Short and Simple Cycle Separators in Planar Graphs

Eli Fox-Epstein[*] Shay Mozes [†] Phitchaya Mangpo Phothilimthana [‡]
Christian Sommer [§]

Abstract

We provide an implementation of an algorithm that, given a triangulated planar graph with m edges, returns a simple cycle that is a 2/3–balanced separator consisting of at most $\sqrt{8m}$ edges. An efficient construction of a short and balanced separator that forms a simple cycle is essential in numerous planar graph algorithms, e.g., for computing shortest paths, minimum cuts, or maximum flows. To the best of our knowledge, this is the first implementation of such a cycle separator algorithm with a worst-case guarantee on the cycle length.

We evaluate the performance of our algorithm and compare it to the planar separator algorithms recently studied by Holzer et al. [ESA 2005, ACM Journal of Experimental Algorithms 2009]. Out of these algorithms, only the Fundamental Cycle Separator (FCS) produces a simple cycle separator. However, FCS does not provide a worst-case size guarantee. We demonstrate that *(i)* our algorithm is competitive across all test cases in terms of running time, balance and cycle length, *(ii)* it provides worst-case guarantees on the cycle length, significantly outperforming FCS on some instances, and *(iii)* it scales to large graphs.

1 Introduction

Separators identify structure in a graph by cleaving it into two balanced parts with little mutual interference. A separator theorem typically provides worst-case guarantees on the balance of the parts and on the size of their shared boundary. Separators have been studied extensively and separator theorems have been found for planar graphs [37, 29, 10, 31, 16, 36, 12], bounded-genus graphs [11, 17, 24], minor-free graphs [3, 34, 35, 4, 23, 40], and others.

Efficient algorithms for these graph classes often exploit the fact that the input graph has *small* separators. For example, divide-and-conquer algorithms rely on decomposing a problem into subproblems with little interference. More formally, a separator of a graph $G = (V, E)$ is a subset $S \subseteq V$ that partitions $V \setminus S$ into two sets $A, B \subseteq V$ of approximately equal size[1] (say $|A|, |B| \leq 2|V|/3$) and no edges between the two parts ($E \cap A \times B = \emptyset$). A smaller set S often implies faster algorithms providing solutions for various problems such as exact shortest paths [15, 19] or approximate vertex cover [30], and many more [18, 27, 13, 33, 6, 22, 32, 21, 28, 5]. For planar graphs, it is known that $|S| = O(\sqrt{|V|})$ is possible, even for worst-case instances [29].

Separators in planar graphs are based on fundamental cycles. For a spanning tree T, the *fundamental cycle* C_{uv} induced by an edge $uv \notin T$ is the simple cycle formed by uv and the u-to-v path in T. Every fundamental cycle C separates the graph into two parts: the subgraph enclosed by C and the subgraph not enclosed by C. An elementary argument shows that, if the graph is triangulated, then there always exists an edge $e \notin T$ such that C_e is a balanced separator in G. Namely, each of the two parts consists of at most $2|V|/3$ vertices. A key observation is that, starting with a breadth-first search tree, the size of any fundamental cycle is at most one plus the diameter of the graph. Therefore, if the diameter is small, the simple *Fundamental Cycle Separator Algorithm* works well: arbitrarily select a root for a breadth-first search, compute a BFS tree, and then return the *best* fundamental cycle (*best* may be defined in terms of balance, length, or both). However, if the diameter is large, any balanced fundamental cycle may be long, and, as a consequence, the separator may be large as well.

In order to provide separators with small worst-case sizes, most separator algorithms first reduce the diameter of the input graph and then use the *Fundamental Cycle Separator Algorithm* as a subroutine. The seminal *Planar Separator Algorithm* of Lipton and Tarjan [29] (henceforth referred to as Lipton-Tarjan) finds a 2/3–balanced separator by identifying two sets S_1, S_2 of

[*]Tufts University, `ef@cs.tufts.edu`

[†]MIT, `shaym@mit.edu`; part of this work was conducted while SM was with Brown University.

[‡]MIT, `mangpo@csail.mit.edu`

[§]`csom@csail.mit.edu`

[1]More generally, one can define separators to be balanced with respect to a weight function on the vertices, edges, and, in embedded graphs, faces.

small size $|S_1|, |S_2| = O(\sqrt{|V|})$, whose removal yields a subgraph with diameter $O(\sqrt{|V|})$ and large enough weight. One can interpret the diameter reduction as shortcutting a fundamental cycle using the vertices of S_1 and S_2.

For many planar graph algorithms such as those computing shortest paths [27, 13, 33, 6, 32], minimum cuts [21, 28], maximum flows [5], it is crucial that the separator S forms a (simple) *cycle* in G. Unfortunately, adding vertices of the sets S_1, S_2 to shortcut a fundamental cycle, as in the Lipton-Tarjan algorithm, results in a separator that does not necessarily form a simple cycle. Neither the Lipton-Tarjan algorithm [29] nor Djidjev's algorithm [10] obtain simple cycles, and the FCS algorithm does not provide any worst-case guarantees on the cycle length.

The algorithms of Miller [31], Gazit and Miller [16], and Djidjev and Venkatesan [12] offer both guarantees: for any triangulated 2–connected planar graph, they can compute a simple cycle separator of length $O(\sqrt{|V|})$ in linear time. In this work, we focus on cycle separator algorithms for planar graphs.

1.1 Related Experimental Work. There is a large body of experimental work on graph partitioning, mostly implementing various heuristics. In this work, we shall focus on algorithms with worst-case guarantees. Theoretical results on separators suggest that they can be used to substantially speed up algorithms. Consequently, Lipton-Tarjan separators and variants have been implemented and evaluated experimentally [14, 2, 20].

Farrag [14] implemented an algorithm of Aleksandrov and Djidjev [1], where, instead of separating V into two sets A, B, the number of pieces can be specified. The separator algorithm is used for load balancing of parallel algorithms: the input graph is partitioned into k pieces, distributing the work evenly among k processors.

Aleksandrov, Djidjev, Guo, and Maheshwari [2] implemented a three-phase algorithm, which *(i)* partitions the graph *by levels*, *(ii)* partitions the graph *by fundamental cycles*, and *(iii)* combines the resulting components into the right number of pieces (*packing*).

The most recent experimental work, and the one most relevant to compare with our work is by Holzer, Schulz, Wagner, Prasinos, and Zaroliagis [20], who implemented the Lipton-Tarjan algorithm [29] and Djidjev's algorithm [10], and provided an extensive experimental evaluation. One of the main findings of their study is that, across their battery of test graphs, the FCS algorithm performs at least as well as their more carefully engineered counterparts, despite the lack of worst-case guarantees.

To the best of our knowledge, cycle separator algorithms with worst-case guarantees on the cycle length have not been implemented and evaluated yet.

1.2 Contributions. We provide an implementation of a cycle separator algorithm with a worst-case guaranteed size of $\sqrt{8|E|}$. This algorithm is a variant of the one recently described in [26], and in Klein's forthcoming book [25]. We experimentally evaluate our algorithm and compare it to the Fundamental Cycle Separator (FCS) Algorithm. As mentioned in Section 1.1, the experimental results of Holzer et al. [20] suggest that the FCS algorithm works well for most inputs. We confirm their findings. However, we also identify classes of graphs for which the FCS algorithm returns arbitrarily long cycles.[2] We demonstrate that *(i)* our algorithm is competitive with respect to FCS for the graphs in [20], *(ii)* it provides worst-case guarantees on the cycle length, significantly outperforming FCS on some instances, and *(iii)* it scales to large graphs.

We are hopeful that this implementation will pave the way to implementing the many theoretically efficient algorithms that rely on simple cycle separators in planar graphs.

2 Preliminaries

For a tree T and an edge $e \in T$, let T_e denote the subtree of T rooted at the leafward endpoint of e. A *breadth-first search* (BFS) yields a tree, which is the subgraph of the input with the same vertices and exactly the edges traversed in the search. One can *seed* a BFS with a number of vertices or edges. To do this, imagine the search started from a super-vertex connected to all of the seeds. Doing so may yield a forest instead of a tree.

For a rooted tree T, and a set S of nodes of T, a *leafmost* node in S is a node s of S with the property that s does not lie on the root-to-s' path in T for any other node $s' \in S$. Similarly, a node s is *rootmost* if there is no other node $s' \in S$ that lies on the s-to-root path in T.

For a spanning tree T of G and an edge e of G not in T, the *fundamental cycle* of e with respect to T in G is the simple cycle consisting of e and the unique simple path in T between the endpoints of e.

We provide a brief review of basic definitions and facts related to planar graphs, combinatorial embeddings and planar duality. For elaboration, see also [25].

[2]In the hard instances for FCS, the length of the cycle heavily depends on the choice of the root of the BFS tree. For some roots, the fundamental cycles are short, while for other roots the cycles are very long.

2.1 Embeddings and Planar Graphs. Let E be a finite set, the edge-set. We define the corresponding set of *darts* to be $E \times \{\pm 1\}$. For $e \in E$, the darts of e are $(e, +1)$ and $(e, -1)$. We think of the darts of e as oriented versions of e (one for each orientation). We define the involution $\text{rev}(\cdot)$ by $\text{rev}((e, i)) = (e, -i)$. That is, $\text{rev}(d)$ is the dart with the same edge but the opposite orientation.

A graph on E is defined to be a pair (V, E) where V is a partition of the darts. Thus each element of V is a nonempty subset of darts. We refer to the elements of V as *vertices*. The endpoints of an edge e are the subsets $v, v' \in V$ containing the darts of e. The *head* of a dart d is the subset $v \in V$ containing d, and its *tail* is the head of $\text{rev}(d)$.

An *embedding* of (V, E) is a permutation π of the darts such that V is the set of orbits of π. For each orbit v, the restriction of π to that orbit is a permutation cycle. The permutation cycle for v specifies how the darts with head v are arranged around v in the embedding (in, say, counterclockwise order). We refer to the pair (π, E) as an embedded graph.

Let π^* denote the permutation $\text{rev} \circ \pi$, where $\circ$ denotes functional composition. Then (π^*, E) is another embedded graph, the *dual* of (π, E). (In this context, we refer to (π, E) as the *primal.*)

The *faces* of (π, E) are defined to be the vertices of (π^*, E). Since $\text{rev} \circ (\text{rev} \circ \pi) = \pi$, the dual of the dual of (π, E) is (π, E). Therefore the faces of (π^*, E) are the vertices of (π, E). Throughout the paper we denote the primal graph by G and its dual by G^*.

We define an embedded graph (π, E) to be *planar* if $n - m + \phi = 2\kappa$, where n is the number of vertices, m is the number of edges, ϕ is the number of faces, and κ is the number of connected components. Since taking the dual swaps vertices and faces and preserves the number of connected components, the dual of a planar embedded graph is also planar.

Note that, according to our notation, we can use e to refer to an edge in the primal or the dual.

Combinatorial embeddings are useful in practice as well. In our implementation, embedded planar graphs are represented as permutations on their darts. Darts and vertices are represented by integer values. Graphs are efficiently traversed by retaining lookup tables allowing one to walk around permutations and find incident darts and vertices. In our implementation, each graph takes $4|E| + |V| + |F|$ integers for the primal and dual representations, which is sufficiently compact as to accommodate graphs with millions of vertices with commodity hardware.

To provide more intuition we use geometric embeddings of planar graphs in the figures in this paper. Both the primal and the dual graphs are embedded on the same plane, so their edges are in one-to-one correspondence. The edges of the dual graph in the figures are rotated by roughly 90 degrees clockwise with respect to their primal counterparts.

We focus on undirected embedded planar graphs with no self-loops or parallel edges. For a graph G, we use $V(G), E(G), F(G)$ to denote the vertices, edges, and faces of G, respectively.

We use the following properties of planar graphs:

FACT 2.1. (SIMPLE-CYCLE/SIMPLE-CUT DUALITY [39]) *A set of edges forms a simple cycle in a planar embedded graph G iff it forms a simple cut in the dual G^*.*

Since a simple cut in a graph uniquely determines a bipartition of the vertices of the graph, a simple cycle in a planar embedded graph G uniquely determines a bipartition of the faces.

DEFINITION 1. (ENCLOSES) *Let C be a simple cycle in a connected planar embedded graph G. Then the edges of C form a simple cut $\delta_{G^*}(S)$ for some set S of vertices of G^*, i.e. faces of G. Thus C uniquely determines a bipartition $\{F_0, F_1\}$ of the faces of G. Let f_∞, f be faces of G. We say C* encloses *f with respect to f_∞ if exactly one of f, f_∞ is in S. For a vertex/edge x, we say C* encloses *x (with respect to f_∞) if it encloses some face incident to x (encloses* strictly *if in addition x is not part of C).*

FACT 2.2. ([38]) *For any spanning tree T of G, the set of edges of G not in T form a spanning tree of G^*.*

For a spanning tree T of G, we typically use T^* to denote the spanning tree of G^* consisting of the edges not in T.

2.2 Cycle Separators in Planar Graphs. We define an α–balanced separator to be a tripartition of the vertices of the graph into (A, B, S) that is

separated: there are no edges from any node in A to any node in B

balanced: $|A|$ and $|B|$ are each at most $\alpha\,|V(G)|$

small: $|S| \leq f(|E(G)|)$ for some f

Typically, as well as throughout this paper, $f(m) = O(\sqrt{m})$ and $\alpha = 2/3$.

Let G be a triangulated biconnected simple planar graph. Any simple cycle C in G separates G into the interior of C (the subgraph enclosed by C) and the exterior of C (the subgraph not strictly enclosed by C), such that there are no edges between vertices strictly enclosed by C and vertices not enclosed by C. We

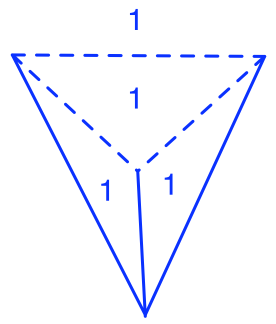


Figure 1: A triangulated graph with unit face weights and a spanning tree (solid edges) for which no fundamental cycle is 2/3–balanced. In this example all fundamental cycles are 3/4–balanced. In fact, this exemplifies the worst case. That is, a balance of 3/4 is always achievable, provided no single face accounts for more than 3/4 the total weight.

call a simple cycle C a small balanced cycle separator in G if the separator defined by the interior of C, the exterior of C, and C itself is 2/3–balanced and $|C| = O(f(|E(G)|))$ for some f.

We note that balance can be defined in more general terms than number of vertices. Let w be a function assigning real weights to vertices, edges, and faces of G. A cycle separator C is balanced with respect to the weight function w if the total weight of vertices, edges, and faces strictly enclosed (respectively, not enclosed) by C is at most 2/3 the total weight of G. General weights can be handled by considering just face weights: arbitrarily assign the weight of any vertex or edge to an incident face. Any cycle separator that is balanced with respect to the new weight assignment is necessarily balanced with respect to the original one. Therefore, without loss of generality, we only refer to face weights in this paper.

For triangulated planar graphs with general face weights there may not exist a 2/3–balanced fundamental cycle separator, even if the weight of any single face is at most 2/3 the total weight. This is illustrated in Figure 1. It is not difficult to see that one can always achieve a balance of 3/4. However, if the weight of each face is negligible with respect to the total weight (as is the case when separating according to just the number of vertices in a graph with many vertices), this is not a problem and a balance of almost 2/3 is achievable. For the sake of generality, we use a balance of 3/4 in our proofs. Experimentally, since the balance criterion we use is the number of vertices, we always observe a balance of at most 2/3.

If the input graph is not triangulated, one can always add edges to triangulate it. In this case the cycle separator does not necessarily form a cycle in the input graph. However, topologically, the separator does form a cycle. For some applications such a topological separation suffices, while in others it is possible to retain the additional edges without affecting the application.

3 The Cycle Separator Algorithm

In this section we describe our simple cycle separator algorithm. It roughly follows the overall structure of Miller's algorithm [31], but is significantly different. The algorithm is similar to the one suggested recently in [26], also described in Klein's forthcoming book [25].

3.1 Levels and Level Components. We define levels with respect to an arbitrarily chosen face f_∞, which we designate as the infinite face.

Definition 2. *The* level $\ell^F(f)$ *of a face f is the minimum number of edges on a f_∞–to–f–path in the dual G^* of G. We use L_i^F to denote the faces having level i, and we use $L_{\geqslant i}^F$ denote the set of faces f having level at least i.*

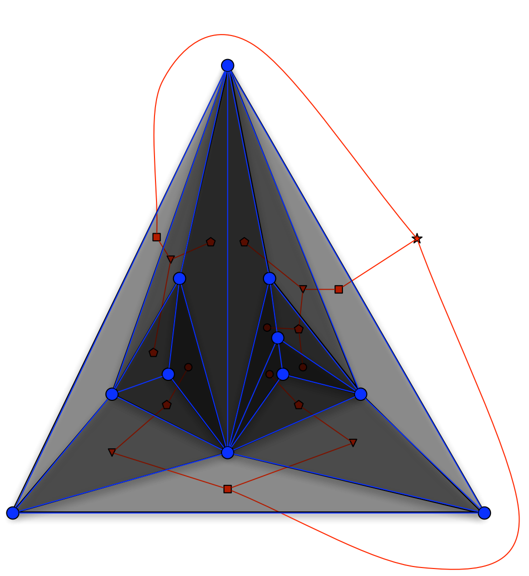
Definition 3. *For an integer $i \geq 0$, a connected component of the subgraph of G^* induced by $L_{\geqslant i}^F$ is called a* level-i component*, or, if we do not want to specify i, a* level component*. We use $\mathcal{K}_{\geqslant i}$ to denote the set of level-i components. A level-i component K is said to have level i, and we denote its level by $\ell^{\mathcal{K}}(K)$. A* nonroot *level component is a level component whose level is not zero. The set of vertices of G^* (faces of G) belonging to K is denoted $F(K)$.*

Note that we use K (not K^*) to denote a level component even though it is a connected component of a subgraph of the planar dual.

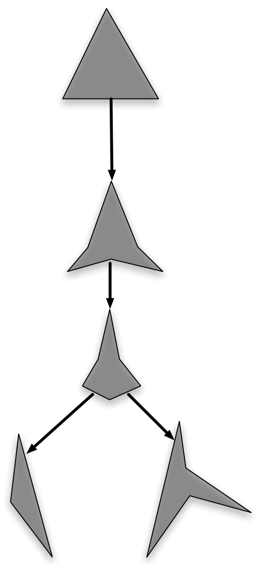
Fact 3.1. *For any nonroot level component, the subgraph of G^* consisting of faces not in $F(K)$ is connected.*

Corollary 3.1. *For any nonroot level component K, the edges crossing the cut $(F(K), G^* \setminus F(K))$ (i.e., the set of edges in G^* with exactly one endpoint in $F(K)$) form a simple cycle in the primal G.*

In view of Corollary 3.1, for any nonroot level component K, we use $X(K)$ to denote the simple cycle in the primal G consisting of the edges of the cut $(F(K), G^* \setminus F(K))$. We refer to $X(K)$ as the *level cycle* bounding K.



(a) A triangulated graph (blue vertices and edges) along with a dual BFS tree (in red, faces at different BFS levels are indicated by different shapes). The (primal) components $\mathcal{K}_{\geqslant k}$ are indicated as shaded subgraphs. The deeper the level of a component, the darker its shade is.



(b) The corresponding component tree $\mathcal{K}$. Each node of the component tree corresponds to a connected component of G^*.

Figure 2: Illustration of the component tree $\mathcal{K}$.

Definition 4. *The* component tree $\mathcal{K}$ *is the rooted tree whose nodes are the level components, and in which K is an ancestor of K' if the faces of K include the faces of K'.*

The root of the component tree is the unique level-0 component consisting of all of G^*. Figure 2 illustrates the definition of the component tree.

Definition 5. *An edge ff' of G^* has level i if f has level i and f' has level $i+1$. We write $\ell^E(ff')$ for the level of ff'. We use L_i^E to denote the set of edges of level i.*

Note that not every edge of G^* has a level.

3.2 Description of the Algorithm.

Pseudocode of the algorithm is given as Algorithm 1. An overview of our algorithm is as follows. Let W be the sum of the weights of all the faces.

The algorithm starts by computing the component tree $\mathcal{K}$. Next, the algorithm finds a small range of levels, where a short balanced cycle separator is guaranteed to exist (see appendix). This is done by identifying the leafmost component K' whose weight is at least half the total weight. Let i_0 denote the level of K'. The algorithm finds two levels, i_- and i_+, each with few (at most $\sqrt{m/2}$) edges, that bracket i_0 (lines 5, 7). There is a single component K_0 at level i_- that contains K'. We denote the level-i_+ components contained in K_0 by $K_1, K_2, \ldots$. Let H be the subgraph of G induced by the faces in $K_0 \setminus \bigcup_{j\geq 1} K_j$. The edges of H at level i_- are on a single face of H. This is the bounding cycle $X(K_0)$ of the component K_0 at level i_- that contains K'. Similarly, the edges of H at level i_+ are on a set of faces that correspond to the bounding cycles $\{X(K_j)\}_{j\geq 1}$ of the components $\{K_j\}_{j\geq 1}$. If any $X(K_i)$ happens to be a good separator, it is returned (line 9).

Otherwise, roughly speaking, the algorithm finds a balanced fundamental cycle in K_0 and shortcuts it along one of the cycles $X(K_j)$. Care must be taken to ensure that the resulting cycle is simple. This is ensured by an appropriate choice of spanning tree, and by appropriately defining how to shortcut the fundamental cycle along $X(K_j)$.

More precisely, the algorithm initializes a forest F with all the edges of $X(K_0)$ except an arbitrary one. It iteratively adds to F edges of the cycles $X(K_j)$ ($j \geq 1$) that do not introduce cycles (line 13). In line 14 it further extends F into a spanning tree T of G by performing a breadth-first search starting from the component of F that contains the edges of $X(K_0)$. By

Algorithm 1: Cycle Separator Algorithm

```
1  triangulate $G$ and choose $f_\infty$ arbitrarily
2  construct the component tree $\mathcal{K}$
3  let $K'$ be the maximum-level component with weight at least $W/2$
4  let $i_0$ be the level of $K'$
5  let $i_-$ be the maximum level not exceeding $i_0$ such that $|L^E_{i_-}| \le \sqrt{m/2}$
6  let $K_0$ be the component at level $i_-$ that contains $K'$
7  let $i_+$ be the minimum level no less than $i_0$ such that $|L^E_{i_+}| \le \sqrt{m/2}$
8  let $K_1, K_2, \ldots$ be the components at level $i_+$ contained in $K_0$
9  if $W/4 \le w(K_j) \le 3W/4$ for any $j \ge 0$ then return $X(K_j)$
10 initialize a forest $F$ with all the edges of $X(K_0)$ except for an arbitrary one
11 foreach cycle $X(K_j)$ $(j \ge 1)$ do
12     foreach edge $e$ of $X(K_j)$ do
13         add $e$ to $F$ if it does not introduce a cycle in $F$
14 extend $F$ into a spanning tree $T$ of $G$ by a breadth-first search, starting from the component of $F$ that
   contains the edges of $X(K_0)$
15 let $T^*$ be the spanning tree of $G^*$ rooted at $f_\infty$ that consists of edges not in $T$
16 let $e^*$ be a most balanced edge separator of $T^*$
17 if $e \in K_0 \setminus \bigcup_{j>0} K_j$ then return the fundamental cycle of $e$ w.r.t. $T$
18 let $j > 0$ be such that $e \in K_j$
19 let $H$ be the subgraph of $G$ induced by the faces in $K_0 \setminus \bigcup_{l>0} K_l$
20 let $H'$ denote the set of faces in $T^*_e$ that belong to $H$
21 let $C$ be the boundary of $H'$
22 let $C_1, C_2, \ldots, C_\ell$ be a decomposition of $C$ into simple cycles
23 let $H_k$ denote the set of faces enclosed by $C_k$ (for $1 \le k \le \ell$)
24 if $w(H_k) \ge W/4$ for some $1 \le k \le \ell$ then return $C_k$
25 else
26     let $r$ be such that $W/4 \le w\left(K_j \cup \bigcup_{1 \le k \le r} H_k\right) \le 3W/4$
27     return the boundary of $K_j \cup \bigcup_{1 \le k \le r} H_k$
```

this we mean that the BFS is seeded with the component T of F that contains the edges of $X(K_0)$. Whenever a vertex in a component T' of F is first visited by the search, T' is added to T, and all the vertices of T' are marked as visited. This three-step construction of the spanning tree T is important for ensuring that the cycle returned by the procedure is a simple cycle (see appendix).

The algorithm next computes a spanning tree T^* of G^*, consisting of the edges not in T. It finds a most balanced edge separator e^* in T^*. If $e \in H$, then the fundamental cycle of e w.r.t. T is returned (line 17). Otherwise, $e^* \in K_j$ for some $j \ge 1$ ($e^* \notin K_0$ since, by construction, such fundamental cycles are not balanced). Let T^*_e denote the subtree of T^* rooted at e^*. Note that the vertices of T^*_e are exactly the set of faces enclosed by the fundamental cycle of e w.r.t. T. Let H' be the set of faces in $T^*_{e^*}$ that do not belong to K_j.

Let C be the bounding cycle of H'. C decomposes into one or more more simple cycles $C_1, C_2, \ldots, C_\ell$. If any C_k is a balanced separator, it is returned (Line 24). Otherwise, there must be a prefix of the C_k's whose interior, together with the faces of component K_j is a set of faces whose boundary is a balanced simple cycle separator.

We prove in the appendix that Algorithm 1 always returns a 3/4–balanced simple cycle separator with at most $\sqrt{8m}$ edges. As discussed in Section 2, if the weight is defined as the number of vertices then 3/4 can be replaced with 2/3 throughout the paper.

4 Experiments

In this section we evaluate the performance of our algorithm and compare it to prior results. One of the striking findings in the experiments of Holzer et al. [20] is that the Fundamental Cycle Separator algorithm is usually very effective in finding small, balanced cycle separators. Our goal in this paper is to establish that our algorithm, which *does* provide a worst-case guarantee on both separator size and balance, is competitive with FCS both in terms of runtime and in average-case cycle size and balance. We do not directly compare our results with the other algorithms presented in [2, 20], primarily because they do not produce simple cycle separators, but also because FCS's performance was shown to be dominant in most cases.

The FCS algorithm [20] operates as follows: first, it computes a primal BFS tree T spanning the graph. Recall that the edges *not* in T form a spanning tree T^* of the dual graph. Each primal edge $e = uv$ not in T defines a fundamental cycle, the one formed by e and the unique path from u to v in T. Working from leafs of T^* towards its root, we can efficiently compute the weight enclosed by each fundamental cycle of T. The algorithm returns one of these cycles that is a balanced separator.

There is a trade-off between balance and cycle length. If one prefers short cycles, it makes sense to return the shortest cycle amongst those that are at least 2/3–balanced. On the other hand, one cannot necessarily return the most balanced cycle of length $O(\sqrt{m})$, as FCS has no such cycle size guarantee. In this case, for both FCS and our algorithm, we return the first balanced separator found. Experimenting with different choices produced results not significantly different from those reported here.

We used two variants of our algorithm and two variants of FCS in order to illustrate various issues that arise in our experiments. Recall that our algorithm may return a level cycle (line 9 in Algorithm 1), a fundamental cycle (line 17), or a fundamental-and-level-cycle merger (lines 24, 27). The first variant of our algorithm implements the algorithm as presented in Section 3. In particular, it terminates as soon as it encounters a cycle that is a 2/3–balanced separator. We refer to this variant as *unoptimized*. The second variant, which we refer to as *optimized for balance*, computes all candidate separators and returns the most balanced candidate found.

Similarly, the first variant of FCS is the one that terminates as soon as it encounters a cycle that is a 2/3–balanced separator. We refer to this variant as *unoptimized*. The second variant, which we refer to as *optimized for length*, returns the shortest fundamental cycle among all those that are 2/3–balanced.

4.1 Data Sets and Experimental Setup. To effectively compare our algorithms, we draw extensively from the graphs tested experimentally in [2, 20]. Each graph is triangulated before testing. Note that there is some degree of freedom in triangulation (Holzer et al. [20] use the triangulation routines provided by LEDA). As our graphs are represented by permutations of the darts (see [25] for more on representing embeddings), we simply triangulate by walking through the permutation describing the faces, and if any orbit is larger than three, we insert an edge to produce a triangle and reduce the size of the orbit by one.

Below is a list of the classes of graphs.

1. `grid` are square grid graphs; `rect` are rectangular grid graphs. `c-grid` are two such graphs connected via 5 joining vertices that form a perfectly balanced cycle separator.

2. `sixgrid` graphs are tessellated hexagons.

3. A k-iteration `tri` graph starts with a triangle, and on each of k iterations, each face except f_∞ has a new vertex embedded within it and connected to each vertex on the face's boundary.

4. `globe` graphs approximate spheres, and are implemented by wrapping a `rect` into a cylinder and adding a vertex on top and bottom connected to the vertices of the top and bottom rows, respectively. We call very skewed globes `eggs`.

5. `cylinder` graphs are similar to `globe` graphs, with the addition of an extra vertex in every square. BFS trees produced for `cylinder` and (triangulated) `globe` graphs differ substantially.

6. A `diameter-`k is essentially a narrow, length-k strip, triangulated in a way that maintains a diameter of k and a very small separator (cf. Figure 7 in [20]).

7. The `airfoil` graph is a finite-element mesh of real-world data [9].

8. The graph `col` is the `USA-COL` road network used in the 9th DIMACS Implementation Challenge — Shortest Paths [8], accessible online [7]. We repeatedly removed vertices of degree at most 1. Then, we interpreted the graph and the coordinates as a straight-line embedding and we added vertices whenever two edges intersect geometrically.

All tests are run on a machine with two Intel Xeon X5650 processors and 47.3 gigabytes of RAM. The code is compiled using GCC 4.4.5 targeting x86_64. The operating system is Debian. Runtime tests were run single-threaded on an otherwise-idle machine. Time is measured in CPU clock ticks, using the `clock` function in C. Instances were sufficiently large that the clock granularity is negligible.

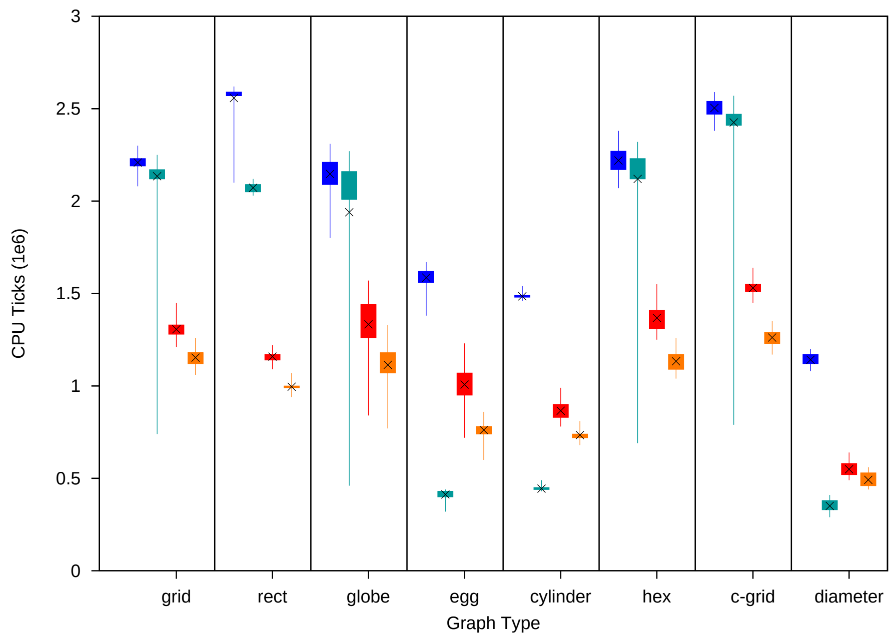


Figure 3: Running time (in CPU clocks) for our algorithm, both optimized for balance (blue) and unoptimized (teal); and for FCS, optimized for length (red) and unoptimized (orange).

4.2 Results and Interpretation. Following Holzer et al. [20], we use whisker plots (Figures 5, 7(a), and 7(b)) to show the runtimes for, and the balance and separator size produced by our algorithm and FCS for various types of graphs. For each graph, we tried a large sample of possible faces as roots of the component trees for our algorithm, and possible vertices as roots of the primal BFS trees for FCS. The box in each plot corresponds to the middle 50% of values obtained for all choices of root vertices; the whiskers span the entire range of values observed. The 'X' mark in each plot indicates the mean of observed values.

4.2.1 Running Time. Figure 3 shows running times of the two variants of our algorithm and the two variants of FCS on various graphs, all with roughly 1,000,000 vertices. Specifically, we test on a 1000 by 1000 `grid`, 100 by 10000 `rect`, 1000 by 1000 `globe`, 100 by 10000 `egg`, 10000 by 50 `cylinder`, 700 square `hex`, 707 by 707 `c-grid`, and a `diameter`-333333 graph.

It is evident that our algorithm is typically slower than FCS, but only by a factor of at most two. This is to be expected since both our algorithm and FCS compute a BFS tree and find a fundamental cycle separator, but our algorithm also performs a dual BFS to compute the component tree and level cycles.

We note that for the `egg`, `cylinder`, and `diameter` graphs, as well as occasionally, for `grid`, `globe`, and `hex`, the unoptimized variant of our algorithm is faster than both variants of FCS. To explain this behavior, we examined the type of separator returned by the unoptimized variant of our algorithm. These are shown in Figure 4 (left). Recall that the unoptimized variant of our algorithm terminates as soon as it finds a 2/3–balanced cycle. The graphs for which this variant is fast are exactly those for which it typically returns a level cycle as the separator. For these graphs, this variant only computes the component tree and level cycles, but does not perform a primal BFS nor the search for a fundamental cycle separator. In graphs with skewed aspect ratio, such as `egg`, and `cylinder`, dual BFS levels are necessarily small, hence there exists a level cycle that forms a short, balanced separator.

Interestingly, it is extremely rare that the unoptimized algorithm resorts to shortcutting a fundmental cycle separator using cycle levels. This infrequent outcome accounts for a disproportionate amount of the algorithm's complexity (conceptually, but not so much in lines of code), but is required for providing the worst-case guarantee. This implies that complementing the FCS algorithm with computing level cycles (i.e., computing a dual BFS) is in itself a useful and efficient simple cycle separator heuristic.

Figure 4 (right) shows that the balance-optimized variant of our algorithm does shortcut fundamental cycles using level cycles for most starting faces of the

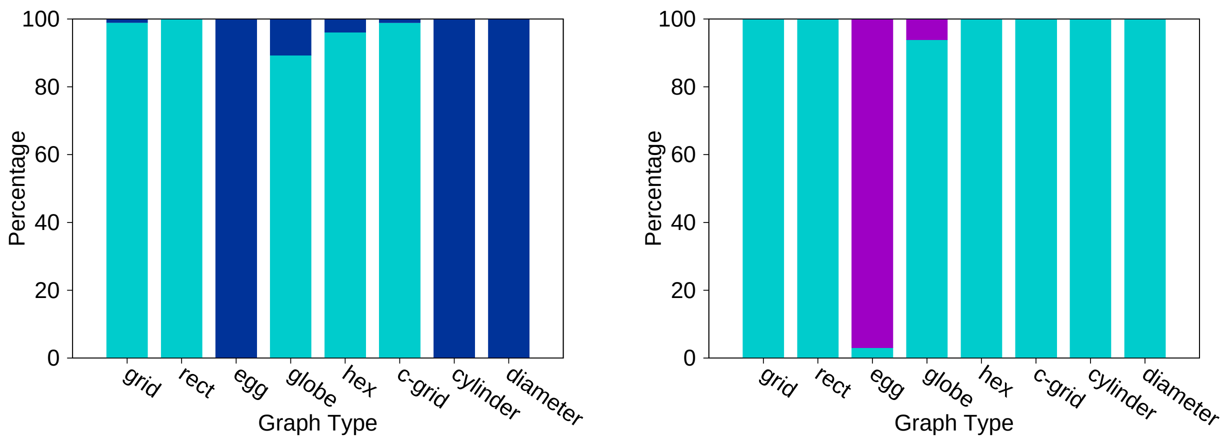


Figure 4: The type of separator returned by our algorithm – unoptimized (left) and optimized for balance (right) on the graphs used in Figure 3. Teal corresponds to fundamental cycles (line 17 in Algorithm 1), dark blue to level cycles (line 9), and purple to a fundamental-and-level cycle merger (lines 24–27).

`egg` graph and for some of the `globe` graphs. In those instances, the time spent on shortcutting the FCS is a little less than the time spent on finding the FCS.

Figure 5 shows the running time as a function of the size of various square grids. As expected, both FCS and our algorithm appear to scale linearly in the number of vertices. The change in minimum running time around 20 million vertices can be most likely attributed to caching effects. Again, we see here that, on average, our algorithm takes no more than twice as long as FCS. The slopes of the lines of best-fit (3.1 and 1.9 for our algorithm and FCS, respectively) suggest that our algorithm tends to be 1.6 times slower than FCS. As shown in Figure 7(b), one might have to run FCS several times to locate a separator with size and balance within the guarantees provided by our algorithm. Therefore, we consider the runtime of our algorithm to be competitive.

4.2.2 Separator Balance and Size.

Figures 7(a) and 7(b) show results for balance and size, respectively. Each graph tested (except for `colorado`) has approximately 10,000 vertices and is triangulated. In particular, `grid` is 100 by 100, `rect` is 10 by 1000, `egg` is a 10 by 1000 globe, `globe` is 100 by 100, `hex` is 70 by 70, `c-grid` is 71 by 71, `tri` is ten iterations, `cylinder` is 1000 by 5, and `diameter` has diameter 3333.

In Figure 7(a), we see that unoptimized FCS never produces significantly more balanced separators than our unoptimized algorithm. However, for the graphs `egg`, `cylinder`, `tri`, and `diameter`, our algorithm produces nearly perfectly balanced separators every time, whereas FCS tends to find separators that are 2/3–balanced. Recall from Figure 4 and the discussion above that, for these graphs, our algorithm typically returns a level cycle as the separator. Since candidate level cycles are searched for starting from the median–balanced level (level i_0 in Section 3), and since in these graphs, BFS levels consist of few edges, the algorithm finds small level cycles that are nearly perfectly balanced. We have verified that optimizing both algorithms for balance (i.e. taking the most balanced separator encountered) eliminates those differences. Note, however, that optimizing FCS for balance is not guaranteed to return a short separator. In fact, for `cylinder` graphs, this is often the case.

Figure 7(b) shows that in most test cases, our algorithm returns slightly longer separators than FCS, but still well below the $\sqrt{8m}$ guarantee. However, for `egg` and `cylinder` graphs, FCS often produces separators whose length is well above our guarantee. Below, we focus on the classes of graphs for which FCS performs poorly.

To understand the extent to which FCS performs poorly on `cylinder` graphs, we compute the percentage of viable starting vertices (for FCS) and faces (for our algorithm) on a 1000 by 5 cylinder. Figure 6 is a graph of the cumulative frequency diagram for separator size. The results are shown for the unoptimized variant of our algorithm, and for three variants of FCS: the unoptimized and shortest–balanced versions previously described, and a randomized version of the shortest–balanced heuristic. To randomize FCS, at each level of constructing the BFS tree, we shuffle the order in which vertices are processed. This heuristic is one of those studied in [20].

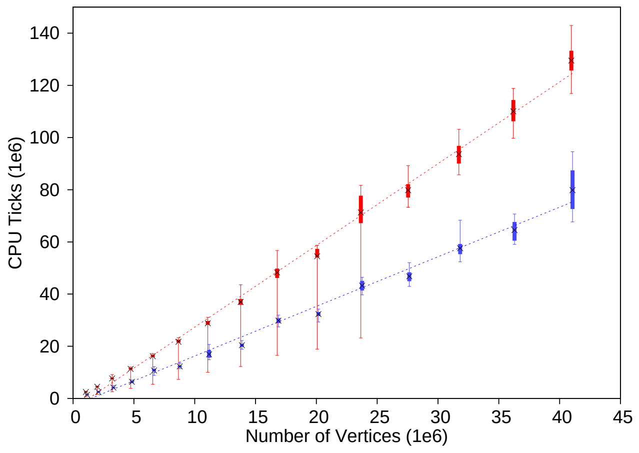


Figure 5: Runtime (in CPU clocks) versus number of vertices for square grid graphs for our algorithm (red) and FCS (blue).

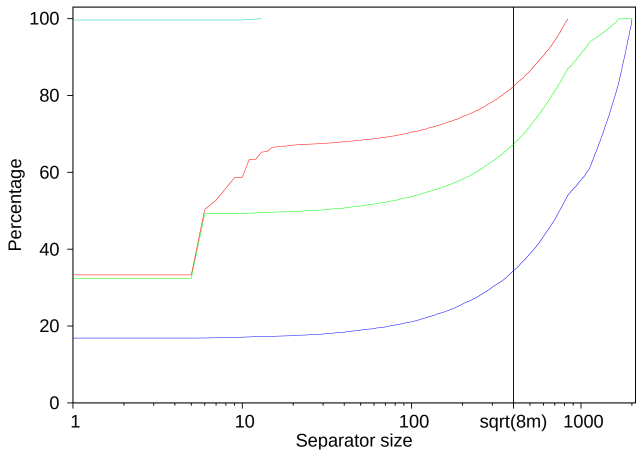


Figure 6: X-axis: separator size. Y-axis: percentage of starting vertices achieving a separator of at most this size. Comparing our algorithm (teal), shortest–balanced randomized FCS (red), shortest–balanced deterministic FCS (green), and unoptimized deterministic FCS (dark blue).

Our algorithm reaches 100% well before the guaranteed $\sqrt{8m}$ size bound, while all versions of FCS have many poor choices of starting vertex. This implies that one might need to try FCS several times before locating a viable separator. Note that, especially for the shortest–balanced randomized version of FCS, the expected number of attempts is small (less than two).

5 Conclusions

In this paper we described an implementation of a simple cycle balanced separator algorithm for planar graphs with proven worst-case guarantees on the separator's size. To the best of our knowledge, the only other algorithm that has been implemented and guarantees a simple cycle separator is the Fundamental Cycle Separator Algorithm. However, unlike our algorithm, FCS does not provide worst-case size guarantees.

Our experiments show that the running time of our algorithm is comparable to that of FCS on all instances. We demonstrate families of graphs for which our algorithm finds extremely small separators, while FCS often finds separators that are much larger than even the worst-case guarantee of our algorithm. However, when FCS is optimized for returning a shortest balanced FCS, and employs randomization in the construction of the BFS tree, it finds a small separator with high probability. Note that using heuristics diminishes the attractiveness of the FCS algorithm, which stems from its simplicity, whereas our algorithm performs well without further optimizations or heuristics. An interesting conjecture in this context is that FCS works well with constant probability on any input graph, where the probability is over the choices of BFS trees (choice of root as well as choice of order in which nodes are visited).

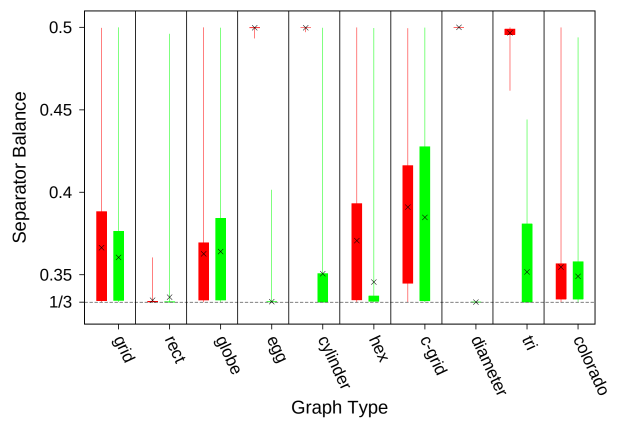
We further observe that our algorithm seldom requires the somewhat complicated last phase, which combines a long fundamental cycle with short level cycles to produce a short separator. For almost all test instances, our algorithm returns either a level cycle or a fundamental cycle as the short separator. This implies that complementing the FCS algorithm with computing level cycles (i.e., computing both the primal and dual BFS) is in itself a useful and efficient simple cycle separator algorithm (albeit without the theoretical worst-case guarantee).

We conclude that our algorithm is a viable alternative to FCS that outperforms it in certain cases. We believe that optimizations such as those studied by Holzer et al. would further enhance its performance.

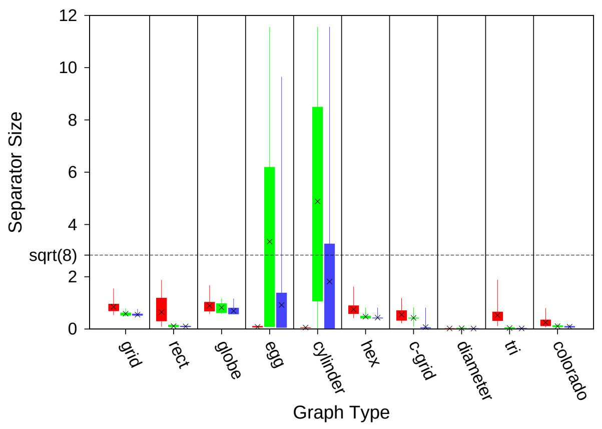
An interesting direction for future work is implementing the separator algorithm in [26], which is quite similar to the algorithm discussed in the current paper. An implementation of the algorithm in [26] can be used to compute r–divisions in asymptotic linear time. It would be interesting to see how well that algorithm actually performs in practice.

Acknowledgements

This work originated from MIT course 6.889 on "Algorithms for planar graphs and beyond." The authors would like to thank Erik Demaine and Philip Klein for fruitful discussions. Additionally, we thank Philip Klein and his team for providing us with their planar graph



(a) Separator balance (number of vertices in smaller part divided by the total number of vertices) for our unoptimized algorithm (red) with unoptimized FCS (green).



(b) Separator size divided by $\sqrt{m}$ for our unoptimized algorithm (red), unoptimized FCS (green), and shortest–balanced FCS (blue).

Figure 7: Separator balance and size for various graphs

library.

CS was partially supported by the Swiss National Science Foundation. Work was conducted while CS was at MIT. SM was partially supported by National Science Foundation Grants CCF-0964037 and CCF-1111109.

References

[1] Lyudmil Aleksandrov and Hristo Nicolov Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM Journal on Discrete Mathematics*, 9(1):129–150, 1996.

[2] Lyudmil Aleksandrov, Hristo Nicolov Djidjev, Hua Guo, and Anil Maheshwari. Partitioning planar graphs with costs and weights. *ACM Journal of Experimental Algorithmics*, 11, 2006. Announced at ALENEX 2002.

[3] Noga Alon, Paul D. Seymour, and Robin Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990. Announced at STOC 1990.

[4] Punyashloka Biswal, James R. Lee, and Satish Rao. Eigenvalue bounds, spectral partitioning, and metrical deformations via flows. *Journal of the ACM*, 57(3), 2010. Announced at FOCS 2008.

[5] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In *52nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 170–179, 2011.

[6] Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1–2):361–381, 2012. Announced at SODA 2006.

[7] C. Demetrescu, A. Goldberg, and D. Johnson. 9th DIMACS Implementation Challenge — Shortest Paths. `http://www.dis.uniroma1.it/challenge9/download.shtml`; accessed 21 Oct 2012.

[8] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson. Implementation challenge for shortest paths. In *Encyclopedia of Algorithms*. 2008.

[9] Ralf Diekmann and Robert Preis, 1998. `http://www2.cs.uni-paderborn.de/fachbereich/AG/monien/RESEA` accessed 21 Oct 2012.

[10] Hristo Nicolov Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic and Discrete Methods*, 3:229–240, 1982.

[11] Hristo Nicolov Djidjev. A linear algorithm for partitioning graphs of fixed genus. *Serdica. Bulgariacae mathematicae publicationes*, 11(4):369–387, 1985. Announced in *Comptes Rendus de l'Académie Bulgare des Sciences*, 34:643–645, 1981.

[12] Hristo Nicolov Djidjev and Shankar M. Venkatesan. Reduced constants for simple cycle graph separation. *Acta Informatica*, 34:231–243, 1997.

[13] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006. Announced at FOCS 2001.

[14] Lamis M. Farrag. Applications of graph partitioning algorithms to terrain visibility and shortest path problems. Master's thesis, School of Computer Science, Carleton University, 1998.

[15] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

[16] Hillel Gazit and Gary L. Miller. Planar separators and the Euclidean norm. In *SIGAL International Symposium on Algorithms*, pages 338–347, 1990.

[17] John R. Gilbert, Joan P. Hutchinson, and Robert Endre Tarjan. A separator theorem for graphs of bounded

genus. *Journal of Algorithms*, 5(3):391–407, 1984. Announced as TR82-506 in 1982.

[18] Michael T. Goodrich. Planar separators and parallel polygon triangulation. *Journal of Computer and System Sciences*, 51(3):374–389, 1995. Announced at STOC 1992.

[19] Monika Rauch Henzinger, Philip Nathan Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997. Announced at STOC 1994.

[20] Martin Holzer, Frank Schulz, Dorothea Wagner, Grigorios Prasinos, and Christos D. Zaroliagis. Engineering planar separator algorithms. *ACM Journal of Experimental Algorithmics*, 14, 2009. Announced at ESA 2005.

[21] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *43rd ACM Symposium on Theory of Computing (STOC)*, pages 313–322, 2011.

[22] Ken-ichi Kawarabayashi, Philip Nathan Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 135–146, 2011.

[23] Ken-ichi Kawarabayashi and Bruce A. Reed. A separator theorem in minor-closed classes. In *51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 153–162, 2010.

[24] Jonathan A. Kelner. Spectral partitioning, eigenvalue bounds, and circle packings for graphs of bounded genus. *SIAM Journal on Computing*, 35(4):882–902, 2006. Announced at STOC 2004.

[25] Philip N. Klein. *Flatworlds: Optimization Algorithms for Planar Graphs*. Draft available online at `http://www.planarity.org`.

[26] Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. 2012. Manuscript, preprint available on the arXiv `http://arxiv.org/abs/1208.2223`.

[27] Philip Nathan Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998. Announced at WADS 1993.

[28] Jakub Lacki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log \log n)$ time. In *19th European Symposium on Algorithms (ESA)*, pages 155–166, 2011.

[29] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[30] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980. Announced at FOCS 1977.

[31] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986. Announced at STOC 1984.

[32] Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–222, 2012.

[33] Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. In *18th Annual European Symposium on Algorithms (ESA)*, 2010.

[34] Serge A. Plotkin, Satish Rao, and Warren D. Smith. Shallow excluded minors and improved graph decompositions. In *5th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 462–470, 1994.

[35] Bruce A. Reed and David R. Wood. A linear-time algorithm to find a separator in a graph excluding a minor. *ACM Transactions on Algorithms*, 5(4), 2009. Announced at EuroComb 2005.

[36] Daniel A. Spielman and Shang-Hua Teng. Disk packings and planar separators. In *12th Symposium on Computational Geometry (SoCG)*, pages 349–358, 1996.

[37] Peter Ungar. A theorem on planar graphs. *Journal of the London Mathematical Society*, s1-26(4):256–262, 1951.

[38] Karl Georg Christian von Staudt. *Geometrie der Lage*. Bauer und Raspe, Nürnberg, 1847.

[39] Hassler Whitney. Non-separable and planar graphs. *Transactions of the American Mathematical Society*, 34(2):339–362, 1932.

[40] Christian Wulff-Nilsen. Separator theorems for minor-free and shallow minor-free graphs with applications. In *52nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 37–46, 2011.

A Correctness of Algorithm 1

A.1 Simple Cycle

LEMMA A.1. *For every K_i ($i \geq 0$) there is a single edge $e^* \in T^*$ whose rootward endpoint is not a face of K_i and whose other endpoint is a face of K_i.*

Proof. The claim is true for $i = 0$ since the root of T^* is f_∞ and since exactly one edge of $X(K_0)$ is in T^*. For $i \geq 1$, assume there is more than one edge where T^* enters K_i. Consider two such edges e_1^*, e_2^*. Consider the embedding of H inherited from G. The unique path in T^* between e_1^* and e_2^* partitions the embedding of H into at least two regions such that the vertices of $X(K_i)$ are not all in a single region. Since T^* does not contain any edges of the forest F, this contradicts the fact that the vertices of $X(K_i)$ are connected in F. □

For a given $i \geq 0$, we say that T^* enters K_i at the unique edge e of $X(K_i)$ that belongs to T^*. We use the edge e at which T^* enters K_i to define an order on the vertices of $X(K_i)$. Namely, the order in which the vertices of $X(K_i)$ are encountered in a clockwise walk along $X(K_i)$, starting from e.

LEMMA A.2. *Fix $j \geq 1$. Let $x_1, x_2, \ldots x_\ell$ be the set of vertices in $X(K_0) \cap X(K_j)$, in clockwise order along $X(K_0)$, starting from the edge of $X(K_0)$ at which T^* enters K_0. Then this is also the order in which these vertices are visited in clockwise order along $X(K_j)$, starting from the edge of T^* at which T^* enters K_j.*

Proof. The lemma is trivial for $\ell < 2$. Assume, therefore, that $\ell \geq 2$. T^* enters K_0 between x_ℓ and x_1. Lemma A.1 implies that T^* enters K_j between x_ℓ and x_1 as well. Suppose that x_{i+1} does not follow x_i in the clockwise order of vertices along $X(K_j)$. Then either $X(K_0)$ does not enclose $X(K_j)$, or $X(K_0)$ is not a simple cycle – a contradiction.

We establish some properties of T and T^* that are critical for our implementation. The first two lemmas follow immediately from the definition of T.

LEMMA A.3. *Let u, v be vertices on $X(K_0)$. The unique u-to-v path in T consists only of edges of $X(K_0)$.* □

LEMMA A.4. *Let u, v be vertices on $X(K_j)$ for any fixed $j \geq 0$. The unique u-to-v path in T consists only of edges in F.* □

LEMMA A.5. *Let u, v be vertices on $X(K_i)$ for any fixed $i \geq 1$. Any u-to-v path P that consists only of edges of $\cup_{j \geq 1} X(K_j)$ uses only edges of $X(K_i)$.*

Proof. The proof is by contradiction. Assume P contains some edge that does not belong to $X(K_i)$. Without loss of generality, P contains no edges of $X(K_i)$ (otherwise choose such a subpath of P). Let C be a simple cycle formed by a u-to-v subpath of $X(K_i)$ and by P. Observe that C consists of level i_+ edges, so all faces enclosed by C have level at least i_+. However, since the $X(K_j)$'s are edge disjoint, C must enclose some face that is not enclosed by any $X(K_j)$, i.e., a face whose level is smaller than i_+, a contradiction. □

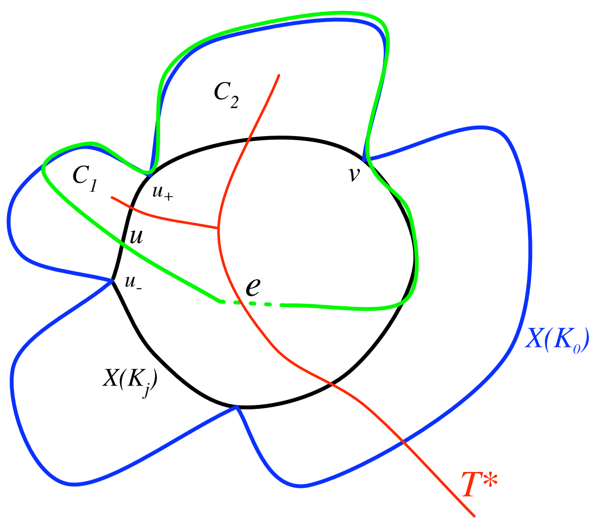


Figure 8: Illustration of the interaction between $X(K_0), X(K_j)$ and T used in the various lemmas in this section. $X_(K_0)$ is shown in blue. $X(K_j)$ is black. The fundamental cycle of e is green. Some (dual) edges of T^* are shown in red.

Combining the above lemmas we get:

LEMMA A.6. *Let u, v be vertices on $X(K_i)$ for some fixed $i \geq 1$. The unique u-to-v path P in T can be decomposed into five (possibly empty) subpaths:*

1. *P_1 consisting only of edges of $X(K_i)$*
2. *P_2 that starts at a vertex of $X(K_i)$, ends at a vertex of $X(K_0)$ and is internally disjoint from vertices of both $X(K_0)$ and $X(K_i)$*
3. *P_3 that is a subpath of $X(K_0)$. If the first (last) vertex on P_3 belongs to $X(K_i)$ then P_2 (P_4) is empty.*
4. *P_4 that starts at a vertex of $X(K_0)$, ends at a vertex of $X(K_i)$ and is internally disjoint from vertices of both $X(K_0)$ and $X(K_i)$*

5. *P_5 consisting only of edges of $X(K_i)$*

Proof. Refer to Figure 8. Let P_1, P_5 be the maximal prefix and suffix of P consisting only of edges of $X(K_i)$. Let P' denote the subpath of P between P_1 and P_5. If P' is non-empty, then it is a path between vertices of $X(K_i)$ that includes some edge not in $X(K_i)$. By Lemma A.5, it cannot consist solely of edges in $\cup_{j\geq 1}X(K_j)$. By Lemma A.4, P consists only of edges of F. Hence, P' must include some edge of $X(K_0)$. By Lemma A.3, all vertices of $X(K_0)$ in P form a subpath of P that consists only of edges of $X(K_0)$. Hence this is also a subpath of P'. Denote it by P_3. The remaining two subpaths of P', which we denote by P_2 and P_4, consist only of edges not in $X(K_0)$. By Lemma A.4, P_2 and P_4 do not contain any vertices of $X(K_i)$, other than the first vertex of P_2 and the last vertex of P_4. Hence, if the first (last) vertex of P_3 belongs to $X(K_i)$, P_2 (P_4) must be empty. □

Consider the case where $e^* \in K_j$. Let C be the fundamental (and thus simple) cycle of T w.r.t. e. Let Q be the maximal subpath of C that contains e and has no internal vertices that are vertices of $X(K_j)$. Write $C = P \circ Q$. Since P is a path in T between vertices that are on $X(K_j)$, it can be written as $P = P_1 \circ P_2 \circ P_3 \circ P_4 \circ P_5$, as specified in Lemma A.6. Let u, v be the endpoints of P_2 and P_4, respectively, that belong to $X(K_j)$, where u appears before v in the order of vertices of $X(K_j)$.

Lemma A.7. *The unique path $P' = P_2 \circ P_3 \circ P_4$ in T between u and v visits no vertices of $X(K_j)$ that appear before u or after v in the order of vertices along $X(K_j)$.*

Proof. By Lemma A.6, since P_2 and P_4 are internally vertex disjoint from $X(K_j)$, it suffices to show that P_3 visits no vertices of $X(K_j)$ that appear before u or after v in the order of vertices along $X(K_j)$. If P_3 contains no vertices of $X(K_j)$, the lemma is trivially true. Otherwise, $X(K_0) \cap X(K_j)$ is non empty.

Let u_- (u_+) be the vertex in $X(K_0) \cap X(K_j)$ that weakly precedes (follows) u in the order of vertices along $X(K_j)$. Assume first, that such predecessor and successor exist. Similarly, Let v_- (v_+) be the vertex in $X(K_0) \cap X(K_j)$ that precedes (follows) v in the order of vertices along $X(K_j)$. Let Q^0_u be the unique u_--to-u_+ subpath of $X(K_0)$ that does not include the edge at which T^* enters $X(K_0)$. Let Q^j_u be the unique u_--to-u_+ subpath of $X(K_j)$ that does not include the edge at which T^* enters $X(K_j)$. By planarity, P_2 is confined to the interior of the cycle formed by Q^0_u and Q^j_u. Hence, the first vertex x of P_3 must either be u (in case $u \in X(K_0)$, or an internal vertex of Q^0_u. A similar argument shows that the last vertex y of P_3 is either v or an internal vertex of Q^0_v (where Q^0_v is the unique v_--to-v_+ subpath of $X(K_0)$ that does not include the edge at which T^* enters $X(K_0)$).

Since neither P_2 nor P_4 use edges of T^*, and since P' is simple, x must precede y in the order of vertices of $X(K_0)$ (see Figure 8). Recall that P_3 is the unique x-to-y subpath of $X(K_0)$ that does not include the edge at which T^* enters $X(K_0)$. Therefore the vertices in P_3 appear in increasing order of the vertices of $X(K_0)$. This means that if P_3 contains a vertex of $X(K_j)$, the first such vertex is u_+, and the last such vertex is v_-. It follows, by Lemma A.2 that P_3 only visits vertices of $X(K_j)$ that follow u and precede v in the order of vertices along $X(K_j)$.

The case where the predecessor u_- or the successor u_+ does not exist is handled in a similar manner. Note that only one of these can happen simultaneously, since we assume $K(X_0) \cap X(K_j) \neq \emptyset$. Also note that this implies $u \notin X(K_0)$ since otherwise $u = u_- = u_+$. Suppose the predecessor does not exist (the other case is symmetric). In this case we define u_- to be the first vertex of $X(K_0)$. As above, let Q^0_u be the unique u_--to-u_+ subpath of $X(K_0)$ that does not include the edge at which T^* enters $X(K_0)$. Note that u_+ is the only vertex of Q^0_u that belongs to $X(K_j)$. Since P_2 does not use any edges of T^*, the first vertex x of P_3 must be a vertex of Q^0_u. The remainder of the argument proceeds as above. □

Let H be the subgraph of G induced by the faces in $K_0 \setminus \cup_{i>0}K_i$. Let H' denote the set of faces in T^*_e that belong to H. The boundary C of H' consists of the path P' described in Lemma A.7, and of the subpath Q of $X(K_j)$ between u and v that does not contain the edge of T^* that enters K_j. Lemma A.7 shows that even though C might not be a simple cycle, the non-simplicities arise in a structured, monotonic way. Specifically, the vertices of P' that belong to $X(K_j)$ are all vertices of Q and appear in the same order along both P' and Q. Put in other words, and using the notation $\{C_i\}, \{H_i\}$ as defined in the algorithm (lines 21–23), we have the following lemma.

Lemma A.8. *The cycle returned by the algorithm is simple.*

Proof. The cycle returned in Lines 9 is simple since component boundaries are simple cycles. The cycle returned in Line 17 is simple since it is a fundamental cycle. The cycle returned in Line 24 is simple by definition of the C_k's. It remains to show that the boundary of $K_j \cup \bigcup_k H_k$ is a simple cycle. The boundary of $K_j \cup \bigcup_k H_k$ consists of a prefix of P' and of $\bar{Q}$, the path consisting of a suffix of Q and of the edges of $X(K_j)$

that do not belong to Q The lemma follows since, by the argument preceding this lemma, P' and $\bar{Q}$ are vertex disjoint, and since both are simple paths with the same endpoints. □

We have shown that the cycle returned by the algorithm is simple. We next argue that this simple cycle is a short balanced separator.

A.2 Balance and Cycle Length

LEMMA A.9. *The simple cycle returned by the algorithm is $3/4$–balanced*

Proof. Clearly, the cycles returned in Lines 9, 17, or 24 are balanced separators. It only remains to argue that if Line 26 is reached, then there exists an r such that $K_j \cup \bigcup_{1\le k\le r} H_k$ is $3/4$–balanced. By construction of T, the fundamental cycle of e w.r.t. T is enclosed by $X(K_0)$. Since this fundamental cycle is a balanced separator (albeit not a short one), this implies that $w\left(K_j \cup \bigcup_{1\le k\le \ell} H_k\right) \ge W/4$. Since the conditions in line 9 and 24 are both false, $w(K_j) < W/4$ and the weight enclosed by any individual C_k is at most $W/4$. Hence, an appropriate r must exist. □

LEMMA A.10. *The forest F consists of at most $\sqrt{2m}$ edges.*

Proof. By choice of the levels i_- and i_+, each level consists of at most $\sqrt{m/2}$ edges. Since F consists of at most all edges of these two levels, it consists of at most $\sqrt{2m}$ edges. □

LEMMA A.11. *Let u be a vertex in H. The root-to-u path in T consists of at most $\sqrt{m/2}$ edges that do not belong to F.*

Proof. Let f be a face to which u is incident. The level of f is between i_- and i_+. By definition of levels of faces, there exists a face f' that is incident to $X(K_0)$ and whose distance in the dual of H from f is at most $(i_+ - i_-)$. Since each face is a triangle, this implies that there exists a path in H from u to some vertex of $X(K_0)$ whose length is at most $(i_+ - i_-)/2$. Since T is obtained from F by breadth-first search, the root-to-u path in T consists of at most $(i_+ - i_-)/2$ edges not in F.

It remains to bound $i_+ - i_-$. Since the cycles bounding different components are edge disjoint, and since every level between i_- and i_+ consists of at least $\sqrt{m/2}$ edges, $(i_+ - i_-) * (\sqrt{m/2}) < m$. This shows that $i_+ - i_- < \sqrt{2m}$. Hence, the root-to-u path in T consists of at most $\sqrt{m/2}$ edges that are not in F. □

THEOREM A.1. *The algorithm returns a $3/4$–balanced simple cycle separator with at most $\sqrt{8m}$ edges.*

Proof. Lemma A.8 shows that the cycle returned by the algorithm is simple. Lemma A.9 shows that it is a $3/4$–balanced separator. It remains to bound the length of the cycle. The cycle returned in Line 9 consist of at most $\sqrt{m/2}$ edges. The fundamental cycle returned in Line 17 consists of two paths of T between the endpoints of e and vertices on $X(K_0)$ and from a subpath of $X(K_0)$. The length of each of the former two paths is bounded by Lemma A.11 by $\sqrt{m/2}$. By choice of i_-, the length of $X(K_0)$ is at most $\sqrt{m/2}$. Hence, the length of the cycle returned in Line 17 is at most $3\sqrt{m/2} < \sqrt{8m}$.

The cycle returned in Line 27 consists of edges of $X(K_j)$, edges of $X(K_0)$ and two paths in T between vertices of $X(K_j)$ and vertices of $X(K_0)$. By choice of i_- and i_+, the length of $X(K_0)$ and the length of $X(K_j)$ are both bounded by $\sqrt{m/2}$. Note that the edges of $X(K_0)$ and $X(K_j)$ belong to F . By Lemma A.11, each of the paths of T consists of at most $\sqrt{m/2}$ edges not in F. Hence the boundary of $K_j \cup \bigcup_{1\le k\le r} H_k$ consists of at most $4\sqrt{m/2} = \sqrt{8m}$ edges. □