Algorithms for the Edge-Width of an Embedded Graph*

Sergio Cabello† Éric Colin de Verdière‡ Francis Lazarus§

February 15, 2012

Abstract

Let G be an *unweighted* graph of complexity n embedded in a surface of genus g, orientable or not. We describe improved algorithms to compute a shortest non-contractible and a shortest non-separating cycle in G.

If k is an integer, we can compute such a non-trivial cycle with length at most k in $O(gnk)$ time, or correctly report that no such cycle exists. In particular, on a fixed surface, we can test in linear time whether the edge-width or face-width of a graph is bounded from above by a constant. This also implies an output-sensitive algorithm to compute a shortest non-trivial cycle that runs in $O(gnk_0)$ time, where k_0 is the length of the cycle.

We also give an approximation algorithm for the shortest non-trivial cycle. If a parameter $0 < \varepsilon < 1$ is given, we compute in $O(gn/\varepsilon)$ time a non-trivial cycle whose length is at most $1+\varepsilon$ times the length of the shortest non-trivial cycle.

Keywords: Topological graph theory, computational topology, edge-width, face-width, surface, embedded graph.

1 Introduction

Let Σ be a surface, possibly non-orientable, of genus g with b boundaries. Let G be an unweighted and undirected graph embedded on Σ of *complexity* n (this is the total number of vertices and edges of G), where all the faces of G are disks. In this article, we are interested in the *edge-width* and *face-width* of G, which roughly measure how G "locally looks planar". Specifically, the *edge-width* of G is the minimum number of edges in a non-contractible cycle in G [2, 23]. The *face-width* of G (also known as *representativity*) is the minimum number of points, over all non-contractible closed curves γ on Σ, that γ and G have in common [24] (note that the curves γ can go through vertices of G). There are also the corresponding concepts of non-separating edge-width and non-separating face-width, where the requirement of being non-contractible is strengthened into being

*A preliminary version appeared in Proc. ACM Symp. on Computational Geometry, 2010, pp. 147–155. Research partially supported by the Slovenian Research Agency, program P1-0297 and project BI-FR/09-10-PROTEUS-014, funded by the French Ministry of Foreign and European Affairs.

†Department of Mathematics, IMFM, and Department of Mathematics, FMF, University of Ljubljana, Slovenia. E-mail: sergio.cabello@fmf.uni-lj.si.

‡Laboratoire d'informatique, École normale supérieure, CNRS, Paris, France. Email: Eric.Colin.de.Verdiere@ens.fr.

§GIPSA-Lab, CNRS, Grenoble, France. E-mail: Francis.Lazarus@gipsa-lab.grenoble-inp.fr.

non-separating in Σ. This article gives improved algorithms to compute these parameters. Before describing our new results in detail, we discuss related works.

Related Algorithmic Results. There has been much research on computing a shortest *non-trivial* cycle on an embedded graph. Here, non-trivial means either non-contractible or non-separating. Such cycles are the fundamental tool to perform surgery on embedded graphs. The first algorithm, by Thomassen [26], finds a shortest non-trivial cycle in cubic time. In essence, for each vertex of G, the algorithm computes a breadth-first search (BFS) tree T rooted at that vertex, and, for every non-tree edge e, tests whether the cycle formed by T and e is non-trivial; finally, the shortest such cycle is returned. In particular, the shortest non-trivial cycle has no repeated vertex.

Other works on this topic include the computation of shortest non-trivial cycles in the more general situation of non-negatively *weighted* graphs. Erickson and Har-Peled [11] extend Thomassen's algorithm to this setting and decrease its complexity to $O(n^2 \log n)$, by interleaving Dijkstra's algorithm with tests for triviality. This is the best current result, and an algorithm with subquadratic running time would give rise to a subquadratic-time algorithm to find the girth of sparse graphs [5]. In the same paper, Erickson and Har-Peled [11, Corollary 5.8] give an $O(gn \log n)$-time algorithm that computes a non-trivial cycle whose length is at most twice the length of the shortest non-trivial cycle.

Efficient algorithms for the case where the genus is small have also been investigated [6, 19]. The best known algorithm by Cabello and Chambers [3] computes the shortest non-contractible (resp. non-separating) cycle on an orientable surface in $O((g+b)g^2 n \log n)$ (resp. $O(g^3 n \log n)$). As a consequence, the (possibly non-separating) edge-width and face-width of a graph in a fixed orientable surface can be computed in $O(n \log n)$ time. In another paper [4], we also consider the more general scenario of finding shortest non-trivial cycles in *directed* graphs.

Kawarabayashi and Mohar[1] point out that their results in [16] imply that the *face-width* k_1 of a graph can be computed in $2^{O(gk_1)}n$ time, assuming that $k_1 \geq 3$. This approach relies on graph minors, and in particular, the relations between the face-width and tree-width of embedded graphs. Their algorithm has an exponential dependency on the genus and the face-width; it relies on an unknown, though theoretically computable, list of minimal graphs and it does not extend to the problem of computing the edge-width.

The Unweighted Case. It is natural to ask whether the algorithms for the weighted case can be made faster when restricted to unweighted graphs. They all use shortest path trees, computed in $O(n \log n)$ time in the weighted case using Dijkstra's algorithm; this step can be speeded up to linear time in the unweighted case, for which BFS trees are shortest path trees. However, the current results are also relying on the minimum cut problem in planar graphs [19] or on dynamic trees [3], and require an extra logarithmic factor that is independent from the shortest path tree computation. The $O(n^2 \log n)$ time algorithm by Erickson and Har-Peled [11, Lemma 5.2] immediately gives an $O(n^2)$ time algorithm for the non-contractible case, but in the non-separating case, the complexity is still $O(n^2 \log n)$, because the extra logarithmic factor appears in their recurrence, independently of the shortest path tree computation.

Our Results. Recall that, in this paper, G is an unweighted graph of complexity n cellularly embedded on a surface Σ (orientable or not) of genus g with b boundaries. Our main result is an algorithm to decide in $O((g+b)nk)$ time whether the edge-width of G is at most k. In particular, on a fixed surface, we can decide in linear time whether the edge-width is bounded from above by

[1] Private communication.

a constant. By running the decision algorithm for exponentially increasing values of k, we obtain an output-sensitive algorithm to compute the edge-width k_0 of G in $O((g+b)nk_0)$ time. We obtain similar results for the non-separating edge-width, where in the running time the term $g+b$ is replaced by g.

We also give an algorithm to compute a $(1+\varepsilon)$-approximation of the edge-width (resp. non-separating edge-width) in $O((g+b)n/\varepsilon)$ (resp. $O(gn/\varepsilon)$) time. This is an enhancement over the 2-approximation algorithm by Erickson and Har-Peled [11, Corollary 5.8] mentioned above.

Incidentally, we give alternate algorithms to find shortest non-trivial loops in linear time. The algorithm works for weighted graphs with an extra logarithmic factor. Our algorithms are arguably simpler to implement than those by Erickson and Har-Peled [11, Lemma 5.2]; some ideas of the proof are inspired from the paper by Erickson and Whittlesey to compute shortest homotopy generators [12]. Compared to Erickson and Har-Peled [11], our algorithms are faster by a logarithmic factor for the non-separating case in unweighted graphs, and have the same asymptotic running-time otherwise.

We note that all our results concerning the edge-width parameter have immediate counterparts for the face-width. Indeed, the *vertex-face incidence graph* Γ of G (also called *radial graph*) is also embedded on Σ; the graph Γ has the same asymptotic complexity as G, and its edge-width is twice the face-width of G [22, Proposition 5.5.4]. We also emphasize that all our algorithms are quite simple and do not require heavy data structures like the self-adjusting top trees needed by Cabello and Chambers [3].

Our output-sensitive complexity can be combined with combinatorial bounds on the edge-width and face-width that are known. Hutchinson [15] showed that a triangulation with m vertices in an orientable surface without boundary has edge-width $O(\sqrt{m/g}\log g)$ if $g \leq m$ and $O(\log g)$ if $g > m$. This result can be extended to non-orientable surfaces, and the same bound applies to the face-width of arbitrary embedded graphs [6, Lemma 13 and Theorem 14]. Therefore, our algorithm implies that the edge-width of a triangulation and the face-width of an arbitrary graph in a surface (orientable or not) without boundary can be computed in $O(n^{3/2}g^{1/2}\log g)$ time, provided that $g \leq n$. This time bound is subquadratic unless $g\log^2 g = \Omega(n)$.

Applications. Edge-width and face-width were introduced in the field of topological graph theory (see Mohar and Thomassen [22, Chapter 5] for a survey). Graph embeddings with large face-width share many properties with planar graphs. In topological graph theory there are several results of the following form: for any fixed surface Σ there exists a constant $c = c(\Sigma, \Pi)$ such that any graph embedded in Σ with face-width (or edge-width) at least c has property Π. See for example [27, 9, 28, 23, 21, 16], and [22, Chapter 5]. Our results provide a linear-time algorithm to test if the hypotheses of those results are fulfilled.

Getting bounds on the edge-width or the face-width has been explicitly used as subroutine in algorithms that work in linear time on a surface of bounded genus, for computing crossing numbers of graphs [17], for graph isomorphism of graphs that admit polyhedral embeddings [16], and for finding certain induced cycles in embedded graphs [18]. In these articles, the authors use the 2-approximation of the edge-width mentioned above. Using this 2-approximation instead of the real edge-width affects exponentially the running time; however, this is hidden in the O-notation because the authors consider a fixed surface. Using our new algorithm to compute the edge-width removes this overhead.

Also, there is a closed formula for computing the orientable genus of a graph that can be embedded in the projective plane. Indeed, Fiedler et al. [13] have shown that a graph G that can be embedded in the projective plane with face-width $k \neq 2$ has orientable genus $\lfloor k/2 \rfloor$. Our results

imply an algorithm to compute the orientable genus $g(G)$ of such graphs in $O(g(G)n)$ time.

The techniques we use here to work with loops are also useful in another article [4], where we obtain efficient algorithms that find shortest non-trivial cycles in a more general setting than previously studied, namely, for *directed* weighted graphs on surfaces.

Overview of the Techniques. Our algorithm to decide if the edge-width is bounded by k consists of two steps: (1) We first compute a set of vertices K such that any non-trivial cycle has to use some vertex of K. (2) For every vertex s in K, we compute the shortest non-trivial cycle passing through s in the graph induced by the vertices at distance at most $k/2$ from s. The key idea in our approach is to find an efficient way to carry Step (2) simultaneously for several basepoints that are far apart on the surface. This idea, in turn, requires to choose K in Step (1) adequately. This strategy also requires to be able to test in constant time whether a loop with a special structure is trivial; we introduce a technique for this purpose, which is of independent interest.

The algorithm to $(1+\varepsilon)$-approximate the edge-width first computes a 2-approximation of the edge-width. Then we proceed as in the previous paragraph, but since we are only interested in an approximately shortest non-trivial cycle, we can discard some vertices of K to speed up the algorithm.

2 Background and Terminology

2.1 Graph Theory

All the considered graphs may have loop edges and multiple edges. We sometimes denote by xy an edge with endpoints x and y: even if this notation is ambiguous in presence of multiple edges, it will always be clear from the context which edge is considered. A *path* is a walk without repeated vertices; a *cycle* is a closed walk without repeated vertices. A *loop* with basepoint s is a closed walk with a distinguished occurrence of vertex s.

Suppose G is connected; consider a spanning tree T of G. For any vertex s and any edge uv of G, we denote by $\tau(T, s, uv)$ the loop consisting of the path in T from s to u, the edge uv, and the path in T from v to s. We also denote by $\tau(T, uv)$ the closed walk consisting of the edge uv and the path in T between u and v. Note that $\tau(T, uv)$ is a cycle, if uv is not in T.

2.2 Topology

We assume that the reader is familiar with topology of surfaces. See any of the books by Hatcher [14], Massey [20], or Stillwell [25] for a comprehensive treatment.

We will work with surfaces (i.e., 2-manifolds), possibly with boundary and possibly non-orientable. We will use Σ to denote a surface, with g for its genus and b for its number of boundary components. For simplicity, we use the *Euler genus* $\bar{g}$ of the surface, which is defined as $\bar{g} = 2g$ when Σ is orientable and $\bar{g} = g$ otherwise. $\overline{\Sigma}$ denotes the surface without boundary obtained by attaching a disk to each boundary component of Σ.

Homotopy of loops in G is considered with a fixed basepoint. We will consider homology with $\mathbb{Z}_2$ coefficients. A simple loop is null-homologous on $\overline{\Sigma}$ if and only if it separates $\overline{\Sigma}$ (or, equivalently, Σ). As in previous articles, when we write "separating on Σ", we really mean "null-homologous on $\overline{\Sigma}$": these two notions coincide for simple loops, but the latter is also defined for non-simple loops, which turns out to be useful. Therefore, a contractible loop is a separating loop, even for non-simple loops. Henceforth, we use the term *non-trivial* as a shorthand for non-contractible or non-separating.

2.3 Graph Embeddings

In this article, G is *cellularly embedded* on Σ if the faces of G on $\overline{\Sigma}$ are open disks. In particular, each face of a cellular embedding on Σ is an open disk with zero, one, or more disjoint open disks removed; the boundaries of these open disks belong to the boundary of Σ. When considering the problem of finding a shortest non-contractible loop or cycle, we assume that every face of G contains at most one boundary of Σ. This is possible since several boundary components of Σ in one face of G can be replaced with one single boundary component without changing the contractibility character of the loops in G. When considering the computation of shortest non-separating loops or cycles, we assume our input surface Σ has no boundary. This is valid since a cycle is separating in Σ if and only if it is separating in $\overline{\Sigma}$.

We assume that the embedding is represented in a suitable way, like for example the *gem representation*, using the incidence graph of *flags* (vertex-edge-face incidences) discussed by Eppstein [10], or rotation systems [22]. For orientable surfaces, one can also use the DCEL that is customary in Computational Geometry (see, e.g., de Berg et al. [8]). More precisely, we store the embedding of G on $\overline{\Sigma}$, and mark within each face whether it contains a boundary component of Σ.

If G is a graph embedded on Σ, we will denote by $\Sigma \backslash\backslash G$ the surface with boundary obtained after cutting Σ along G. Note that $\Sigma \backslash\backslash G$ is a surface with boundary. In contrast, $\Sigma \setminus G$ is a set operation where we remove from Σ the points in (the image of the embedding of) G. In particular, if G is cellularly embedded, $\overline{\Sigma} \backslash\backslash G$ is a union of closed disks, whereas $\overline{\Sigma} \setminus G$ is a union of open disks.

Let G be a graph cellularly embedded in Σ. Its *dual graph*, denoted by G^*, has for vertices the set of faces of Σ and for edges the set of edges (dual to) $E(G)$: two faces are adjacent if they share an edge of G. The edge dual to e is denoted by e^*, and it connects the two faces adjacent to e in the embedding. The dual graph G^* has a natural embedding in Σ: each vertex f^* of G^*, corresponding to face f of G, is assigned to a point p_f of the interior of the face f; for each edge uv of G, incident to faces f and f', the dual edge $(uv)^*$ is assigned a curve that connects the points p_f and $p_{f'}$ and crosses G precisely at the edge uv. For our discussion, it is convenient to make the following assumptions: for every face f of G containing a boundary component (which we assumed to be unique in that face) the point p_f belongs to that boundary component, and we add to G^* a loop edge, whose image is the boundary component. (Such loop edges correspond to no edge of the primal graph G.) For a set of edges $A \subseteq E(G)$, we use the notation $A^* = \{e^* \mid e \in A\}$.

3 Shortest Non-Trivial Loops

In this section, we develop tools to work with non-contractible and non-separating loops.

As already noted, a linear-time algorithm for computing a shortest non-contractible loop follows directly from Erickson and Har-Peled [11, Lemma 5.2] by replacing Dijkstra's algorithm with a breadth-first search. For a non-separating loop we provide a new algorithm that improves upon previous articles [11, 6, 3] for unweighted graphs. The ideas are closely related to Erickson and Whittlesey's algorithm to compute a shortest system of loops [12] (specifically, a shortest non-separating loop is the first loop computed by their greedy algorithm, although they do not compute it in linear time). The idea of computing shortest non-trivial loops using this method appears in one of the authors' course notes [7].

Let T be an arbitrary spanning tree of G; let C^* be the subgraph of G^* with the same vertex set as G^* and edge set $E(G^*) \setminus E(T)^*$. (In particular, C^* contains every loop edge of G^* in a boundary component of Σ.)

Lemma 1. *C^* is a cut graph of Σ; that is, $\Sigma \setminus C^*$ is (homeomorphic to) an open disk.*

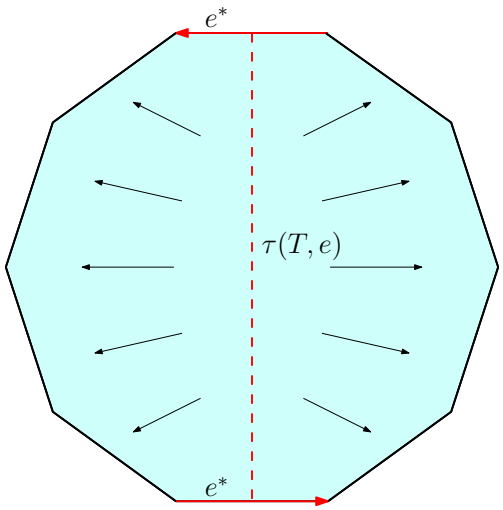

Figure 1: The retraction in the proof of Lemma 2. The boundary of the disk is the boundary of $\Sigma \setminus\!\setminus C^*$.

Proof. $\Sigma \setminus C^*$ can be obtained by taking all faces of G^* (which are open disks) and gluing them in a tree-like fashion according to the tree T, i.e., along the edges of $E(T)^*$. Since attaching disks in this way gives a disk, we obtain that $\Sigma \setminus C^*$ is a disk. □

The next lemma comes from Erickson and Whittlesey [12, Section 3.4]. Recall that a subspace A of a topological space X is a *deformation retract* of X if there is a continuous map $F : X \times [0, 1] \to X$ such that for every x in X and a in A, we have $F(x, 0) = x$, $F(x, 1) \in A$, and $F(a, 1) = a$.

Lemma 2. *Let $e \in E(G) \setminus E(T)$. Then $C^* - e^*$ is a deformation retract of $\Sigma \setminus \tau(T, e)$.*

Proof. The cycle $\tau(T, e)$ cuts C^* at exactly one point. Therefore this cycle corresponds to a path intersecting the boundary of the closed disk $\Sigma \setminus\!\setminus C^*$ exactly at its endpoints. (See Figure 1.) Both halves of the disk retract to the corresponding portion of the boundary of the disk. Therefore, $\Sigma \setminus \tau(T, e)$ retracts to $C^* \setminus (e^* \cap \tau(T, e))$. This in turn retracts to $C^* - e^*$. □

Corollary 3. *Let $e \in E(G) \setminus E(T)$. The cycle $\tau(T, e)$ is separating on Σ if and only if $C^* - e^*$ is not connected. The cycle $\tau(T, e)$ is contractible if and only if $C^* - e^*$ has a connected component that is a tree (possibly reduced to a single vertex).*

Proof. We will use the following topological property: if A is a deformation retract of X, then A and X have the same number of connected components, and A has a non-contractible loop if and only if X has a non-contractible loop.

The cycle $\tau(T, e)$ is separating if and only if $\Sigma \setminus \tau(T, e)$ is not connected; by Lemma 2, this holds if and only if $C^* - e^*$ is not connected.

$\tau(T, e)$ is contractible if and only if one component of $\Sigma \setminus \tau(T, e)$ is a disk, i.e., has no non-contractible loop. By Lemma 2, this holds if and only if one component of $C^* - e^*$ has no non-contractible loop, i.e., is a tree. □

A cycle $\tau(T, e)$ is of one of the following three topological types: contractible, non-contractible but separating, and non-separating. We can partition the edges e^* of C^* into three sets, depending on the corresponding type of $\tau(T, e)$. Figure 2 illustrates this classification.

For later use, it will be convenient to use $E_{\text{non-con}}(T)$ (resp. $E_{\text{non-sep}}(T)$) for the set of edges e in $E(G) \setminus E(T)$ such that $\tau(T, e)$ is non-contractible (resp. non-separating). As before, we use $E_{\text{non-triv}}(T)$ as an ambiguous term to refer to $E_{\text{non-con}}(T)$ or $E_{\text{non-sep}}(T)$ as needed.

Lemma 4. *The sets $E_{\text{non-con}}(T)$ and $E_{\text{non-sep}}(T)$ can be computed in $O(n)$ time.*

Proof. We construct the cut graph C^* in linear time. By Corollary 3, $E_{\text{non-con}}(T)$ can be obtained by the following procedure: starting with C^*, we repeatedly remove edges with an incident vertex of degree one; the edge set of the resulting graph is then exactly $E_{\text{non-con}}(T)^*$. To obtain $E_{\text{non-sep}}(T)$, note that, by Corollary 3, $E_{\text{non-sep}}(T)^*$ is precisely the set of non-bridge edges in C^*. The computation of the bridge edges of a graph in linear time using depth-first search is part of the folklore (see Aho et al. [1, Section 5.3] for the similar problem of determining biconnected components). □

Let s be an arbitrary vertex of G. We have the following structural result on shortest non-trivial loops based at s.

Lemma 5. *Assume T is a BFS tree from root s. Every shortest loop among the loops $\tau(T, s, e)$, where $e \in E_{\text{non-triv}}$, is a shortest non-trivial loop through s.*

Proof. A proof appears in Thomassen [26] in a more general setting. We provide an ad hoc short proof. Let ℓ be a non-trivial loop with basepoint s. We show that some non-trivial loop $\tau(T, s, e)$ is no longer than ℓ. Let $e_1, e_2, \ldots, e_k$ be the edges of ℓ, in the same order as they appear along ℓ. Since ℓ is homotopic in Σ (and therefore also homologous in $\overline{\Sigma}$) to the concatenation of loops $\tau(T, s, e_1) \cdot \tau(T, s, e_2) \cdots \tau(T, s, e_k)$, at least one of the loops $\tau(T, s, e_i)$ is non-trivial. However, $\tau(T, s, e_i)$ is a shortest loop through s that contains e_i because T is a BFS tree, whence $\tau(T, s, e_i)$ is no longer than ℓ. Furthermore, e_i cannot belong to T, for otherwise the loop $\tau(T, s, e_i)$ would be contractible (and separating). □

Theorem 6. *Let G be an unweighted graph of complexity n cellularly embedded on a surface, possibly with boundary. We can compute a shortest non-contractible or non-separating loop through a given basepoint in G in $O(n)$ time.*

Proof. We construct a BFS tree T of G from s in linear time. We attach to each vertex u of G a label $d(u)$ equal to its distance from s. We then compute $E_{\text{non-triv}}(T)$, again in linear time, using Lemma 4. Among the edges e of $E_{\text{non-triv}}(T)$, we select an edge e_0 minimizing the length of $\tau(T, s, e)$, or equivalently, minimizing the sum $d(u) + d(v)$ where u and v are the endpoints of e. Finally, we report the loop $\tau(T, s, e_0)$. The procedure takes linear time; its correctness follows from Lemma 5 and from the fact that $\tau(T, e)$ is non-trivial if and only if $\tau(T, s, e)$ is non-trivial. □

Our algorithm trivially extends to the weighted case, at the expense of a logarithmic factor, by replacing the BFS with a shortest path tree computation.

Also, the same ideas yield algorithms to compute shortest loops with an odd number of edges and shortest one-sided loops (which reverse the orientation of the surface, when it is non-orientable). Lemma 5 extends immediately to these two problems. To compute a shortest loop with an odd number of edges, it suffices to store, on each vertex of G, the parity of the number of edges to the root; then $\tau(T, s, e)$ has an odd number of edges if and only if the parities of the vertices of G incident with e are the same. To compute a shortest one-sided loop, it suffices to choose local orientations of the surface at each vertex that are consistent across each edge of T; then $\tau(T, s, e)$ is one-sided if and only if the orientations of the two vertices of G incident with e do not match across edge e. The results of the next section extend to the problem of finding shortest one-sided cycles, but do *not* extend to the problem of finding a shortest cycle with an odd number of edges.

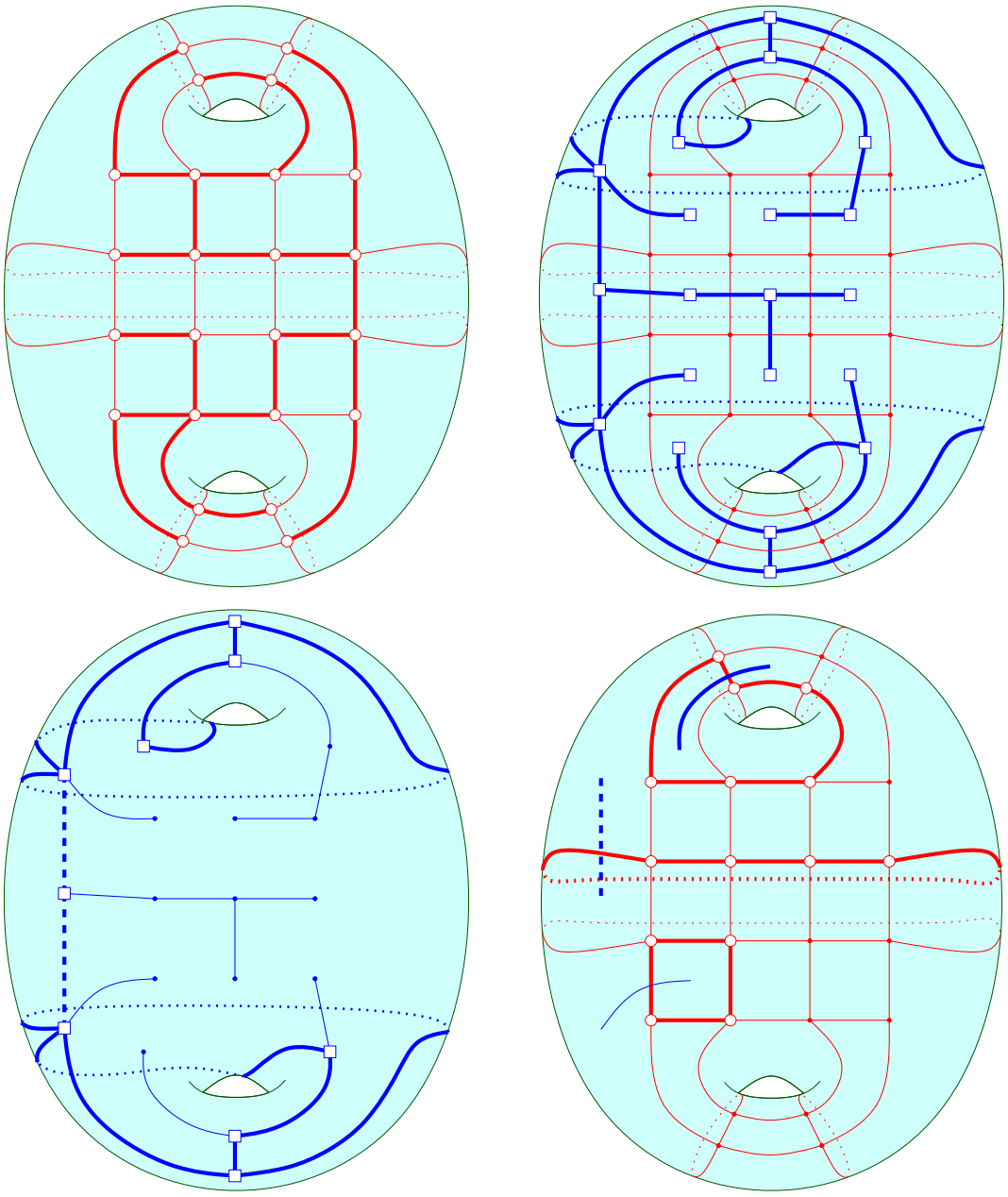

Figure 2: Top left: A graph G embedded in a double torus with a spanning tree T marked with bolder edges. Top right: The cut graph C^* is marked with bold edges; thinner edges are from the primal graph. Bottom left: classification of the edges of C^*. The bold solid edges are $E_{\text{non}-\text{sep}}(T)^*$; the bold dashed edges are $(E_{\text{non}-\text{con}}(T) \setminus E_{\text{non}-\text{sep}}(T))^*$; the thin edges are $E(C^*) \setminus E_{\text{non}-\text{con}}(T)^*$. The dotted parts of edges correspond to their hidden parts and do not provide any information about their type. Bottom right: examples of the loop $\tau(T, e)$ in bold for different types of $e^* \in C^*$.

4 Tools for Shortest Non-Trivial Cycles

4.1 Shortest Non-Trivial Loops with Remote Sources in Parallel

For a subset $X \subseteq V(G)$, we use $G[X]$ to denote the subgraph of G induced by X. The following result will be our tool to work with several sources simultaneously.

Lemma 7. *Let $V_1, \ldots, V_t$ be subsets of $V(G)$ that are pairwise disjoint, let $s_1, \ldots, s_t$ be vertices with $s_i \in V_i$ for each i, and let $\mathbb{L}_i$ be the set of non-trivial loops through s_i contained in $G[V_i]$. In $O(n)$ time we can find a shortest loop in $\bigcup_i \mathbb{L}_i$, or correctly report that $\bigcup_i \mathbb{L}_i$ is empty.*

Proof. We first describe the algorithm interlaced with an analysis of its running time, and then discuss its correctness.

For each i, we construct a BFS tree T_i with root s_i of the component of $G[V_i]$ that contains s_i. To each vertex u of G, we attach two labels, $c(u)$ and $d(u)$. The label $c(u)$ has value i if u is in the same connected component of $G[V_i]$ as s_i, and has value 0 otherwise. The label $d(u)$ is the distance between u and $s_{c(u)}$ if $c(u) \geq 1$, and undefined if $c(u) = 0$. Since the sets $V_1, \ldots, V_t$ are disjoint, the labels $c(u)$ and $d(u)$ are uniquely defined. These labels can be computed in $O(n)$ time using the BFS trees $T_1, \ldots, T_t$.

We then extend the forest $T_1, \ldots, T_t$ to a spanning tree T of G. This can also be done in linear time. Next, we compute $E_{\text{non}-\text{triv}}(T)$ using Lemma 4. Let $\tilde{E}$ be the subset of the edges $uv \in E_{\text{non}-\text{triv}}(T)$ such that $c(u) = c(v)$ and this number is non-zero. If $\tilde{E}$ is empty, we report that $\bigcup_i \mathbb{L}_i$ is empty. Otherwise, we compute

$$xy = \arg\min\{d(u) + d(v) \mid uv \in \tilde{E}\}$$

and return the loop $\tau(T, s_{c(x)}, xy)$. The construction of $E_{\text{non}-\text{triv}}(T)$ and $\tilde{E}$ takes linear time, and we spend additional constant time per edge in $\tilde{E}$ to find the edge xy. This concludes the description of the algorithm.

We now show the correctness of the algorithm. Let E_i be the set of edges uv in $E(G)$ with endpoints in T_i, i.e., such that $c(u) = c(v) = i$. Also, let $E_{\text{non}-\text{triv}}(T_i)$ be the subset of the edges e in E_i such that $\tau(T_i, s_i, e)$ is non-trivial.

By the same arguments as in the proof of Lemma 5, if $\mathbb{L}_i$ is not empty, then it contains a shortest loop of the form $\tau(T_i, s_i, e)$ for some edge e in E_i. As a consequence, a shortest non-trivial loop in $\mathbb{L}_i$ is given by $\tau(T_i, s_i, e)$ where

$$e = \arg\min\{d(u) + d(v) \mid uv \in E_{\text{non}-\text{triv}}(T_i)\}.$$

For any edge e in E_i, it holds that $\tau(T_i, s_i, e) = \tau(T, s_i, e)$ and $\tau(T_i, e) = \tau(T, e)$. It follows that

$$E_{\text{non}-\text{triv}}(T_i) = E_{\text{non}-\text{triv}}(T) \cap E_i,$$

whence $\tilde{E} = \bigcup_i E_{\text{non}-\text{triv}}(T_i)$. The correctness of the algorithm directly follows. □

4.2 Surface Decomposition

Lemma 8. *Let s be a vertex of G. In $O(n)$ time, we can compute a set K of vertices of G that intersects every non-trivial closed walk in G and is the union of the vertex sets of m shortest paths from s, where $m = 2\bar{g} + b$ for the non-contractible case and $2\bar{g}$ for the non-separating case.*

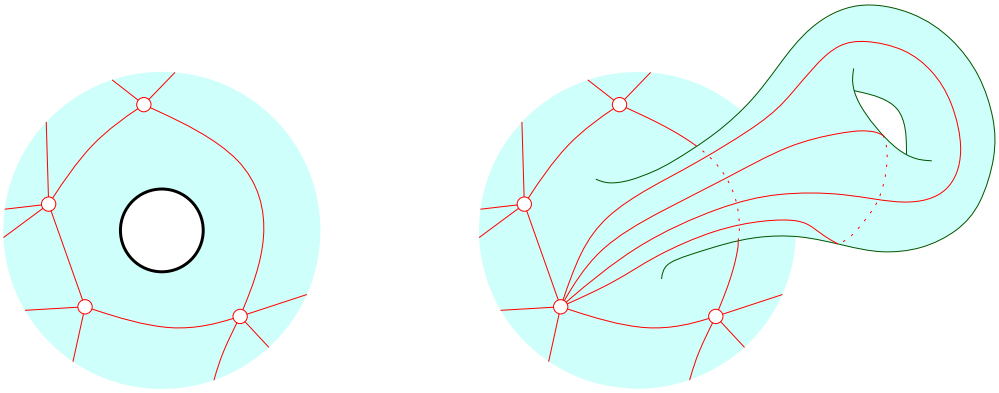

Figure 3: Detail of the graph $\hat{G}$ cellularly embedded in $\hat{\Sigma}$, as constructed in the proof of Lemma 8. We attach a handle to each boundary component and add two loop edges to obtain a cellular embedding.

Proof. Assume first that Σ has no boundary (this is the only relevant situation in the non-separating case). We use the tree-cotree decomposition of Eppstein [10]. Consider a BFS tree T from the vertex s. Let $(T')^*$ be an arbitrary spanning tree of $G^* - E(T)^*$. Thus T and T' are edge-disjoint in G; we let X be the remaining edges of G. It follows from Euler's formula that X has $\bar{g}$ edges.

Consider the set of loops $\mathbb{L} = \{\tau(T, s, e) \mid e \in X\}$: their union $\bigcup \mathbb{L}$ is a cut graph. Indeed, $\Sigma \backslash\backslash (T \cup X)$ is a set of faces connected according to the dual tree T', hence a disk. But $\bigcup \mathbb{L}$ is obtained from $T \cup X$ by iteratively removing a degree one vertex with its incident edge, and this operation preserves the fact that the complement is a disk. Let K be the set of vertices in $\mathbb{L}$. Since $\bigcup \mathbb{L}$ is a cut graph, each non-trivial closed walk must intersect K. Moreover, since T is a BFS tree, each of the $\bar{g}$ loops $\tau(T, s, e)$ in $\mathbb{L}$ consists of two shortest paths from s.

K can be computed in linear time, as we describe next. Computing a BFS tree T from s, the spanning tree in the dual graph, and computing the set of edges X takes linear time. We mark in T the vertex s and the endpoints of the edges in X. By definition of the loops in $\mathbb{L}$, the set K is the set of vertices of the minimal subtree of T that includes the marked vertices. This subtree is obtained in linear time by recursively removing unmarked degree one vertices.

It remains to consider the case where we want to compute a shortest non-contractible cycle on a surface Σ with boundary. In this case we attach a handle to every boundary component of Σ, obtaining a surface $\hat{\Sigma}$ without boundary. To make G cellular on $\hat{\Sigma}$, we just have to add two loop edges per boundary component of Σ; see Figure 3. Let $\hat{G}$ be the new graph. Then we apply the previous construction to this new graph. We obtain a set $\hat{K}$ that intersects every cycle of $\hat{G}$ that is non-trivial on $\hat{\Sigma}$; furthermore, $\hat{K}$ is in the union of $2\bar{g} + b$ shortest paths from the source s. (The two loop edges e_1 and e_2 defining a handle contribute to a single shortest path to K, namely, the shortest path from s to the common endpoint of e_1 and e_2.) Hence $\hat{K}$ intersects also every cycle in G that is non-trivial in $\hat{\Sigma}$. Furthermore, the shortest paths from s in G and $\hat{G}$ are the same, because we only added loop edges, so $\hat{K}$ lies on $2\bar{g}+b$ shortest paths from s in G. To conclude, note that a cycle is contractible in $\hat{\Sigma}$ if and only if it is contractible in Σ. So we can take $K = \hat{K}$. □

4.3 2-Approximation of the Edge-Width

Erickson and Har-Peled have proposed an algorithm to compute a 2-approximation of the edge-width [11, Corollary 5.8]. They describe it in the case of weighted graphs; in our setting, we can

shave off a logarithmic factor using Theorem 6. We sketch the proof for convenience, and also because most of the tools have already been presented.

Proposition 9. *We can compute a non-trivial cycle whose length is at most twice the length of a shortest non-trivial cycle, in $O((g+b)n)$ time for the non-contractible case and $O(gn)$ time for the non-separating case.*

Proof. Let π be a shortest path in G. Erickson and Har-Peled [11, Lemma 5.6] give an algorithm that computes in $O(n \log n)$ time a non-trivial loop intersecting π whose length is at most twice the length of the shortest non-trivial loop intersecting π. Their algorithm is the following: (1) contract π to a single point; (2) compute the shortest non-trivial loop ℓ through that point; (3) expand back π, and return the loop resulting from ℓ after the expansion. Since π is a shortest path, the expansion can at most double the length of the loop, which is therefore a 2-approximation of the shortest non-trivial loop intersecting π. Since our graph is unweighted, we can use Theorem 6 in step (2), and therefore the algorithm runs in $O(n)$ time.

By Lemma 8, we can compute a set of $O(g+b)$ or $O(g)$ shortest paths intersecting every non-trivial loop. We can apply the previous algorithm to each of these shortest paths to obtain a non-trivial loop ℓ' whose length is at most twice the length of a shortest non-trivial cycle. By construction ℓ' is either a cycle or a loop composed of a starting path concatenated with a cycle and the reverse of the starting path. In the latter case, we simply remove the starting and ending paths to get a non-trivial cycle. □

5 Main Results

5.1 Decision and Output-Sensitive Algorithm

Theorem 10. *Let G be an unweighted graph of complexity n cellularly embedded on a surface of genus g with b boundaries. Given an integer k, we can decide in $O((g+b)nk)$ time (resp. $O(gnk)$ time) if the edge-width (resp. non-separating edge-width) of G is at most k. If it is the case, we can also obtain a shortest non-contractible (resp. non-separating) cycle in G.*

Proof. We start by computing the set K of vertices as in Lemma 8. It then suffices to compute a shortest non-trivial loop based at some vertex in K, or to determine that every such loop has length larger than k.

Let S_i be the set of vertices in K at distance exactly i from s ($0 \leq i \leq n$). Each S_i has cardinality at most $2\bar{g}+b$ (non-contractible case) or $2\bar{g}$ (non-separating case). For simplicity, in the remaining part of the proof, we only consider the non-contractible case; the non-separating case is the same, except that we can replace $2\bar{g}+b$ by $2\bar{g}$.

For each j, $0 \leq j \leq k$, we put the elements of

$$S_j, S_{(k+1)+j}, S_{2(k+1)+j}, \dots$$

into at most $2\bar{g}+b$ batches, each containing at most one element from each of these S_i. In total, we have a partition of K into $O((g+b)k)$ batches such that any two vertices in the same batch are at distance at least $k+1$ from each other (because an element in S_i and an element in S_j are at distance at least $|i-j|$ from each other by the triangle inequality).

Now, consider a single batch $\{s_1, \dots, s_t\}$. For each i, let V_i be the set of vertices at distance at most $k/2$ from s_i; the V_i's are pairwise disjoint. We can thus apply Lemma 7: if there exists a non-trivial loop based at some s_i that has length at most k, we obtain the shortest such loop.

We apply this operation for every batch; thus we computed, for each vertex of K, the shortest loop based at that vertex, unless that loop has length larger than k. If the shortest of the resulting loops has length at most k, this is the output of the algorithm. Otherwise (or if no loop has been found), we report that no non-trivial loop with length at most k exists.

The set K and the $O((g+b)k)$ batches can be computed in $O(n)$ time. Then the $O(n)$ time algorithm of Lemma 7 is applied once for each of the $O((g+b)k)$ batches; thus the total running time is $O((g+b)nk)$. □

The proof of our output-sensitive algorithm is now easy.

Corollary 11. *Let k_0 be the edge-width (resp. non-separating edge-width) of G. We can compute a shortest non-contractible (resp. non-separating) cycle in G in $O((g+b)nk_0)$ (resp. $O(gnk_0)$) time.*

Proof. The edge-width can be computed applying Theorem 10 with $k = 2^0, 2^1, \ldots, 2^i, \ldots$ until a non-contractible cycle is found. The total cost is

$$O((g+b)n(2^{\lceil \log k_0 \rceil + 1})) = O((g+b)nk_0).$$

(Alternatively, we may use Proposition 9 to compute a 2-approximation of the edge-width, and then apply Theorem 10 only once.) The same arguments, with $g+b$ replaced by g, yield the result for the non-separating edge-width. □

5.2 Approximation Algorithm

Finally, we can obtain the following approximation algorithm.

Theorem 12. *Let G be an unweighted graph of complexity n cellularly embedded on a surface of genus g with b boundaries, and let ε be a parameter such that $0 < \varepsilon < 1$. We can compute a non-contractible (resp. non-separating) cycle in G whose length is at most $1+\varepsilon$ times the length of the shortest such cycle, in $O((g+b)n/\varepsilon)$ (resp. $O(gn/\varepsilon)$) time. In particular, we approximate the (possibly non-separating) edge-width of G up to a factor of ε.*

Proof. Again, we only consider the non-contractible case; the non-separating case is handled in the same way. Let k_0 be the edge-width of G. Using Proposition 9, we compute in $O((g+b)n)$ time an integer k' such that $k_0 \leq k' \leq 2k_0$.

Let K be the set of vertices of G obtained by applying Lemma 8; in particular, K intersects every non-trivial cycle. For $i \geq 0$, let S_i be the set of vertices of K at distance i from the fixed vertex s. We put $S'_i = S_{i \cdot \lceil k'\varepsilon/4 \rceil}$ and consider the subset $K' = \cup_i S'_i$ of K.

We claim that some non-trivial loop of length at most $(1+\varepsilon)k_0$ must intersect K'. Indeed, let ℓ be a shortest non-trivial cycle; ℓ contains a vertex v of K. From the construction of K, the vertices of the shortest path π from v to s are in K. This shortest path contains a vertex w in K' at distance at most $k'\varepsilon/4 \leq k_0\varepsilon/2$ from v. Using the subpath of π between w and v as an approach path to ℓ, we obtain a non-trivial loop of length at most $(1+\varepsilon)k_0$ through w. This proves the claim.

Recall that, as a consequence of Lemma 8, every set S'_i contains at most $2\bar{g}+b$ elements. Let $t = \left\lceil \frac{(1+\varepsilon)(k'+1)}{\lceil k'\varepsilon/4 \rceil} \right\rceil = O(1/\varepsilon)$. For each j, $0 \leq j < t$, we put the elements of

$$S'_j, S'_{t+j}, S'_{2t+j}, \ldots$$

into at most $2\bar{g}+b$ batches, each containing at most one element from each of these S'_i. In total, we have a partition of K' into $O((g+b)/\varepsilon)$ batches such that any two vertices in the same batch

are at distance at least $t\lceil k'\varepsilon/4\rceil \geq (1+\varepsilon)(k'+1)$ from each other (because an element in S_i and an element in S_j are at distance at least $|i-j|$ from each other by the triangle inequality). Within each batch, we apply Lemma 7, where for each source s_i in the batch we take V_i as the set of vertices at distance at most $(1+\varepsilon)k'/2$ from s_i. In $O(n)$ time, we find a shortest non-trivial loop of length at most $(1+\varepsilon)k'$ through a given vertex in the batch, or determine that no such loop exists.

We do this for each batch. This takes $O((g+b)n/\varepsilon)$ time in total. Finally, we return the shortest loop between the shortest non-trivial loops found over all batches. We know by the above claim that the computation will find in some batch a non-trivial loop of length at most $(1+\varepsilon)k_0$. As at the end of the proof of Proposition 9, it remains to extract a cycle from this loop, if needed. □

Acknowledgments

We thank Jeff Erickson for pointing out reference [15] and its implications, and Luca Castelli Aleardi for an inspiring discussion.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer programs*. Addison-Wesley, 1974.

[2] M. O. Albertson and J. P. Hutchinson. The independence ratio and genus of a graph. *Transactions of the American Mathematical Society*, 226:161–173, 1977.

[3] S. Cabello and E. W. Chambers. Multiple source shortest paths in a genus g graph. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 89–97, 2007.

[4] S. Cabello, É. Colin de Verdière, and F. Lazarus. Finding shortest non-trivial cycles in directed graphs on surfaces. In *Proceedings of the 26th Annual ACM Symposium on Computational Geometry (SOCG)*, pages 156–165, 2010.

[5] S. Cabello, M. DeVos, J. Erickson, and B. Mohar. Finding one tight cycle. *ACM Transactions on Algorithms*, 6(4), Art. no. 61, 2010.

[6] S. Cabello and B. Mohar. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. *Discrete & Computational Geometry*, 37(2):213–235, 2007.

[7] É. Colin de Verdière. Algorithms for graphs on surfaces. Course notes, available at `http://www.di.ens.fr/~colin/cours/algo-graphs-surfaces.pdf`, 2008.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[9] M. DeVos, K.-i. Kawarabayashi, and B. Mohar. Locally planar graphs are 5-choosable. *J. Comb. Theory Ser. B*, 98(6):1215–1232, 2008.

[10] D. Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 599–608, 2003.

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

[12] J. Erickson and K. Whittlesey. Greedy optimal homotopy and homology generators. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1038–1046, 2005.

[13] J. R. Fiedler, J. P. Huneke, R. B. Richter, and N. Robertson. Computing the orientable genus of projective graphs. *J. Graph Theory*, 20(3):297–308, 1995.

[14] A. Hatcher. *Algebraic topology*. Cambridge University Press, 2002. Available at `http://www.math.cornell.edu/~hatcher/`.

[15] J. P. Hutchinson. On short noncontractible cycles in embedded graphs. *SIAM Journal on Discrete Mathematics*, 1(2):185–192, 1988.

[16] K.-i. Kawarabayashi and B. Mohar. Graph and map isomorphism and all polyhedral embeddings in linear time. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 471–480, 2008.

[17] K.-i. Kawarabayashi and B. Reed. Computing crossing number in linear time. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 382–390, 2007.

[18] Y. Kobayashi and K.-i. Kawarabayashi. Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1146–1155, 2009.

[19] M. Kutz. Computing shortest non-trivial cycles on orientable surfaces of bounded genus in almost linear time. In *Proceedings of the 22nd Annual ACM Symposium on Computational Geometry (SOCG)*, pages 430–438, 2006.

[20] W. S. Massey. *Algebraic Topology: An Introduction*, volume 56 of *Graduate Texts in Mathematics*. Springer-Verlag, 1977.

[21] B. Mohar and N. Robertson. Flexibility of polyhedral embeddings of graphs in surfaces. *J. Comb. Theory Ser. B*, 83(1):38–57, 2001.

[22] B. Mohar and C. Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2001.

[23] N. Robertson and P. D. Seymour. Graph minors. VII. Disjoint paths on a surface. *Journal of Combinatorial Theory, Series B*, 45:212–254, 1988.

[24] N. Robertson and R. P. Vitray. Representativity of surface embeddings. In B. Korte, L. Lovász, and Prömel, editors, *Paths, flows, and VLSI-layout*, pages 293–328. Springer-Verlag, Berlin, 1990.

[25] J. Stillwell. *Classical topology and combinatorial group theory*. Springer-Verlag, New York, 1980.

[26] C. Thomassen. Embeddings of graphs with no short noncontractible cycles. *Journal of Combinatorial Theory, Series B*, 48(2):155–177, 1990.

[27] C. Thomassen. Five-coloring maps on surfaces. *J. Comb. Theory Ser. B*, 59(1):89–105, 1993.

[28] X. Yu. Disjoint paths, planarizing cycles, and spanning walks. *Transactions of the American Mathematical Society*, 349:1333–1358, 1997.