum ab-separating subgraph S reduces the genus of the surface, S may not be a minimum ab-separating subgraph in $G \nmid C$ since the minimum ab-separating subgraph in $G \nmid C$ may separate copies of C and so may not be separating in G. To overcome this, we use Lemma 4 to witness a tight path P connecting the two copies of C in $G \nmid C$. $G \nmid (P \cup C)$ then has one boundary formed by two copies of each edge in $P \cup C$. By Claim 1, a separating subgraph in $G \nmid (P \cup C)$ will also be separating in G.

▶ **Lemma 5.** *Let G be a graph embedded on a surface with b boundaries and genus g. Let S be the minimum ab-separating subgraph of G. If S is composed of at most g cycles then there is a tight cycle C and a tight C-to-C path P such that*
1. *$G \nmid (C \cup P)$ is embedded on a surface of genus $g - 1$ with $b + 1$ boundaries and*
2. *neither C nor P crosses S.*
Moreover, S is the minimum ab-separating subgraph in $G \nmid (C \cup P)$.

Proof. Since S has at most g cycles, there is a non-separating cycle C' in $G \nmid S$. Let H' be the shortest even subgraph that is homologous to C', and let C be any cycle in the cycle decomposition of H'. By Lemma 3, H' does not cross S, so, in particular, C does not cross S. Since C is non-separating, $G \nmid C$ has genus one less than G, or $g - 1$. Taking the copies of C in $G \nmid C$ to be boundaries, $G \nmid C$ has $b + 2$ boundaries.

Since C is non-separating for $G \nmid S$, there exists a path P' between the two copies of C in $G \nmid C$ that does not cross S. Let H be the minimum subgraph such that $P' \oplus H$ is null-homologous. By Lemma 4, H contains a tight path P that does not cross S and connects the two copies of C in $G \nmid C$. Then $G \nmid C \nmid P$ has genus $g - 1$ and $b + 1$ boundaries.

Since S does not cross C and P, it is ab-separating in $G' = G \nmid (C \cup P)$. For a contradiction, suppose that $S' \neq S$ is the minimum ab-separating subgraph in G', that is $w(S') < w(S)$. By Claim 1, S' is ab-separating in G, which contradicts the assumption that S is the minimum ab-separating subgraph in G. ◀

We call the cycle and path described by Lemma 5 a *tight cycle-path pair*. The following lemma ensures the possibility of reducing the genus of a graph, while looking for a minimum ab-separating subgraph, by cutting a long either a tight cycle or a tight cycle path pair.

▶ **Lemma 6.** *Let S be a minimum ab-separating subgraph in a graph G embedded in a surface of genus g. If $g \geq 1$, at least one of the following conditions holds.*
1. *There is a tight cycle $C \subsetneq S$. In this case, $S \backslash C$ is a minimum ab-separating subgraph in $G \nmid C$.*
2. *There is a tight cycle-path pair (C, P) in G such that C and P do not cross S. In this case, S is a minimum ab-separating subgraph in $G \nmid (C \cup P)$.*

3.2 A collection of planar problems

We recursively use Lemma 6 to construct a set of planar graphs, each annotated with a set of non-separating cycles of the original graph, that collectively contain the minimum ab-separating subgraphs for all pair of faces, a and b. Starting with graph G of genus g, we create a set of new graphs with genus $g - 1$, each obtained by either cutting along a tight cycle C that will belong to the separating subgraphs in the derived graphs (as per Lemma 6, part (1)) or by cutting along a tight cycle-path pair that will not cross the separating subgraphs of the derived graphs (as per Lemma 6, part (2)).

We use $\text{sep}(H, a, b)$ and $w(\text{sep}(H, a, b))$ to refer to the minimum ab-separating subgraph in H and its weight, respectively. Let $\mathcal{C}$ be the set of all tight cycles in G. Note that $|\mathcal{C}| = 2^{2g}$ by the definition of $\mathbb{Z}_2$-homology. Let $\mathcal{CP}$ be the set of all tight cycle-path pairs. For each tight cycle C and each homology class h in $G \nmid C$, $\mathcal{CP}$ contains one pair (C, P), where P is the shortest path in homology class h that connects copies of C in $G \nmid C$. Therefore, $|\mathcal{CP}| \leq 2^{2g} \times 2^{2g} = 2^{4g}$. We find the following lemma helpful for computing $\mathcal{C}$ and $\mathcal{CP}$.

▶ **Lemma 7** (Erickson and Nayyeri [12], Theorem 6.3). *Let G be an undirected graph with nonnegative edge weights, cellularly embedded on a surface of genus g with b boundary components. A minimum-weight even subgraph and a minimum weight cycle or path (with endpoints on the boundary) of G in every $\mathbb{Z}_2$-homology class can be computed in $2^{O(g+b)} n \log n$ time.*

▶ **Lemma 8.** *The sets $\mathcal{C}$ and $\mathcal{CP}$ can be computed in $2^{O(g)} n \log n$ time.*

In the following, the positive real numbers are the annotations to the planar graphs that correspond to the set of tight cycles that are cut along that should belong to the separating subgraphs (as per Lemma 6, part (1)).

▶ **Lemma 9.** *Let G be a graph embedded in a surface of genus g. There exist a set $\mathcal{H}$ of at most 2^{2g^2} planar graphs, each annotated with a set of at most g cycles, such that:*
1. *For any $(H, \mathcal{C}) \in \mathcal{H}$, $F(H)$ is $F(G)$ plus a set of boundary faces.*
2. *For any $(H, \mathcal{C}) \in \mathcal{H}$, for any $a, b \in F(G)$, $w(\text{sep}(H, a, b)) + w(\mathcal{C}) \geq w(\text{sep}(G, a, b))$.*
3. *For any $a, b \in F(G)$ there exists $(H, \mathcal{C}) \in \mathcal{H}$ such that $\text{sep}(H, a, b) \cup \mathcal{C} = \text{sep}(G, a, b)$.*
Moreover, $\mathcal{H}$ can be computed in $2^{O(g^2)} n \log n$ time.

Proof. We use induction on the genus of G. For $g = 0$, $\mathcal{H} = \{(G, \emptyset)\}$ and the properties of the lemma trivially hold.

Let $\mathcal{C}_G$ and $\mathcal{CP}_G$ be the set of tight cycles and tight cycle-path pairs for graph G of genus g. For any $C \in \mathcal{C}_G$, $G \nmid C$ has genus $g - 1$; let $\mathcal{H}_{G \nmid C}$ be the set of at most $2^{2(g-1)^2}$ annotated planar graphs guaranteed by the inductive hypothesis for the graph $G \nmid C$. For any $(C, P) \in \mathcal{CP}_G$, $G \nmid (C \cup P)$ has genus $g - 1$; let $\mathcal{H}_{G \nmid (C \cup P)}$ be the set of at most $2^{2(g-1)^2}$ annotated planar graphs guaranteed by the inductive hypothesis for the graph $G \nmid (C \cup P)$. Let

$$\mathcal{H} = \left(\bigcup_{C \in \mathcal{C}_G} \{(H, \mathcal{C} \cup C) : (H, \mathcal{C}) \in \mathcal{H}_{G \nmid C}\} \right) \cup \left(\bigcup_{(C,P) \in \mathcal{CP}_G} \mathcal{H}_{G \nmid (C \cup P)} \right).$$

Since $|\mathcal{H}_{G \nmid C}| \leq 2^{2(g-1)^2}$ and $|\mathcal{H}_{G \nmid (C \cup P)}| \leq 2^{2(g-1)^2}$ by the inductive hypothesis,

$$|\mathcal{H}| = 2^{2g} \times 2^{2(g-1)^2} + 2^{4g} \times 2^{2(g-1)^2} \leq 2 \times 2^{4g} \times 2^{2(g-1)^2} \leq 2^{2g^2}.$$

Let $T(n, g)$ be the running time of our algorithm to compute $\mathcal{H}$ (for a graph of genus g with n faces). Lemma 8 implies that $\mathcal{C}$ and $\mathcal{CP}$ can be computed in $2^{O(g)} n \log n$ time. Since $G \nmid C$ has two more faces that G and $G \nmid (C \cup P)$ than G,

$$T(n, g) \leq 2^{O(g)} T(n + 2, g - 1) + 2^{O(g)} n \log n = 2^{O(g^2)} n \log n.$$

We show that $\mathcal{H}$ satisfies the remaining properties of the lemma.

Property (1). For any C (resp. (C, P)) we have $F(G \nmid C) \supseteq F(G)$ (resp. $F(G \nmid (C \cup P)) \supseteq F(G)$) since cutting along C (resp. $C \cup P$) only adds boundary faces to G. Thus, this property holds by induction.

Property (2). Let $a, b \in F(G)$. We prove that Property (2) holds for the annotated graphs derived from $\mathcal{H}_{G \nmid C}$ and from $\mathcal{H}_{G \nmid (C \cup P)}$ separately.

Consider any $C \in \mathcal{C}$. If S is an ab-separating subgraph in $G \nmid C$ then $S \cup C$ is ab-separating in G by Claim 1. In particular, the minimum ab-separating subgraph in G has length at most $w(\text{sep}(G \nmid C, a, b)) + w(C)$: $w(\text{sep}(G \nmid C, a, b)) + w(C) \geq w(\text{sep}(G, a, b))$. By induction, we have that for any $(H, \mathcal{C}) \in \mathcal{H}_{G \nmid C}$, $w(\text{sep}(H, a, b)) + w(\mathcal{C}) \geq w(\text{sep}(G \nmid C, a, b))$. Therefore, since $(H, \mathcal{C} \cup C) \in \mathcal{H}$, and by combining the previous two inequalities gives $w(\text{sep}(H, a, b)) + w(\mathcal{C} \cup C) = w(\text{sep}(H, a, b)) + w(\mathcal{C}) + w(C) \geq w(\text{sep}(G \nmid C, a, b)) + w(C) \geq w(\text{sep}(G, a, b))$.

Likewise, for any $(C, P) \in \mathcal{CP}$, if S is an ab-separating subgraph in $G \nmid (C \cup P)$ then it must be ab-separating in G, also by Claim 1. In particular, the minimum ab-separating subgraph in G is not heavier than S: $w(\text{sep}(G \nmid (C \cup P), a, b)) \geq w(\text{sep}(G, a, b))$. By the inductive hypothesis, for any $(H, \mathcal{C}) \in \mathcal{H}_{G \nmid (C \cup P)}$, we have $\text{sep}(H, a, b) + w(\mathcal{C}) \geq \text{sep}(G \nmid (C \cup P), a, b)$. Since $(H, \mathcal{C}) \in \mathcal{H}$ and by combining these two inequalities we have $\text{sep}(H, a, b) + w(\mathcal{C}) \geq w(\text{sep}(G, a, b))$.

Property (3). Let S be the minimum ab-separating subgraph in G. At least one of the conditions of Lemma 6 must hold.

If the first condition of Lemma 6 holds, then there is a tight cycle $C \in \mathcal{C}$ such that $S \backslash C$ is a minimum ab-separating subgraph in $G \nmid C$. By the induction hypothesis, there is an annotated planar graph $(H, \mathcal{C}) \in \mathcal{H}_{G \nmid C}$ such that $\text{sep}(H, a, b) \cup \mathcal{C} = \text{sep}(G \nmid C, a, b) = S \backslash C$. Since $(H, \mathcal{C} \cup C) \in \mathcal{H}$ and $\text{sep}(H, a, b) \cup \mathcal{C} \cup C = \text{sep}(G \nmid C, a, b) \cup C = S = \text{sep}(G, a, b)$, we achieve the desired property.

If the second condition of Lemma 6 holds, then there is a tight cycle-path pair $(C, P) \in \mathcal{CP}$ such that S is a minimum ab-separating subgraph in $G \nmid (C \cup P)$. By the induction hypothsis, there is an annotated planar graph $(H, \mathcal{C}) \in \mathcal{H}_{G \nmid (C \cup P)}$ such that $\text{sep}(H, a, b) \cup \mathcal{C} = \text{sep}(G \nmid (C \cup P), a, b)$. Since $(H, \mathcal{C}) \in \mathcal{H}$ and $\text{sep}(H, a, b) \cup \mathcal{C} = \text{sep}(G \nmid (C \cup P), a, b) = S = \text{sep}(G, a, b)$, we achieve the desired property.   ◀

3.3   The algorithm

We compute the set of annotated planar graphs $\mathcal{H}$ as per Lemma 9, and then for each $(H, \mathcal{C}) \in \mathcal{H}$, we solve the all-pairs separating cycle problem in the graph H; note that it is only necessary to compute the minimum separating cycle for all pairs of non-boundary faces of H. We represent the set of minimum separating cycles of H using a minimum cut-tree T_H for the dual of the graph H. Since H is planar, we can compute T_H in $O(n \log^3 n)$ time using the planar algorithm (Section 4). We then increase the weight of each edge in T_H by $w(\mathcal{C})$.

Let $\mathcal{T}$ be the collection of all such cut trees. Note that $|\mathcal{T}| = |\mathcal{H}| = 2^{O(g^2)}$. All cut trees in $\mathcal{T}$ can be computed in $2^{O(g^2)} n \log^3 n$ time, and can be stored using $2^{O(g^2)} n$ space. For faces a and b of G, a and b are non-boundary faces of H (Lemma 9, Property (1)). The minimum ab-separating cycle in H corresponds to the minimum $a^* b^*$-cut represented by T_H, and can be determined in $O(1)$ time using a Cartesian tree representation of T_H.

By Lemma 9 (Property (2) and (3) and the weight added to each edge of T_H), the weight of the minimum ab-separating cycle is given by

$$\min_{T \in \mathcal{T}} w(\text{minimum } a^* b^*\text{-cut in } T).$$

Therefore, the minimum ab-separating cycle can be determined in $O(|\mathcal{T}|) = O(|\mathcal{H}|) = 2^{O(g^2)}$ time.

In the full version, we show how to merge k cut trees to preserve minimality in time $O(kn \log^2 n)$ time. That is, we show how to, from the set $\mathcal{T}$, compute a single min-cut tree T for minimum separating subgraphs in G (or minimum cuts in G^*). The total time to compute T by this method is $2^{O(g^2)}(n \log^2 n)$ plus the time to compute $\mathcal{T}$, or $2^{O(g^2)}(n \log^3 n)$. This will give:

▶ **Theorem 10.** *Let G be a graph embedded on a surface of genus g. The Gomory-Hu tree of G can be computed in $2^{O(g^2)} n \log^3 n$ time.*

4   Speed-up for planar graphs

Borradaile, Sankowski and Wulff-Nilsen [3, 4] gave an $O(n \log^4 n)$-time *cycle-based* algorithm for computing a Gomory–Hu tree of a planar graph $G = (V, E)$, assuming that minimum cuts are unique and so any two minimum cuts are guaranteed to nest. The algorithm is guided by a *recursive decomposition* of the graph by small, balanced separators. Working from leaf-to-root in this recursive decomposition, the algorithm finds all the minimum cuts

between unseparated vertices in a *piece* of the decomposition by finding the corresponding minimum separating cycles in the dual graph. The cycles are found by computing shortest paths explicitly within the piece and implicitly outside the piece by relying on precomputed distances outside the piece between all pairs of boundary vertices of the piece represented by an *external dense distance graph*. Computing one cycle incurs a $\log^3 n$ factor in the runtime; combined with the logarithmic depth of the recursive decomposition results in a $\log^4 n$ factor in the running time. We overcome this bottleneck by instead computing the maximum flow between each pair of unseparated vertices and then extracting the minimum cut from this flow. Similar ideas have been used for flow problems, but not for cut problems [2, 19].

In both the cycle- and flow-based algorithms, a partial region tree is updated with each newly found cut. The running time due to this update, in the original cycle-based algorithm, also met $\log^4 n$-factor bottleneck. We improve this by using a slightly modified version of the region-tree update step of the cycle-based algorithm. As in [4], we may assume, without loss of generality, that G is triangulated and has bounded degree.

4.1   Recursive Decomposition

A *piece P* is a subset of E that we regard as a subgraph of G that inherits its embedding from G. A *boundary vertex* of P is a vertex of P incident in G to a vertex not in P and we let δP denote the set of boundary vertices of P. A *hole H* of P is a face of P which is not a face of G. We sometimes regard H as the subgraph of G contained in H. Define the *boundary* of H as $\delta H = \delta P \cap H$.

A *decomposition* of P is a set of sub-pieces of P such that every edge of P belongs to a unique subpiece, except that edges with both endpoints in δP may belong to more than one subpiece. A *recursive decomposition* of G is obtained by first finding a decomposition of G and then recursing on each sub-piece until pieces of constant size are obtained. Ancestor/descendant relations between resulting set of pieces $\mathcal{P}$ are defined by their relations in the recursion tree. The specific type of recursive decomposition we use can be computed in $O(n \log n)$ time and has the following properties (Section 6, [4]):

- each piece of $\mathcal{P}$ is connected and has a constant number of holes and a constant number of child pieces
- $\sum_{P \in \mathcal{P}} |E(P)| = O(n \log n)$
- $\sum_{P \in \mathcal{P}} |\delta P|^2 = O(n \log n)$

To simplify the analysis, we shall further assume that each non-leaf piece has exactly two children; generalizing to a constant number of children is straightforward.

4.2   Dense dual-distance graphs

A dense distance graph is a weighted, complete graph on a subset of vertices of the original graph where the weight of an edge equals the shortest path distance in the original graph.

In the cycle-based algorithm, computations of recursive decompositions, distances, etc. stayed completely in the dual graph G^*. In our flow-based algorithm, we instead use a recursive decomposition of G, and compute flows and cuts in G; however, we rely on distances precomputed in G^*. For a piece P, rather than computing distances in G between vertices of δP, we compute distances in G^* between vertices of G^* that correspond to faces of G that are incident to δP. Let $F_I^*(P)$ (resp. $F_E^*(P)$) denote the vertices of G^* that correspond to faces of G that are incident to δP and in P (resp. not in P). Since G is triangulated and has bounded degree, $\sum_{P \in \mathcal{P}} |F_I^*(P)|^2 = O(n \log n)$ and $\sum_{P \in \mathcal{P}} |F_E^*(P)|^2 = O(n \log n)$.

The external dense dual-distance graph $\mathrm{DDG}(P)$ for a piece P is the dense distance graph for the vertex set $F_E^*(P)$ representing distances in the subgraph of G^* induced by the dual of edges that are not in P. Some external distances may not be finite since the complement of P is not necessarily connected; we can represent $\mathrm{DDG}(P)$ instead as a union of dense distance graphs, one corresponding to each component of the complement of P. The set of all external dense dual-distance graphs, $\{\mathrm{DDG}(P) \ : \ P \in \mathcal{P}\}$, can be computed in $O(n \log^3 n)$ time using minor modifications to an algorithm by Łacki, Nussbaum, Sankowski and Wulff-Nilsen [19]; see the full version for details.

4.3   Region subpieces

As in the cycle-based algorithm, our flow-based algorithm *processes* pieces of the recursive decomposition in a leaf-to-root order. Processing a piece P involves separating every pair of unseparated vertices in P. We maintain a region tree as described in the preliminaries. For a pair of vertices s and t in P that are not yet separated, there is a corresponding region R in the region tree that contains s and t. We focus our attention on a *region subpiece* which is $R_P = P \cap R$. Borradaile, Sankowski and Wulff-Nilsen [4] argue that for a leaf-most unprocessed piece P with child pieces P_1 and P_2, a region subpiece contains at most one pair of unseparated vertices, the number of region subpieces corresponding to P is $O(|\delta P_1 \cup \delta P_2|)$, and that all the region subpieces corresponding to P can be computed in time $O((|P| + |\delta P_1 \cup \delta P_2|^2) \log^2 n)$, have total size $O(|P|)$ and inherit a total of $O(|\delta P|)$ boundary vertices from P.

Given these bounds, in the sequel, we focus on a single region subpiece R_P with unseparated vertices s and t. The boundary vertices δR_P of R_P are inherited from P.

4.4   Separating s and t

Separating s and t is done by first computing a maximum st-flow in G which is explicitly represented on $E(R_P)$ and implicitly represented on $E(G) \setminus E(R_P)$. Given $\mathrm{DDG}(R_P)$, the running time of the algorithm is $O((|R_P| + |\delta R_P|^2) \log^2 n)$ which by the properties of the recursive decomposition and the bounds on the region subpieces is $O(n \log^3 n)$ over all pieces of the recursive decomposition. The algorithm is nearly identical to a part of the single-source, all-sinks maximum flows algorithm due to Łacki et al. (Section III C [19]) wherein they compute the flow between two cycles rather than two vertices; since s and t can be regarded as degenerate cycles, we can use the same algorithm. The main difference is that, in order to update the region tree, we must identify the cut edges corresponding to the maximum st-flow which is represented largely implicitly.

In order to explain how we determine the cut edges, it suffices to explain how flows are represented implicitly by Łacki et al., rather than explain their entire algorithm which we use as a black box. The flow is given by an explicit flow f_P on each edge of P and a circulation f_C defined in the entire graph. The latter is given by a potential function ϕ on the set of faces of G that is updated during the algorithm. The circulation f_C is defined by $f_C(uv) = \phi(f_2) - \phi(f_1)$, where $f_1 f_2$ is the (directed) dual edge corresponding to uv. It turns out that it suffices to maintain $\phi(f)$ for faces f incident to δR_P; this compact representation has been used in the recursive planar flow algorithms by Łacki et al. [19] and Borradaile et al. [2]. To see why, consider an edge fg of $\mathrm{DDG}(R_P)$; recall that f and g are faces incident to δR_P not in R_P. Edge fg corresponds to a shortest path Q_{fg} in $G^*[E(G) \setminus E(P)]$. The total flow crossing Q_{fg} is given by the sum of the flow $\phi(u) - \phi(v)$ on each edge uv of Q_{fg}; the sum is telescoping and the total flow crossing Q_{fg} is $\phi(g) - \phi(f)$. Q_{fg} is saturated by

the flow if $\phi(g) - \phi(f)$ is equal to the weight of fg in $\mathrm{DDG}(R_P)$. Since ϕ and $\mathrm{DDG}(R_P)$ are maintained, we can find all such *saturated* edges of $\mathrm{DDG}(R_P)$ in time $O(|\delta R_P|^2)$. Let $\mathrm{DDG}_0(R_P)$ be the subgraph of $\mathrm{DDG}(R_P)$ of saturated edges.

Consider a hole of P with boundary C; take the hole not to be the infinite face and order the vertices of C cyclically in a clockwise order. For two vertices a and b of $V(R_P) \cap C$, there is a residual a-to-b path in $G \setminus R_P$ only if there is no edge of $\mathrm{DDG}_0(R_P)$ from a face f incident to a part of C from a to b to a face g incident to a part of C from b to a; the path in G^* corresponding to fg consists of edges that are saturated from the a side on the hole to the b side of the hole.

For a vertex a of $V(R_P) \cap C$, we argue that we can determine the subset S_a of $V(R_P) \cap C$ of vertices that are reachable by a residual path in $G \setminus R_P$ in $O(|\delta R_P| \log n)$ time. Represent the out-neighbors of a face f of $\mathrm{DDG}_0(R_p)$ in clockwise order around C. By binary search, we can determine the last out-neighbor g of f in this order that is on C between f and a or determine that no such edge of $\mathrm{DDG}_0(R_P)$ exists. This restricts the vertices reachable from a by residual paths in $G \setminus R_P$ to those vertices on C between g and a. By considering each of the faces in order around C starting with the face immediately clockwise of a on C, we can, in this way, determine S_a in time $O(|\delta R_P| \log n)$. Repeating for every vertex of $V(R_P) \cap C$ and for every hole of P, we can build an external reachability graph representing reachability via residual paths in $G \setminus R_P$ among vertices of δR_P in $O(|\delta R_P|^2 \log n)$ time.

The s-side of the cut is given by those vertices reachable by paths that are residual with respect to the flow. We can find the subset of these vertices in R_P by searching alternately inside R_P via a straightforward search along residual edges (since the flow on edges of R_P are represented explicitly) and search along edges of the external reachability graph in time $O(|R_P| + |\delta R_P|^2)$. The total time to compute the flow using the algorithm of Łącki et al., build the external reachability graph and determine the vertices of R_P on the s-side of the cut is dominated by the Łącki et al. algorithm, which is $O((|R_P| + |\delta R_P|^2) \log^2 n)$.

4.5   Updating the region tree

We will describe how to update the region tree with this cut in terms of the corresponding separating cycle C. While performing this update we can, from the s-side of the cut within R_P, represent the minimum separating cycle by a subset of edges (non-residual edges at the boundary of the search) of R_P and a subset of the edges of $\mathrm{DDG}_0(R_P)$; the total size of this representation is $|C| = O(|R_P| + |\delta R_P|)$ and can be determined as part of the search. The algorithm we describe is a modified version of that described by Borradaile, Sankowski and Wulff-Nilsen [4] that achieves a logarithmic speed-up. We say that an edge $e \in E$ is a *boundary edge* of a region R if e is contained in the bounding cycle of R; Borradaile et al. showed that by maintaining the region tree as a top tree, we can determine in $O(\log n)$ time whether a given edge is a boundary edge of a given region. We use this fact in our analysis below.

We show how to update the region tree with C. C partitions the children of region R in the region tree. Let $\mathcal{C}_R$ be the child regions of R in the region tree and let $\mathcal{C}'_R$ be the subset contained in the inside of C. To correctly update the region tree, we need to remove $\mathcal{C}'_R$ as children of R, add them as children of C, and add C as a new child of R. If we can identify $\mathcal{C}'_R$, this update can be done in $O(|\mathcal{C}'_R| \log n)$ time since there are $O(|\mathcal{C}'_R|)$ topological changes and each change requires $O(\log n)$ update time in a top-tree representation of the region tree. If instead we have identified $\mathcal{C}_R - \mathcal{C}'_R$ then updating the region tree can be done in $O(|\mathcal{C}_R - \mathcal{C}'_R| \log n)$ time. We shall run two algorithms in parallel, one identifying $\mathcal{C}'_R$, the other identifying $\mathcal{C}_R - \mathcal{C}'_R$, and terminate both algorithms when the smaller set has been

identified in time $O(\min\{|\mathcal{C}'_R|, |\mathcal{C}_R - \mathcal{C}'_R|\} \log n)$. For simplicity, we assume that $\mathcal{C}'_R$ is the smaller set (the other case is symmetric) and show how to find $\mathcal{C}'_R$.

Let m'_R be the total number of edges (with multiplicity) that bound cycles of regions in $\mathcal{C}'_R$, excluding edges of C. We will show how to identify $\mathcal{C}'_R$ in time $O(m'_R \log^2 n + \log^3 n + |C|)$.

Over the course of all region tree updates, the second term sums up to $O(n \log^3 n)$ since only a linear number of cycles are added to the region tree. Likewise, since $|C| = O(|R_P| + |\delta R_P|)$ and $\sum_{P \in \mathcal{P}} (|R_P| + |\delta R_P|) = O(\sum_{P \in \mathcal{P}} |P|) = O(n \log n)$, the third term adds up to $O(n \log n)$. For the first term, consider distributing this cost among the cycles. Then note that a cycle pays for its edges no more than $O(\log n)$ times. To see this, note that each time a cycle pays, it gets a parent in the region tree with at most $c/2 + 1$ children where c denotes the number of children of the previous parent. This is a constant-factor decrease for $c > 2$. We cannot have $c < 2$ and if $c = 2$ the problem is trivial since the two cycles of $\mathcal{C}'_R$ must be faces (of constant complexity) of G^* since C separates at least one pair of unseparated faces. Hence, we get a total running time for all region tree updates of $O(n \log^3 n)$.

4.5.1   Identifying $\mathcal{C}'_R$

We identify $\mathcal{C}'_R$ in two steps. In the first step, we identify those edges that have exactly one endpoint in C and belong to cycles in $\mathcal{C}'_R$. In the second step, we explore the interior of C starting at these edges to identify the boundaries of all the cycles of $\mathcal{C}'_R$. We shall only describe the first step as the second step is done exactly as by Borradaile et al. in $O(m'_R \log n)$ time [4]. Recall that C is represented by edges of a region subpiece, which we refer to as *regular edges*, together with *super edges* in the external dense dual distance graphs, each representing a path in G^*. In the following, we assume that C consists only of super edges as the regular edges are easy to handle using the top tree representation [4].

Each super edge fg represents a shortest path in G^* and was found using a fast Dijkstra implementation of Fakcharoenphol and Rao's recursive shortest path algorithm [14] to construct a dense dual distance graph. In this construction, the path in G^* corresponding to fg has a recursive representation in line with the recursive decomposition of G. That is, fg decomposes into a path Q_{fg} of edges which themselves are super edges in (internal and external) dense dual distance graphs. The super edges of this path can be recursively decomposed until reaching edges of G^*. The number of recursion levels is at most the depth $O(\log n)$ of the recursive decomposition of G^*. We shall assume that any super edge ab (i) points to the endpoints in G^* of the subpath Q_{ab} that ab decomposes into at the next recursion level, (ii) points to the super edge of Q_{ab} that contains the midpoint of Q_{ab} (as a path in G^*), and (iii) is annotated with its length in terms of number of edges of G^* (as well as its length in terms of weights of those edges). It is easy to maintain this information during the construction of the dense dual distance graphs without an asymptotic time increase.

A minimum separating cycle in a planar graph is an isometric cycle: for any vertex r on C, C consists of two shortest paths Q_1 and Q_2 from r to vertices a and b, respectively, together with the single edge ab of G^* [17]. Borradaile, Sankowski and Wulff-Nilsen show how to find an a, b and r in time $O(\log^3 n + |C|)$ along with a representation of Q_i consisting of at most $|C|$ super edges at the top level of the recursion and $O(\log n)$ paths of super edges from dense dual distance graphs at deeper recursion levels for $i = 1, 2$ (Section 4.2 [4]). For $i = 1, 2$, recall that we need to find all edges of G^* with exactly one endpoint on Q_i that are contained in boundary cycles of $\mathcal{C}'_R$. By symmetry, we restrict our attention to Q_1.

The algorithm is as follows. First, identify the first and last edge (of G^*) on Q_1. Denote them by e_1 and e_2, respectively. Then check if e_1 and e_2 are incident to the same cycle

C' of $\mathcal{C}'_R$. As noted above, each such check can be done in $O(\log n)$ time using a top tree representation of the region tree.

If this check is positive then, since bounding cycles are isometric and shortest paths are unique, all edges of Q_1 are in $C' \cap C$ and we are done since none of these edges can be interior to a cycle of $\mathcal{C}'_R$ since these cycles have disjoint interiors. (A special case is when a cycle of $\mathcal{C}'_R$ is a face of G^* which need not be isometric but, since faces have constant size, this will not affect the analysis.)

If this check is negative, we find an edge e_m of G^* belonging to Q_1 such that a constant fraction of the (regular) edges of Q_1 are before e_m and a constant fraction of them are after e_m. We then recurse on the two subpaths of Q_1 from e_1 to e_m and from e_m to e_2, respectively. If there are k cycles of $\mathcal{C}'_R$ incident to Q_1 then the total number of regular edges e_m considered is $O(k \log n)$. This is where our improvement differs from the original region-tree-updating technique of Borradaile et al. [4]: we show how to identify each edge e_m in $O(\log n)$ time instead of $O(\log^2 n)$ time, which was the bottleneck of their method. Total time to identify all $O(k \log n)$ edges is then $O(k \log^2 n) = O(m'_R \log^2 n)$.

First we need a lemma. For a path Q, we denote by $Q[x, y]$ the subpath from x to y and we call a vertex x a *split point* of Q if at most $1/8$ of the vertices of Q are before resp. after x.

▶ **Lemma 11.** *Let $Q = a \rightsquigarrow c \rightsquigarrow e \rightsquigarrow f \rightsquigarrow d \rightsquigarrow b$ be a directed path where possibly some of the subpaths are empty. Let m (resp. $m(c, d)$) denote the midpoint of Q (resp. $Q[c, d]$), splitting the path into two (almost) equal-size subpaths such that $|Q[c, d]| > \frac{1}{4}|Q|$, $m \in Q[c, d]$, and $m(c, d) \in Q[e, f]$. We have:*

If $m \in Q[c, e]$ then e is a split point of Q; if $m \in Q[f, d]$ then f is a split point of Q; if $m \in Q[e, f]$ and $|Q[e, f]| \leq \frac{1}{4}|Q|$ then e is a split point of Q.

We use this lemma to find an edge e_m with the property described above. Recall that we store the number of edges of G^* represented by each super edge ab as well as a pointer to the sub-super edge that contains the midpoint of Q_{ab}. As described above, path Q_1 is represented by $O(|C| + \log n)$ subpaths and since we know the length of each of them, we can in $O(|C| + \log n)$ identify the subpath Q'_1 that contains the midpoint of Q_1. Q'_1 plays the role of $Q[e, f]$ and Q_1 plays the role of $Q[c, d]$ and $Q[a, b]$ in Lemma 11 (so that $a = c$ and $b = d$). The assumptions in the lemma hold and we can check in constant time whether any of the three cases apply. If so, we are done as we have found a split point and we can pick e_m as an edge incident to this point. Otherwise, we have $m \in Q[e, f]$ and $|Q'_1| = |Q[e, f]| > \frac{1}{4}|Q|$. The path $Q[e, f]$ belongs to a shortest path of a dense dual distance graph, so in constant time, we can identify the super edge uv of this path that contains its midpoint. Now apply the lemma again but with $Q[e, f]$ defined as uv, $Q[c, d]$ defined as Q'_1, and $Q[a, b]$ defined as Q_1. Then the assumptions in the lemma hold again. Applying it, we either find a split point or we decompose uv into a shortest path of super edges, and recurse on the super edge containing the midpoint of this path. Each step runs in $O(1)$ time and since there are $O(\log n)$ recursion levels, we obtain a split point and hence e_m in $O(\log n)$ time.

We conclude from the above that the total time to add bounding cycles to the region tree is $O(n \log^3 n)$.

─── **References** ───

1   G. Borradaile, E. Chambers, K. Fox, and A. Nayyeri. Computing minimum homology basis and minimum cycle basis in surface embedded graphs. In submission., 2015.
2   G. Borradaile, P.N. Klein, S. Mozes, Y. Nussbaum, and C. Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In *Proc. 52nd*

Symp. Found. of Computer Science (FOCS 2011), pages 170–179, 2011. `doi:10.1109/FOCS.2011.73`.

3   G. Borradaile, P. Sankowski, and C. Wulff-Nilsen.   Min *st*-cut oracle for planar graphs with near-linear preprocessing time. In *Proc. 51st IEEE Symp. Foundations of Computer Science (FOCS 2010)*, pages 601–610, 2010. `doi:10.1109/FOCS.2010.63`.

4   G. Borradaile, P. Sankowski, and C. Wulff-Nilsen. Min *st*-cut oracle for planar graphs with near-linear preprocessing time. *ACM Trans. Algorithms*, 2014. To appear.

5   S. Cabello, É. Colin de Verdière, and F. Lazarus. Finding shortest non-trivial cycles in directed graphs on surfaces. In *Proc. of the 26th Annual Symposium on Computational Geometry*, SoCG'10, pages 156–165, New York, NY, USA, 2010. ACM. `doi:10.1145/1810959.1810988`.

6   E. Chambers, J. Erickson, and A. Nayyeri. Homology flows, cohomology cuts. In *Proc. 41st ACM Symposium on Theory of Computing (STOC'09)*, pages 273–282, 2009. `doi:10.1145/1536414.1536453`.

7   E. Chambers, J. Erickson, and A. Nayyeri. Minimum cuts and shortest homologous cycles. In *Proc. 25th Symp. Computational Geometry (SoCG'09)*, pages 377–385, 2009. `doi:10.1145/1542362.1542426`.

8   Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM J. Comput.*, 41:1605–1634, 2009. `doi:10.1137/090766863`.

9   E. Demaine, G. Landau, and O. Weimann. On Cartesian trees and range minimum queries. *Algorithmica*, 68(3):610–625, 2014. `doi:10.1007/s00453-012-9683-x`.

10   J. Erickson. Shortest non-trivial cycles in directed surface graphs. In *Proc. of the 27th Annual Symp. on Computational Geometry*, SoCG'11, pages 236–243, New York, NY, USA, 2011. ACM. `doi:10.1145/1998196.1998231`.

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

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

13   J. Erickson and A. Sidiropoulos. A near-optimal approximation algorithm for asymmetric tsp on embedded graphs. In *Proc. 30th Annual Symposium on Computational Geometry*, SOCG'14, pages 130:130–130:135. ACM, 2014. `doi:10.1145/2582112.2582136`.

14   J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006. `doi:10.1016/j.jcss.2005.05.007`.

15   R. Gomory and T. Hu. Multi-terminal network flows. *Journal of SIAM*, 9(4):551–570, 1961. `doi:10.1137/0109047`.

16   D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM J. Comput.*, 19(1):143–155, 1990. `doi:10.1137/0219009`.

17   D. Hartvigsen and R. Mardon. The all-pairs min cut problem and the minimum cycle basis problem on planar graphs. *SIAM J. on Discrete Math.*, 7(3):403–418, 1994. `doi:10.1137/S0895480190177042`.

18   A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

19   J. Łacki, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Single source – all sinks max flows in planar digraphs. In *Proc. 53rd Symp. Found. of Computer Science (FOCS 2012)*, pages 599–608, 2012. `doi:10.1109/FOCS.2012.66`.

20   G. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32(3):265–279, 1986. `doi:10.1016/0022-0000(86)90030-9`.

21   K. Mulmuley, V. Vazirani, and U. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):345–354, 1987.