Short and Simple Cycle Separators in Planar Graphs

ELI FOX-EPSTEIN, Tufts University
SHAY MOZES, MIT
PHITCHAYA MANGPO PHOTHILIMTHANA, UC Berkeley
CHRISTIAN SOMMER

We provide an implementation of an algorithm that, given a triangulated planar graph with m edges, returns a simple cycle that is a 3/4-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, for example, 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. [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 (1) our algorithm is competitive across all test cases in terms of running time, balance, and cycle length; (2) it provides worst-case guarantees on the cycle length, significantly outperforming FCS on some instances; and (3) it scales to large graphs.

Categories and Subject Descriptors: G.3 [**Mathematical Software**]: Mathematical Software Performance; G.1.1 [**Combinatorics**]: Combinatorial Algorithms; G.1.2 [**Graph Theory**]: Graph Algorithms

General Terms: Algorithms

Additional Key Words and Phrases: Design, algorithms, performance, planar graphs, cycle separator

ACM Reference Format:
Eli Fox-Epstein, Shay Mozes, Phitchaya Mangpo Phothilimthana, and Christian Sommer. 2016. Short and simple cycle separators in planar graphs. J. Exp. Algorithmics 21, 2, Article 2.2 (September 2016), 24 pages.
DOI: http://dx.doi.org/10.1145/2957318

1. INTRODUCTION

Separators identify structure in a graph by cleaving it into two balanced parts with limited 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 [Ungar 1951; Lipton and Tarjan 1979; Djidjev 1982; Miller 1986; Gazit and Miller 1990; Spielman and Teng 1996; Djidjev and Venkatesan 1997], bounded-genus

A preliminary version of this work appeared in Fox-Epstein et al. [2013].

Authors' addresses: E. Fox-Epstein, Dept. of Computer Science, 115 Waterman St, Providence RI 02912, USA; email: ef@cs.brown.edu; E.F. is currently affiliated with Brown University. S. Mozes, Efi Arazi School of Computer Science, The interdisciplinary Center Herzliya, POB 167 Herzliya, 4610101, Israel; email: smozes@idc.ac.il; the bulk of this research was conducted while S. M. was at MIT. Parts of this work were conducted by S. M. while with Brown University and with IDC Herzliya. P. M. Phothilimthana, Department of Computer Science, 595 Soda Hall, Berkeley CA, 94720, USA; email: mangpo@csail.mit.edu; P. M. P. is currently affiliated with UC Berkeley. C. Sommer; email: csom@csail.mit.edu; the bulk of this research was conducted while C.S. was at MIT.

DOI: http://dx.doi.org/10.1145/2957318

graphs [Djidjev 1985; Gilbert et al. 1984; Kelner 2006], minor-free graphs [Alon et al. 1990; Plotkin et al. 1994; Reed and Wood 2009; Biswal et al. 2010; Kawarabayashi and Reed 2010; Wulff-Nilsen 2011], 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 limited interference. More formally, a separator of a graph $G = (V, E)$ is a partition of the vertices into three sets A, B, and S, where S is relatively small, A and B are of approximately the same size[1] (say, $|A|, |B| \leq 3|V|/4$), and no edges exist 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 [Frederickson 1987; Henzinger et al. 1997] or approximate vertex cover [Lipton and Tarjan 1980], and many more [Goodrich 1995; Klein and Subramanian 1998; Fakcharoenphol and Rao 2006; Mozes and Wulff-Nilsen 2010; Cabello 2012; Kawarabayashi et al. 2011; Italiano et al. 2011; Łącki and Sankowski 2011; Borradaile et al. 2011; Mozes and Sommer 2012]. For planar graphs, it is known that $|S| = O(\sqrt{|V|})$ is possible, even for worst-case instances [Lipton and Tarjan 1979].

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 unique u-to-v path in T. Every fundamental cycle C separates an embedded planar 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 twice the diameter of the graph. Therefore, if the diameter is small, the simple *Fundamental Cycle Separator (FCS) algorithm* works well: arbitrarily select a root for a breadth-first search (BFS), 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 [1979] (henceforth referred to as Lipton-Tarjan) finds a 2/3-balanced separator by identifying a set S' of small size $\left|S'\right| = O(\sqrt{|V|})$, 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'.

For many planar graph algorithms such as those computing shortest paths [Klein and Subramanian 1998; Fakcharoenphol and Rao 2006; Mozes and Wulff-Nilsen 2010; Cabello 2012; Mozes and Sommer 2012], minimum cuts [Italiano et al. 2011; Łącki and Sankowski 2011], or maximum flows [Borradaile et al. 2011], it is crucial that the separator S forms a (simple) *cycle* in G. Unfortunately, adding vertices of the set S' to shortcut a fundamental cycle, as in the Lipton-Tarjan algorithm, results in a separator S that does not necessarily form a simple cycle. Neither the Lipton-Tarjan algorithm [Lipton and Tarjan 1979] nor Djidjev's algorithm [Djidjev 1982] obtains simple cycles, and the FCS algorithm does not provide any worst-case guarantees on the cycle length.

The algorithms of Miller [1986], Gazit and Miller [1990], and Djidjev and Venkatesan [1997] offer both guarantees: for any triangulated 2-connected planar graph, they can

[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.

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 [Farrag 1998; Aleksandrov et al. 2007; Holzer et al. 2009].

Farrag [1998] implemented an algorithm of Aleksandrov and Djidjev [1996] 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 et al. [2007] implemented a three-phase algorithm, which (1) partitions the graph *by levels*, (2) partitions the graph *by fundamental cycles*, and (3) 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 et al. [2009], who implemented the Lipton-Tarjan algorithm [1979] and Djidjev's algorithm [1982], 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 $\sqrt{8|E|}$ for triangulated planar graphs with at least 29 edges. We experimentally evaluate our algorithm and compare it to the FCS algorithm. As mentioned in Section 1.1, the experimental results of Holzer et al. [2009] 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 (1) our algorithm is competitive with FCS for the graphs in Holzer et al. [2009]; (2) it provides worst-case guarantees on the cycle length, significantly outperforming FCS on some instances; and (3) it scales to large graphs.

Our algorithm is, in spirit, similar to algorithms given in Miller [1986], Klein et al. [2013], and Klein and Mozes [2013], though it differs significantly in some details. It was designed with implementation ease in mind and evolved throughout the implementation process.

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* 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

[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.

BFS with a number of vertices or edges. To do this, imagine the search started from a super-vertex connected to all the seed vertices. 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 $s' \in S$.

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.

A *simple cut* is a subset $S \subseteq V(G)$ such that $G[S]$ is connected and $G[V(G) \setminus S]$ is connected. Given a set of vertices S, we denote with $\delta_G(S)$ the set of edges of G with exactly one endpoint in S: $\delta_G(S) = \{uv \in E(G) \mid |\{u, v\} \cap S| = 1\}$. When S is a simple cut, $\delta_G(S)$ is referred to as the *edge boundary* of the cut.

We provide a brief review of basic definitions and facts related to planar graphs, combinatorial embeddings, and planar duality. For elaboration, see also Klein and Mozes [2013].

2.1. Embeddings and Planar Graphs

We use an edge-centric definition of graphs suitable for combinatorial embeddings and implementation; see Klein and Mozes [2013] for a more detailed treatment.

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 (e is *incident* to v and v'). 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 article, 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 requires $4|E| + |V| + |F|$ 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 article. 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 without 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 1. (SIMPLE-CYCLE/SIMPLE-CUT DUALITY [WHITNEY 1932]). *A set of edges forms a simple cycle in a planar embedded graph G if and only if it forms the boundary of 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 2.1. (*Encloses*). Let C be a simple cycle in a connected planar embedded graph G with distinguished face f_∞. Then the edges of C form the boundary of a simple cut $\delta_{G^*}(S)$ for some set S of vertices of G^*, that is, 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. (VON STAUDT [1847]). *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 $\alpha|V(G)|$ with $\alpha < 1$.

Let G be a triangulated biconnected simple planar graph.[3] Any simple cycle C in G separates G into the interior of C (the subgraph strictly enclosed by C) and the exterior of C (the subgraph not enclosed by C), such that there are no edges between vertices strictly enclosed by C and vertices not enclosed by C. We 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 α-balanced for some constant α and $|C| = O(\sqrt{|E(G)|})$.

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 (not enclosed, respectively) by C is at most α of the total weight of G. General weights can be handled by considering just face

[3]If G is not biconnected, a simple cycle separator might not exist. If the degrees of faces in G are not bounded, a *small* cycle separator might not exist.

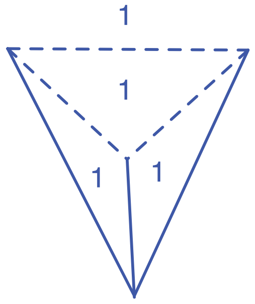


Fig. 1. A triangulated graph with unit face weights and a spanning tree (solid edges) for which no fundamental cycle is 2/3-balanced. In this example, all fundamental cycles are 3/4-balanced. In fact, this exemplifies the worst case. That is, a fundamental cycle with a balance of 3/4 always exists, provided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

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 article.

Recall that, for a biconnected planar graph G with maximum face size d, the algorithm of Miller [1986] finds a 2/3-balanced simple cycle separator with at most $2\sqrt{2\lfloor d/2\rfloor|V(G)|}$ vertices. It is easy to see that in the worst case, 2/3 is the best balance guarantee achievable (e.g., in a graph with three faces and uniform face weights). However, there may not exist a 2/3-balanced fundamental cycle separator, even in a biconnected triangulated planar graph where the weight of any single face is at most 2/3 of the total weight. This is illustrated in Figure 1. It is not difficult to see that if the weight of each face is at most 3/4 one can always achieve a balance of 3/4 on biconnected triangulated planar graphs. Furthermore, 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), a balance of almost 2/3 is achievable. Since our algorithm is based on finding a fundamental cycle separator, 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.

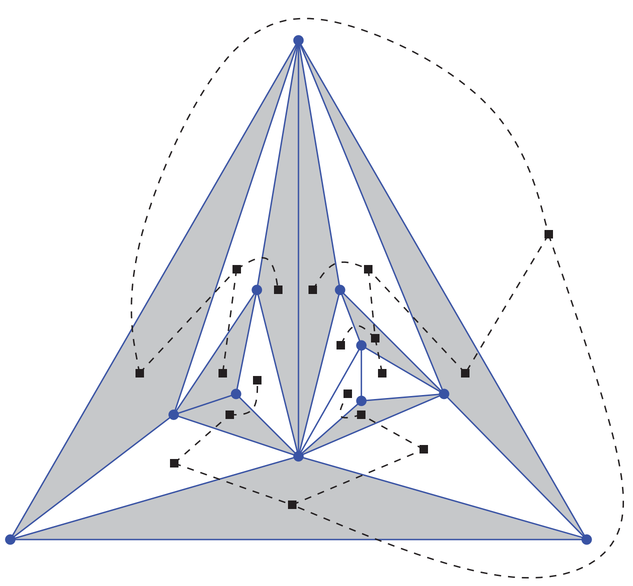
3. THE CYCLE SEPARATOR ALGORITHM

In this section, we describe our simple cycle separator algorithm. It roughly follows the overall structure of Miller's algorithm [Miller 1986] but is significantly different. The algorithm is similar to the one suggested recently in Klein et al. [2013], also described in the forthcoming book [Klein and Mozes 2013].

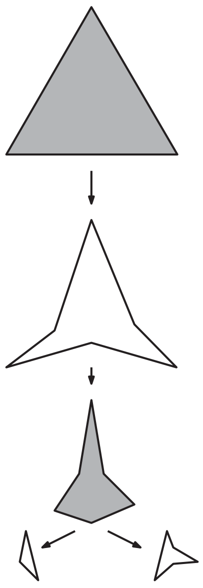
3.1. Levels and Level Components

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

Definition 3.1. The *level* of a face f is the minimum number of edges on an 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 $L_{\geqslant i}^F$ to denote the set of faces f having level at least i.



(a) A triangulated graph (blue discs as vertices, black squares as faces, solid blue primal edges) along with a dual BFS tree (in dashed black, rooted at the infinite face). Alternating levels of the BFS are highlighted gray.



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

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

Definition 3.2. For an integer $i \geq 0$, a connected component of the subgraph of G^* induced by $L^F_{\geqslant i}$ 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 *nontrivial* level component is a level component that is not G. The set of vertices of G^* (faces of G) belonging to a level component K is denoted by $F(K)$.

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

COROLLARY 3.3. *For any nontrivial level component K, $\delta_{G^*}(F(K))$, the edges crossing the cut $F(K)$ form a simple cycle in the primal G.*

In view of Corollary 3.3, for any nontrivial level component K, we use $X(K)$ to denote the simple cycle in the primal G consisting of the edges $\delta_{G^*}(F(K))$. We refer to $X(K)$ as the *level cycle* bounding K.

Definition 3.4. 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-1 component consisting of all of G^* except f_∞.

Figure 2 illustrates the definition of the component tree.

Definition 3.5. An edge ff' of G^* has level i if f has level $i-1$ and f' has level i. We use L^E_i 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

ALGORITHM 1: Cycle Separator Algorithm

```
1  triangulate $G$ and choose $f_\infty$ arbitrarily
2  construct the component tree $\mathcal{K}$
3  if $\exists$ a component $K \in \mathcal{K}$ s.t. $|X(K)| \le \sqrt{8m}$ and $W/4 \le w(K) \le 3W/4$ then return $X(K)$
4  let $i_-$ be the maximum level where $|L^E_{i_-}| \le \sqrt{m/2}$ and $\exists$ a level $i_-$ component $K_0$ with
   $w(K_0) \ge 3W/4$ (or 1 if no level qualifies)
5  let $i_+$ be the minimum level greater than $i_-$ where $|L^E_{i_+}| \le \sqrt{m/2}$
6  let $K_1, K_2, \ldots, K_q$ be the components at level $i_+$ contained in $K_0$
7  let $F$ be a forest, initially containing all edges of $X(K_0)$ except an arbitrary one
8  for $j = 1, 2, \ldots, q$ do
9      foreach edge $e$ of $X(K_j)$ do
10         add $e$ to $F$ if it does not introduce a cycle in $F$
11 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 $X(K_0)$
12 let $T^*$ be the spanning tree of $G^*$ rooted at $f_\infty$ that consists of edges not in $T$
13 let $e^*$ be a most balanced edge separator of $T^*$
14 if $e \in K_0 \setminus \bigcup_{1\le j\le q} K_j$ then return the fundamental cycle of $e$ w.r.t. $T$
15 let $j \ge 1$ be such that $e \in K_j$
16 let $H_1, H_2, \ldots, H_\ell$ be the subtrees of $T^*_e$ rooted at edges of $X(K_j)$
17 if $w(H_k) \ge W/4$ for some $k$ $(1 \le k \le \ell)$ then return the boundary of $H_k$
18 let $r$ be such that$W/4 \le w\left(K_j \cup \bigcup_{1\le k\le r} H_k\right) \le 3W/4$
19 return the boundary of $K_j \cup \bigcup_{1\le k\le r} H_k$
```

The pseudocode of our algorithm is listed in 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 $\mathcal{K}$. If any level cycle is a short and balanced cycle separator, it is returned (line 3).

Next, the algorithm finds a small range of levels where a short balanced cycle separator is guaranteed to exist (see appendix). The algorithm finds two levels, i_- and i_+, each with at most $\sqrt{m/2}$ edges (lines 4 and 5).[4]

Component K_0 is defined as the unique level i_- component enclosing a weight of at least $3W/4$. We denote the level-i_+ components contained in K_0 by $K_1, K_2, \ldots, K_q$. Roughly speaking, the algorithm finds a balanced fundamental cycle in K_0 and shortcuts it along one of the cycles $X(K_j)$ (refer to lines 13–19). Care must be taken to ensure that the resulting cycle is simple. This is ensured by meticulously computing a low-depth spanning tree and by appropriately defining how to shortcut the fundamental cycle along $X(K_j)$.

More precisely, the algorithm builds a forest F containing all edges of $X(K_0)$ except an arbitrary one and as many edges of cycles $X(K_i)$ as possible for each i, $1 \le i \le q$ (line 10). Note that this is not the same as adding all but one of the edges of each of the cycles $X(K_i)$ one after the other because these cycles are not necessarily vertex-disjoint (see, e.g., the situation depicted in the bottom left of Figure 3). In line 11, 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 this we mean that the BFS is seeded with the component T of F that contains the edges of $X(K_0)$. Whenever

[4]If no level meets the requirements for i_-, then $i_- = 1$. Note that, in this case, we have $|L^E_{i_-}| = 3 > \sqrt{m/2}$, so this can only happen when $m < 18$. If no valid i_+ exists, no components K_j are defined for $j \ge 1$.

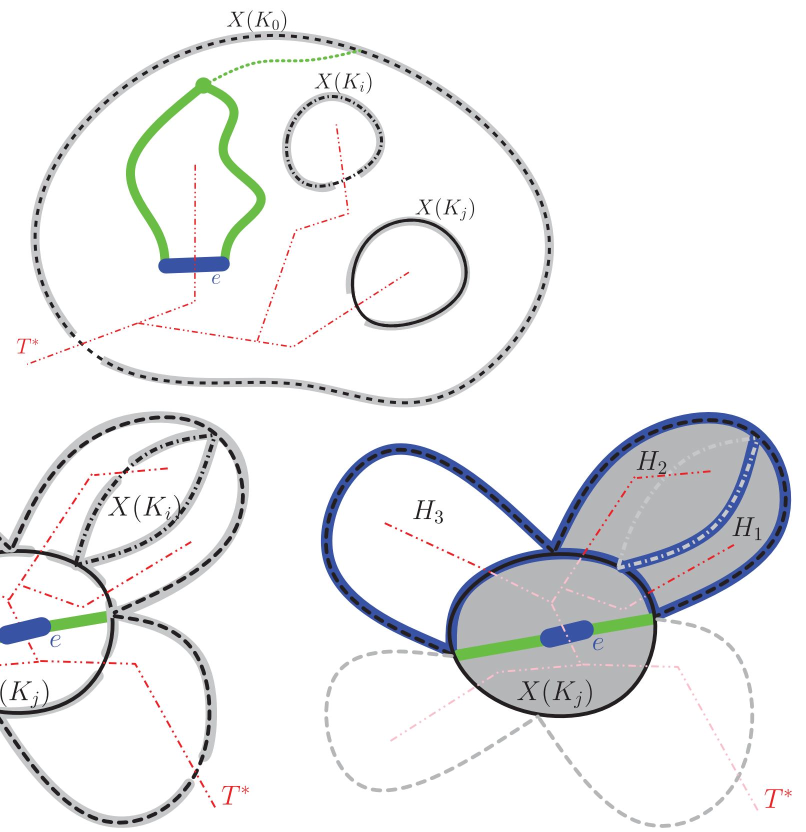


Fig. 3. Illustration of the algorithm. Top: a simple scenario. $X(K_0)$ is in thin dashed black. Two cycles at level i_+ are shown: $X(K_j)$ is in solid black, and $X(K_i)$ is black and dash-dotted. Edges of the forest F are highlighted in gray background. In this case, F includes all edges of the three depicted cycles except one in each cycle. Some edges of T^*, the dual spanning tree, are shown in dash-dot-dotted thin red. The blue edge e (an edge of T^*) induces a balanced fundamental cycle formed by edges of T, the primal tree (thick green). Some additional edges of T are in green dots. This fundamental cycle is short because it only contains edges of T between levels i_- and i_+. Bottom left: a more complicated scenario in which e is within the level i_+ component K_j. Here, the forest F (highlighted in gray) includes all edges of the three cycles except one of $X(K_0)$, five of $X(K_j)$, and one of $X(K_i)$. The parts of the fundamental cycle induced by e within K_j are in green (the rest of the fundamental cycle is composed of edges of F and is not highlighted). This fundamental cycle might be long because the level of e might be much greater than i_+. In this case, the balanced separator is constructed as illustrated in the bottom right figure. Bottom right: the parts of T^*_e not enclosed by K_j are shown in dash-dot-dotted red (other parts are in pink). The boundaries of regions H_1, H_2, and H_3 are indicated in solid blue. In this example, the simple cycle separator returned is indicated by the gray background: the cycle bounding $K_j \cup H_1 \cup H_2$.

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 exactly the edges not in T. The root of T^* is f_∞. It finds a most balanced edge separator e^* in T^*. If e belongs to K_0 but not to any K_j for $j \geq 1$, then the fundamental cycle of e w.r.t. T is returned (line 14). Otherwise, $e^* \in K_j$ for some $j \geq 1$ (it cannot be that $e^* \notin K_0$ 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. We partition the faces of T_e^* outside of K_j into connected subgraphs $H_1, H_2, \ldots, H_\ell$. If any H_k is a balanced separator, it is returned (line 17). Otherwise, there must be a prefix of the H_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 article.

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. [2009] is that the Fundamental Cycle Separator algorithm is usually very effective in finding small, balanced cycle separators. Our goal in this article is to establish that our algorithm, which *does* provide a worst-case guarantee on both separator size and balance, is competitive with FCS in terms of both runtime and average-case cycle size and balance. We do not directly compare our results with the other algorithms presented in Aleksandrov et al. [2007] and Holzer et al. [2009] because they do not produce simple cycle separators.

The FCS algorithm [Holzer et al. 2009] 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 leaves of T^* toward 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. Such a balanced cycle must exist since no single face (vertex of T^*) weighs more than $W/4$. The length of any fundamental cycle of T is bounded by one plus twice the diameter of T. In the worst case, however, there is no guarantee on the diameter of T, so a balanced fundamental cycle separator might not be short.

The algorithm described in Section 3 is guaranteed to return a short and balanced simple cycle separator. There is a tradeoff between balance and cycle length, and between those two properties and the running time of the algorithm. Depending on the application, one property may be more important than the other. We have implemented the following variants of our algorithm and of FCS.

—Fastest-Balanced: Terminates when the first balanced cycle separator is encountered. This variant does not guarantee short separators.
—Shortest-Balanced: Returns the shortest cycle separator among all balanced cycle separators encountered throughout the entire run of the algorithm. This variant guarantees balanced separators. For our algorithm, it also guarantees short separators, but for FCS it does not.
—Most-Balanced-Short: Returns the most balanced cycle separator among all short cycle separators encountered throughout the entire run of the algorithm. This variant guarantees short separators. For our algorithm, it also guarantees balanced separators, but for FCS it does not.
—Fastest-Short-and-Balanced: Terminates when the first short and balanced cycle separator is encountered. Note that FCS is not guaranteed to find such a separator.

Throughout our experiments, we regard any α-balanced separator with $\alpha \leq 2/3$ as a balanced separator, and any separator with at most $\sqrt{8|E|}$ as a short separator.

Other variants (Fastest-Short, Shortest, Most-Balanced, Sparsest) are also easy to implement.

Since different variants differ in the way a good separator is defined, the edge e^* in line 13 of our algorithm may be chosen according to different criteria, not necessarily the most balanced one. For example, the Most-Balanced-Short variant chooses e^* to be the one that induces the most balanced fundamental cycle separator among all edges of T^* that induce short cycles. The Shortest-Balanced variant chooses e^* to be the edge of T^* that induces the shortest fundamental cycle among all those that induce a balanced one. In a similar way, the choice of the best cycle to return in lines 18 and 19 changes between the different variants.

4.1. Datasets

To effectively compare our algorithms, we draw extensively from the graphs tested experimentally in Aleksandrov et al. [2007] and Holzer et al. [2009]. Each graph is triangulated before testing. Note that there is some degree of freedom in triangulation (Holzer et al. [2009] use the triangulation routines provided by LEDA). As our graphs are represented by permutations of the darts, we 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. To follow is a list of the classes of graphs:

(1) `grid` are square grid graphs; `rect` are rectangular grid graphs with about 20 times as many rows as columns.
(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 the 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 graph is essentially a narrow, length-k strip, triangulated in a way that maintains a diameter of k and a very small separator (cf. [Holzer et al. 2009, Figure 7]).
(7) The `airfoil` graph is a finite-element mesh of real-world data [Diekmann and Preis 1998].
(8) The graphs `BAY`, `CAL`, `COL`, `CTR`, `E`, `W`, `FLA`, `LKS`, `NE`, `NW`, `NY`, `USA` are road networks used in the 9th DIMACS Implementation Challenge—Shortest Paths [Demetrescu et al. 2008], accessible online [Demetrescu et al. 2006]. We interpret each graph and the coordinates as a straight-line embedding and we add vertices whenever two edges intersect geometrically.

4.2. Implementation Details

All tests are run on a machine with an Intel Xeon X5650 processor (six Hyper-Threaded cores for 12 execution threads) and 48.4 gigabytes of RAM. The code is compiled using GCC 4.4.5 targeting Intel x86_64. The operating system is Debian. Runtime tests are run single-threaded on an otherwise idle machine. Time is measured using `clock_gettime` provided by `time.h`. Instances tested are sufficiently large as to render clock granularity issues negligible.

Embedded graphs are represented by arrays, such that the value at index i is the identifier of the next dart in a primal or dual permutation from dart i. Additionally, we store a map from each vertex and face to some incident dart and a map from darts to the vertices and faces to which they are incident. In total, this consists of storing approximately $7.5n$ integers for an n-vertex graph.

Reusing a breadth-first search subroutine simplifies the implementation: we compute dual BFS levels for the component tree, span the primal, and generate the cotree using the same subroutine; the only difference is the function that determines whether the search should follow a particular edge. Our breadth-first search trees retain information about parents, children, and levels. The component tree stores a pointer to a representative face for each component and weight information about each component.

Subgraphs are represented by a mapping from vertices or faces to Boolean values; a value of true indicates that the given vertex or face is in the subgraph.

Our implementation constructs the component tree using a simple method backed by a disjoint set data structure in $O(n\alpha(n))$ time (here, $\alpha(n)$ is the inverse Ackermann function, which is at most 4 for any practical purposes [Tarjan 1975]). We build the component tree from the leaves, working rootward to propagate weights up as it is built. Each component stores the number of faces it encloses. Each face belongs to one leafmost component; we store a mapping from each face to this.

Our implementation of the FCS algorithm employs the same breadth-first-search mechanism used for our algorithm. The breadth-first-search mechanism is used to span the primal. Then we use a breadth-first search of the dual, avoiding primal-tree edges, to construct the cotree. We believe that using the same basic mechanisms to implement both algorithms makes the runtime comparison less likely to be biased.

4.3. Results and Interpretation

We compare our algorithm (indicated as “ALG” in figures) with FCS on runtime and resulting separator quality on a variety of graphs. For each graph, we sampled many possible faces as roots of the component trees for our algorithm, and possible vertices as roots of the primal BFS trees for FCS. Following Holzer et al. [2009], we use whisker plots to display the range of observed values. 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.

4.3.1. Running Time. Figure 4 shows running times of the Fastest-Short-and-Balanced and Most-Balanced-Short variants of our algorithms and those of the Fastest-Balanced and Most-Balanced-Short variants of FCS on square grids of increasing size. As theory predicts, the running time of each variant appears to scale linearly with the number of vertices. The Fastest-Short-and-Balanced variant of our algorithm is fastest (linear fit with slope 1.59), Fastest-Balanced FCS is slightly slower (slope 1.84), Most-Balanced-Short FCS is slightly slower than that (slope 2.08), and Most-Balanced-Short is slowest (slope 3.78). On square grids, there always exists a short and balanced level cycle. Therefore, the Fastest-Short-and-Balanced variant of our algorithm terminates after computing just a dual BFS and constructing (perhaps just part of) the component tree. FCS computes a primal BFS, then constructs the cotree (implemented via a dual BFS), and then computes the size and balance of fundamental cycles, working from the leaves of the cotree to the root. Depending on the variant, it either stops the moment the search first encounters a balanced cycle (Fastest-Balanced FCS) or returns the most balanced short cycle after completing the entire rootward computation (Most-Balanced-Short

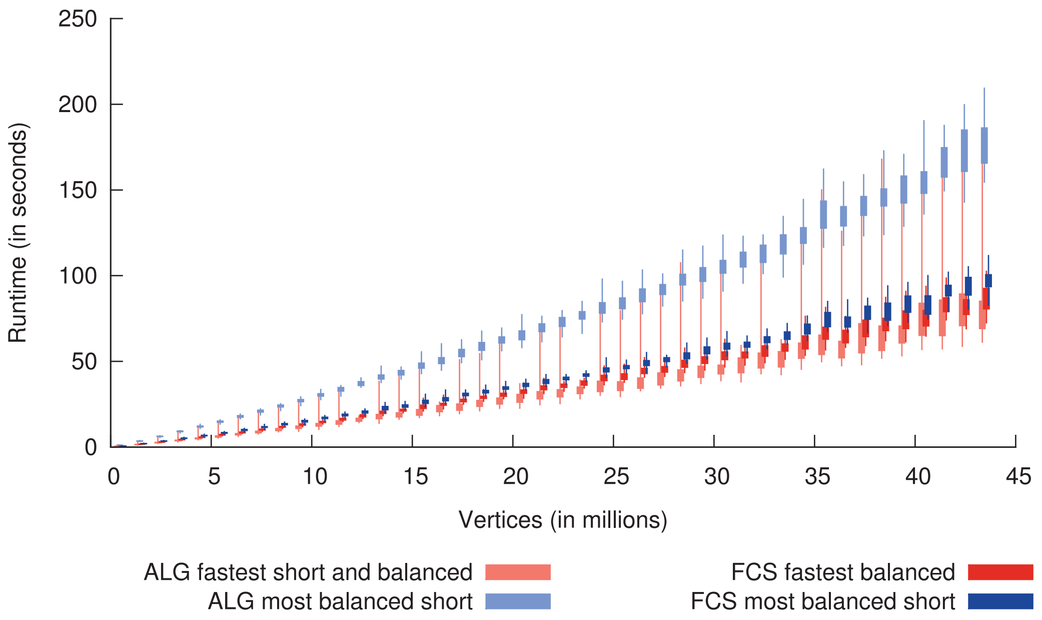


Fig. 4. Running time for our algorithm (Fastest-Short-and-Balanced and Most-Balanced-Short variants) and for FCS (Fastest-Balanced and Most-Balanced-Short variants). We offset the box plots for each graph size slightly for clarity; 60 random seeds are used for each trial in this plot.

FCS). The Most-Balanced-Short variant of our algorithm goes through all the steps of the algorithm and is therefore slower, though by a factor smaller than 3.

Similar behavior for other families of graphs is shown in Figure 5 (also showing the Shortest-Balanced variant for both our algorithm and FCS). All graphs reported in this figure have approximately 213,370 vertices (this is the number of vertices in the road network `NY`). Specifically, we test on a 462-by-462 `grid`, 100-by-2,133 `rect`, 462-by-462 `globe`, 10-by-21,337 `egg`, 21,337-by-5 `cylinder`, 327-square `hex`, and a `diameter`-71,225 graph. Although all graphs have essentially the same size, the execution times differ between different graph types. For example, the fast variant of our algorithm is slower for rectangular graphs (`rect`), and for road networks (`NY`). The reason is that for many seeds of the dual BFS of these graph types, no level cycle is short and balanced, so our algorithm must perform the other steps of the algorithm.

It is interesting that the running times of the Most-Balanced-Short and Shortest-Balanced variants of our algorithm vary across the different graph types. This cannot be attributed to the algorithm terminating at different stages since these variants always execute all steps of the algorithm. We suspect that the reason is due to different cache performance when the BFS tree is wide and shallow or thin and deep.

The running time of the same variants on all road networks as a function of the number of vertices is shown in Figure 6. As was the case for `NY` in Figure 5, our algorithm is typically slower. The reason is that our algorithm seldom finds a short and balanced level cycle and must perform the other steps of the algorithm.

Figure 7 shows the step that contributed the separator returned by our algorithm (for different variants). Recall that our algorithm can return a level cycle (line 3) when the component tree is computed, return a fundamental cycle with respect to the primal BFS tree (line 14), or return a repaired FCS, that is, a combination of a fundamental cycle and a level cycle (lines 17 or 19). The figure for the Fastest-Short-and-Balanced

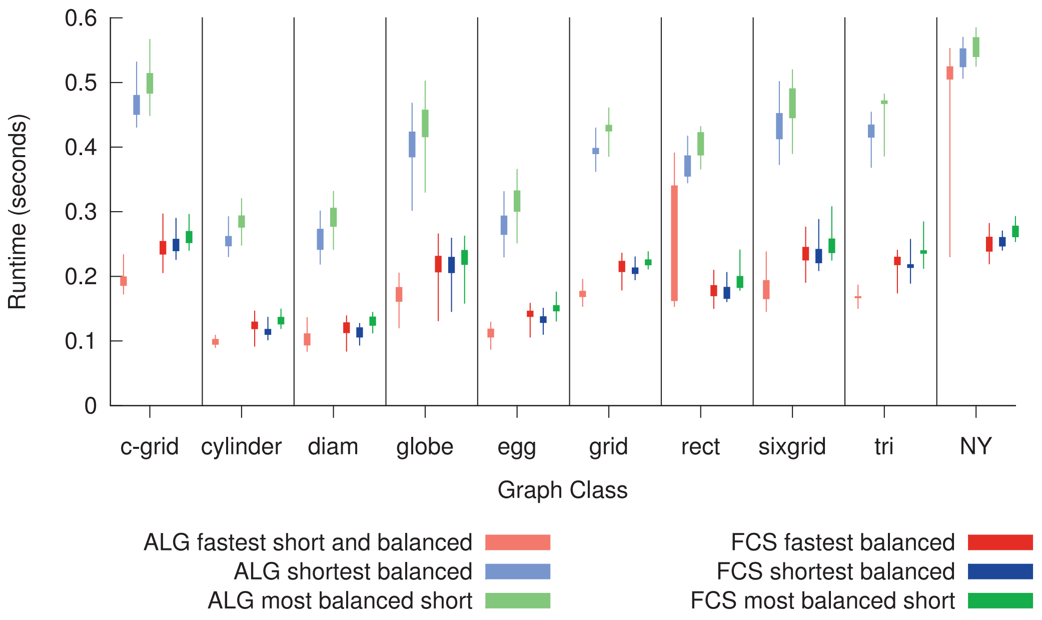


Fig. 5. Running time for our algorithm (Fastest-Short-and-Balanced, Shortest-Balanced, and Most-Balanced-Short variants) and for FCS (Fastest-Balanced, Shortest-Balanced, and Most-Balanced-Short variants) for various types of graphs, all with roughly the same number of vertices; 1,500 random seeds are used for each trial in this plot.

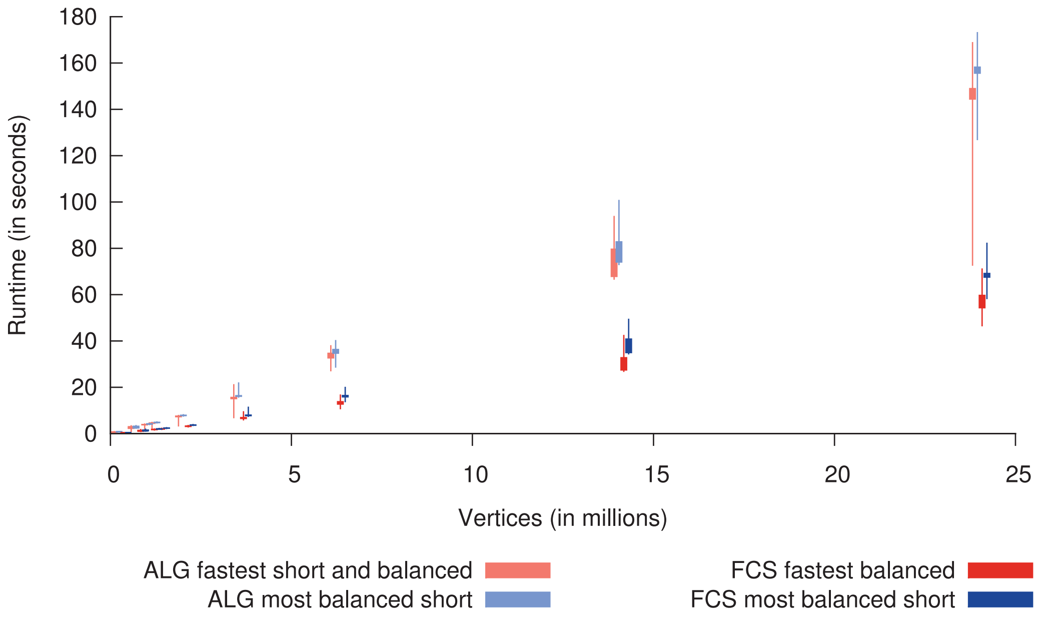


Fig. 6. Running time for our algorithm (Fastest-Short-and-Balanced and Most-Balanced-Short variants) and for FCS (Fastest-Balanced and Most-Balanced-Short variants) on various road networks as a function of the number of vertices. We offset the box plots for each road network slightly for clarity; 100 random seeds are used for each trial in this plot.

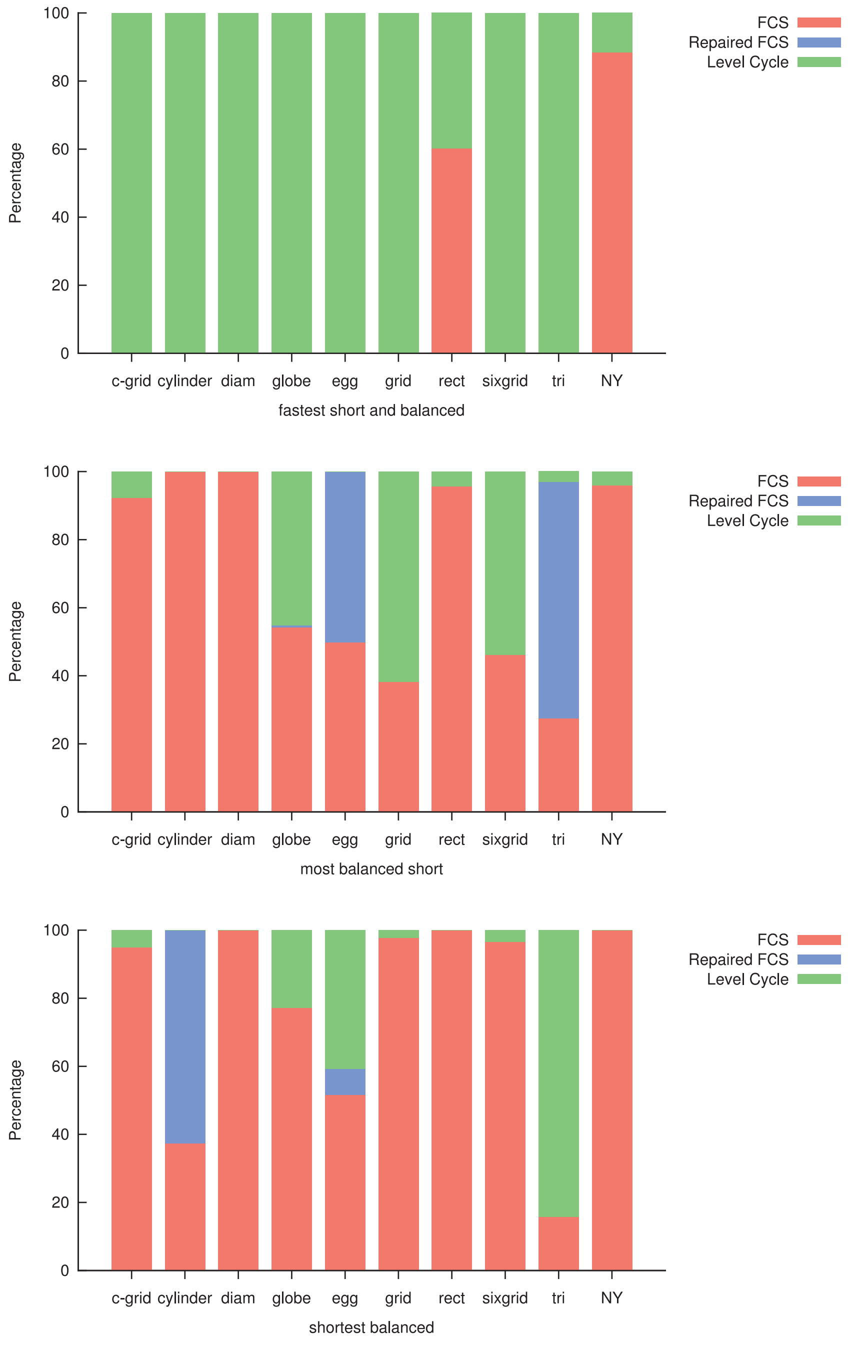


Fig. 7. The type of separator returned by three variants of our algorithm on various graph families. The variants are Fastest-Short-and-Balanced (top), Most-Balanced-Short (middle), and Shortest-Balanced (bottom); 1,500 random seeds are used for each trial in these plots.

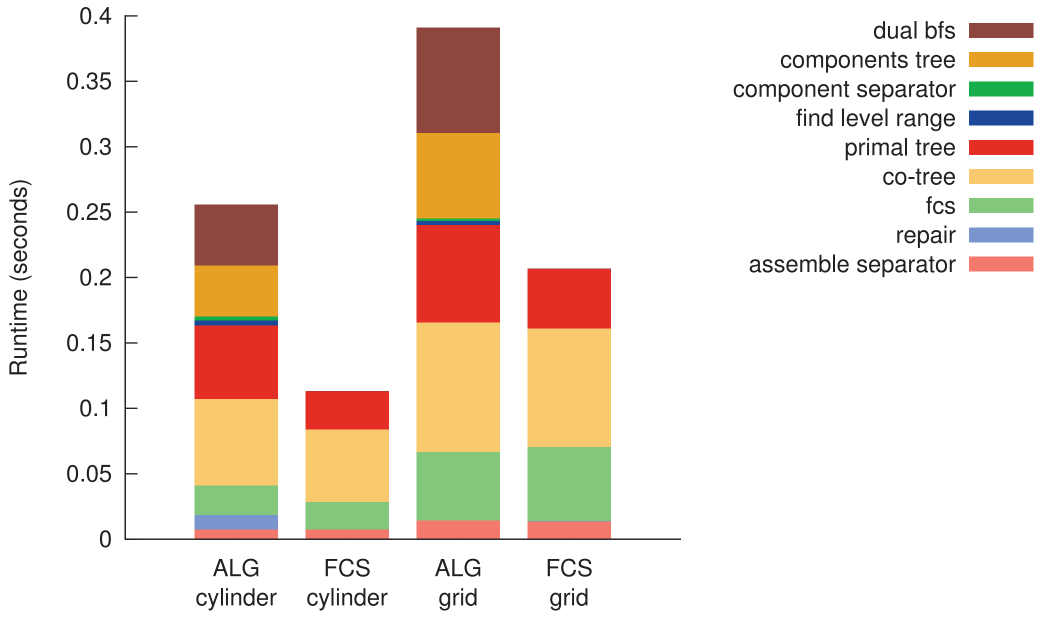


Fig. 8. Breakdown of running time of our algorithm and the FCS algorithm according to the average amount of time spent at each step of the algorithm. Times are shown for two different graphs: `grid` and `cylinder`; 1,500 seeds are used for each trial in this plot.

variant supports the explanation given earlier for the fast execution of this variant. On most graph families, this variant returns a level cycle, so it terminates immediately after generating the component tree. For `rect` and road networks, and to a lesser extent for `grid` and `sixgrid`, the first short and balanced separator encountered is in the fundamental cycle step, which takes additional time. The data for the Shortest-Balanced and Most-Balanced-Short variants shows that the shortest- or most-balanced separators are often produced by different steps that vary between the different graph families and variants. For example, the most-balanced short separator for the skewed `cylinder` graph is always produced by the FCS stage, while the shortest-balanced separator for the same graph is often a repaired FCS. For the `tri` graph, the most-balanced short separator is (almost always) either an FCS or a repaired FCS, but the shortest-balanced separator is usually a level cycle. The fact that in many instances level cycles still produce the best separator with the desired criterion suggests that considering just level cycles is an effective heuristic for producing short and balanced cycle separators. We note that this heuristic does not seem to be very effective on road networks.

The amount of time spent on each of the steps of our algorithm (Most-Balanced-Short variant) and of the FCS algorithm is shown in Figure 8. The steps reported are:

—dual BFS: spanning the dual,
—component tree: constructing the component tree (including the computation of the weight of each component and the length of its boundary (level cycle)),
—components separator: checking all level cycles to find if any of them is a short and balanced separator,
—find level range: identifying levels i_- and i_+,
—primal tree: computing the primal spanning tree T,
—cotree: computing the cotree T^*,

—fcs: finding a fundamental cycle separator,
—repair: repairing a long fcs, if necessary, and
—assemble separator: mark all faces inside the separator and trace boundary.

We use the Most-Balanced-Short variant of our algorithm since it is guaranteed to run through all the stages of the algorithm (results for Shortest-Balanced are nearly identical). We emphasize that while we took care that our implementation is efficient, we did not fully optimize our code. Hence, we refrain from making strong statements about one stage or algorithm being faster than another if the two times are of the same order of magnitude.

It is evident that the first three stages of our algorithm (dual BFS, component tree, and finding if any level cycle is good) run in about the same time as the entire FCS algorithm (in fact, slightly faster). This is consistent with the observation that our algorithm is slightly faster than FCS in instances where a good level cycle is returned. The primal BFS step of our algorithm is slower than that of FCS because of the additional complication arising from computing the forest F and extending it in a breadth-first-search manner (this is implemented by three concurrent searches with different priorities and restrictions on the edges used by each search).

We note that the time to compute the dual BFS is smaller than the time to compute the cotree of the primal BFS tree. This may seem unexpected since both procedures are implemented by a breadth-first search of the dual. The cotree computation takes longer since, for each dart d considered, it needs to check both that d is not in the primal BFS and that the head of d was not previously visited by the search. The dual BFS computation only needs to verify the latter condition.

Another observation is that computing the primal BFS (first step of the FCS algorithm) is faster than computing the dual BFS (first step of our algorithm). The reason for this is that, in triangulated planar graphs, the number of faces is twice the number of vertices. Hence, a dual spanning tree has twice as many edges as a primal one. While both primal and dual breadth-first searches consider each dart exactly once, the size of the queue used to implement the search is larger for the computation in the dual graph.

4.3.2. Separator Balance and Size. We next discuss the quality of the separators produced by our algorithm and by FCS. To this end we analyze the size (number of edges on the cycle separator) and balance of the separators produced on the same graphs used to produce Figure 5. Please refer to Figure 9.

As expected, we see that all variants of our algorithm return separators that are at least 1/3-balanced. This is not the case with the Most-Balanced-Short variant of FCS, which on certain graphs (`egg` and `cylinder`) sometimes fails to meet this balance guarantee (i.e., it finds no balanced fundamental cycle that is also short). We focus on these graphs in more detail later.

We observe that for both the Most-Balanced-Short and the Shortest-Balanced variants, our algorithm tends to find separators that are slightly more balanced than the corresponding variant of FCS. In particular, the Most-Balanced-Short variant of our algorithm usually returns a nearly perfectly balanced separator. The reason is that our algorithm is designed to construct a low-diameter primal spanning tree "around" the most balanced component in the component tree (through its choice of i_- and i_+).

Figure 10 shows that while our algorithm always produces balanced separators that are significantly shorter than the $\sqrt{8m}$ size guarantee, FCS typically produces somewhat shorter separators. The reason is that, in order to guarantee the worst-case behavior, our algorithm forces its fundamental cycles to use the small levels i_+ and i_-. This restriction is enforced even if the length of a fundamental cycle in an unrestricted

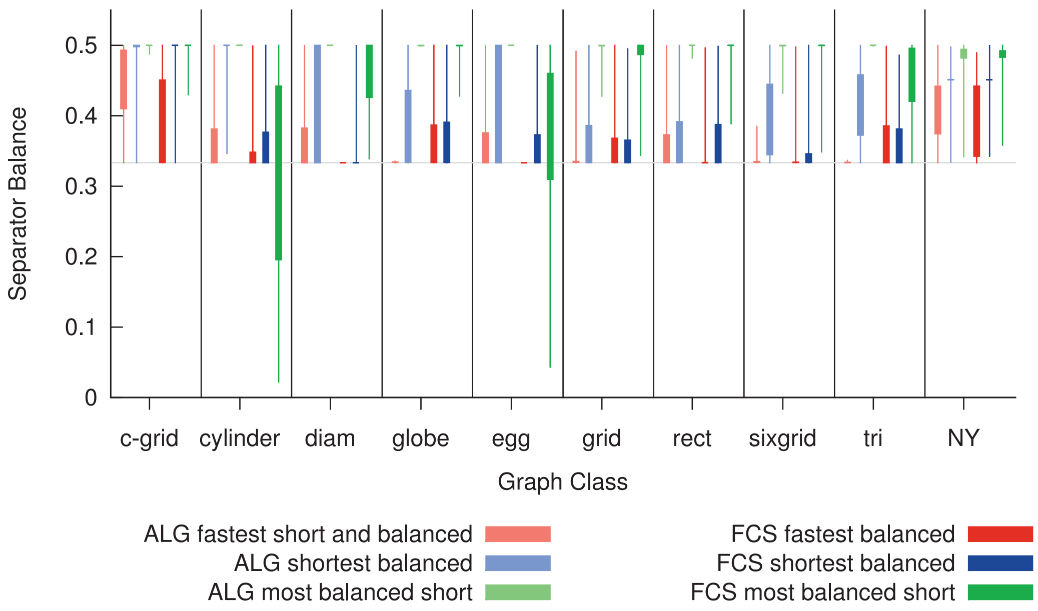


Fig. 9. Separator balance (number of vertices in smaller part divided by the total number of vertices) for the Fastest-Short-and-Balanced, Shortest-Balanced, and Most-Balanced-Short variants of our algorithm, and for the Fastest-Balanced, Shortest-Balanced, and Most-Balanced-Short variants of FCS; 1,500 random seeds are used for each trial in this plot.

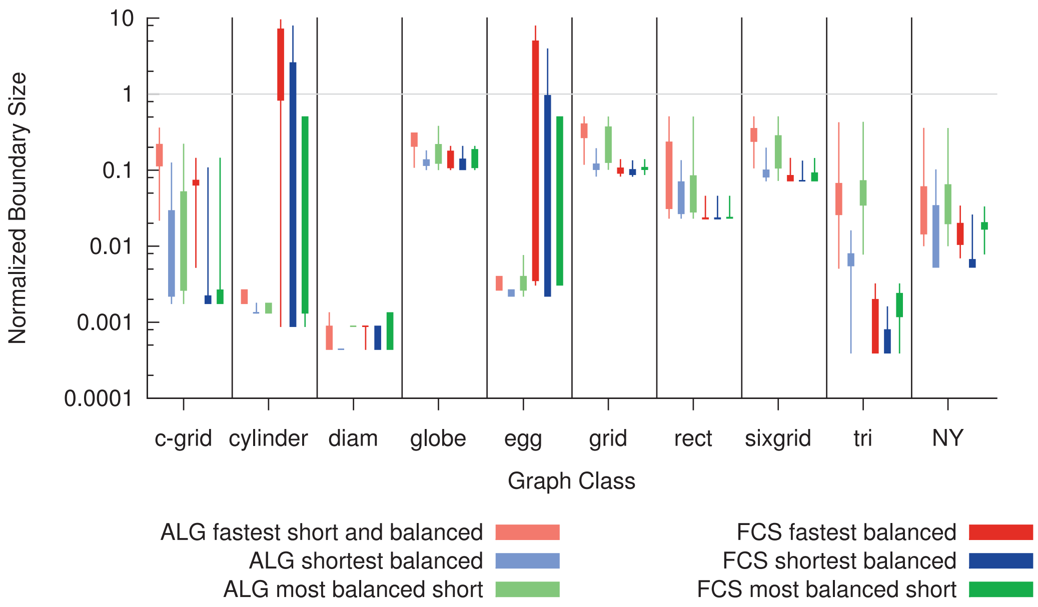


Fig. 10. Separator size divided by $\sqrt{m}$ for variants of our algorithm and FCS; 1,500 random seeds are used for each trial in this plot.

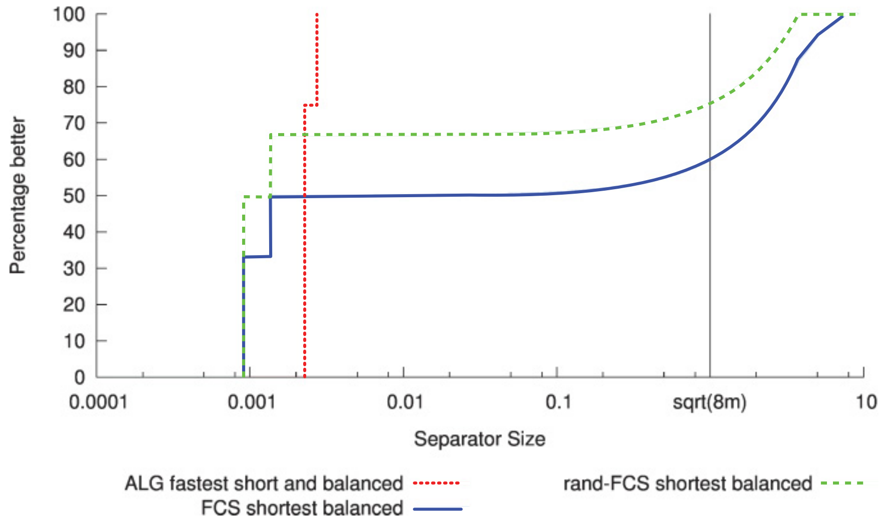


Fig. 11. X-axis: separator size. Y-axis: percentage of starting vertices achieving a separator of at most this size. Our algorithm (red), shortest-balanced randomized FCS (green), and shortest-balanced deterministic FCS (blue) are compared. We tested each possible starting face or vertex for each trial in this plot.

primal tree (i.e., the one used by the FCS algorithm) is not big. This situation is apparent in road networks, where the dual BFS is rather shallow, and level cycles tend to be large. For NY, for example, i_- is typically at most level 50, and the size of X_0 typically exceeds 350 ($\sqrt{m}$ for NY is 461.9). The average distance between vertices on i_- in the primal tree constructed by our algorithm is half the length of X_0 (i.e., 231), while in unrestricted primal BFS the distance would be at most i_- (i.e., 50), via a path that goes to the root level and back.

For egg and cylinder graphs, FCS often produces separators whose length is well above our size guarantee. 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 20,000-by-5 cylinder. The structure of this graph is such that balanced fundamental cycles of the primal BFS trees "travel" along the longer dimension of the cylinder and are thus long. Holzer et al. [2009] studied a randomized heuristic, wherein at each level of construction of the BFS tree, the order in which vertices are processed is chosen randomly. BFS trees constructed in this way are less susceptible to the phenomenon described earlier.

A graph of the cumulative frequency diagram for separator size is shown in Figure 11. The results are shown for the Fastest-Short-and-Balanced variant of our algorithm, and for the Shortest-Balanced variant of FCS (deterministic and randomized heuristic for the construction of the BFS tree). Our algorithm reaches 100% well before the guaranteed $\sqrt{8m}$ size bound, while FCS has many poor choices for the starting vertex. This implies that one might need to try FCS several times before locating a viable separator. However, especially for the shortest-balanced randomized version of FCS, the expected number of attempts is small (less than two).

5. CONCLUSIONS

In this article, we describe 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 tuned for returning a shortest-balanced FCS and employs randomization in the construction of the BFS tree, it finds a small separator with (experimentally observed) high probability. 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 tested 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, which 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 to construct the component tree based on a BFS of the radial graph of G (face-vertex incidence graph) instead of the dual of G, as was suggested by van Walderveen et al. [2013]. This has the effect that level cycles at different levels are vertex disjoint. Combined with our initialization of the primal tree T, this seems to guarantee that the balanced fundamental cycle of T is always short. Another interesting direction is implementing the separator algorithm in Klein et al. [2013], which is quite similar to the algorithm discussed in the current article. An implementation of the algorithm in Klein et al. [2013] 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.

APPENDIX

A. CORRECTNESS OF ALGORITHM 1

LEMMA A.1. *The cycle returned by the algorithm is simple.*

PROOF. The cycle returned in line 3 is simple since component boundaries are simple cycles. The cycle returned in line 14 is simple since it is a fundamental cycle. The cycle returned in line 17 is simple because each H_k is connected in the dual by definition, as is its complement. It remains to show that the boundary of $R = K_j \cup \bigcup_{1 \le k \le r} H_k$ is a simple cycle.

We now show that R is a simple cut in G^*, and thus, by Fact 1, that it is bounded by a simple cycle. Suppose, for the sake of contradiction, that R is not a simple cut. Then some set of faces A not in R must be separated from f_∞ by the boundary of R because the faces of R are connected by definition. By definition of the components tree $\mathcal{K}$, there is an f-to-f_∞ path for each $f \in A$ where the level of each face monotonically decreases. However, since neither f nor f_∞ is enclosed by R, this path must cross the boundary of R at least twice and an even number of times. In particular, the path must contain at least two edges on the boundary of R.

By construction, a face is in R if and only if its closest ancestor in T^* incident to an edge of $X(K_j)$ is in R. Contrapositively, a face is *not* in R if and only if it has no ancestor incident to an edge of $X(K_j)$ or if its closest ancestor incident to $X(K_j)$ is *not* in R. All faces in R are descendants in T^* of some face in K_j. If two faces are *not* descendants of any face in K_j, then the unique path in T^* connecting them does not enter R. As such, faces not in R with no ancestor incident to an edge of $X(K_j)$ are connected to f_∞ outside of R. Such faces, by definition, are not in A. Thus, each face in A has some ancestor in A and not in K_j but incident to an edge of $X(K_j)$. This means that there is some face $f \in A$ incident to an edge of $X(K_j)$ and thus with level $i_+ - 1$.

All faces of A are within K_0 since K_0 is bounded by a simple cycle and R is entirely within K_0. Thus, the first time a level-monotone f-to-f_∞ path encounters an edge bounding R, it is *not* an edge of $X(K_0)$; in particular, since R's boundary is a subset of the edges of F by construction, it must be a level i_+ edge. This precludes any f-to-f_∞ path from having monotonically decreasing levels, raising a contradiction. Thus, R must be bounded by a simple cut and by Fact 1 its boundary is a simple cycle. □

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

LEMMA A.2. *The simple cycle returned by the algorithm is* $3/4$*-balanced.*

PROOF. Clearly, the cycles returned in lines 3, 14, or 17 are balanced separators. It only remains to argue that if line 18 is reached, then there exists an r such that $K_j \cup \bigcup_{1\leq k\leq r} H_k$ is $3/4$-balanced. By construction of T, the fundamental cycle of e with respect to T is enclosed by $X(K_0)$. Since this fundamental cycle is a balanced separator (albeit not a short one), this implies that $w(K_j \cup \bigcup_{1\leq k\leq \ell} H_k) \geq W/4$. Since the condition in line 17 is false, $w(H_k) < W/4$ for each $1 \leq k \leq \ell$. Furthermore, by choice of i_+, $w(K_j) < W/4$. Hence, an appropriate r must exist. □

LEMMA A.3. *If* $m \geq 18$*, the forest* F *consists of at most* $\sqrt{2m} - 2$ *edges.*

PROOF. Each edge of F either has level i_- or level i_+. By construction, each of the levels i_- and i_+ consists of at most $\sqrt{m/2}$ edges. However, F cannot include all edges of either level as the edges on each level form at least one simple cycle in the primal. As such, F has at most $2(\sqrt{m/2} - 1) = \sqrt{2m} - 2$ edges. □

LEMMA A.4. *Suppose* r *is an arbitrary vertex of* $X(K_0)$ *and let* u *be a vertex in* K_0 *incident to a face* f *with level less than* i_+*. The* r*-to-*u *path in* T *consists of at most* $\sqrt{m/2} + 2$ *edges that do not belong to* F *and at most* $\sqrt{m/2} + 1$ *edges if* F *contains at least* $\sqrt{m/2}$ *edges.*

PROOF. By definition of face levels, there exists a face f' that is incident to $X(K_0)$ and whose distance in the dual graph from f is at most $i_+ - i_-$. Since each face is a triangle, this implies that there exists a path in G from u to some vertex v of $X(K_0)$ whose length is at most $\lceil (i_+ - i_-)/2 \rceil$. Since T is obtained from F by breadth-first search, the v-to-u path in T consists of at most $\lceil (i_+ - i_-)/2 \rceil$ edges not in F. F connects all of $X(K_0)$, so there is an r-to-v path using only edges of F.

It remains to bound $i_+ - i_-$. Since the cycles bounding different components are edge disjoint, and since every level strictly between i_- and i_+ consists of at least $\sqrt{m/2}$ edges, $(i_+ - i_- - 1)\sqrt{m/2} \leq m - |F|$. This shows that $i_+ - i_- \leq \sqrt{2m} - |F|/\sqrt{m/2} + 1$. Hence, the root-to-$u$ path in T consists of at most $\lceil (\sqrt{2m} - |F|/\sqrt{m/2} + 1)/2 \rceil \leq \sqrt{m/2} - \lfloor |F|/\sqrt{m/2} \rfloor + 2$ edges that are not in F. □

LEMMA A.5. *The boundary of* H_k*,* $1 \leq k \leq \ell$*, consists solely of edges of* F*, with the exception of a single edge of* $X(K_j)$ *not in* F*.*

Proof. H_k is defined to be the subtree $T^*_{e'}$ for some edge $e' \in X(K_j)$. By Facts 1 and 2, the boundary of H_k is the fundamental cycle of e' with respect to T. Let u and v be the endpoints of e' in G. Since u and v belong to $X(K_j)$, u and v must belong to the same component of F. Therefore, the path in T between u and v must consist only of edges of F. □

Theorem A.6. *The algorithm returns a* 3/4*-balanced simple cycle separator with at most* $\sqrt{8m}$ *edges on graphs when* $m \geq 29$.

Proof. Lemma A.1 shows that the cycle returned by the algorithm is simple. Lemma A.2 shows that it is a 3/4-balanced separator. It remains to bound the length of the cycle.

The cycle returned in line 3 consists of at most $\sqrt{8m}$ edges by definition.

Root T at an arbitrary vertex of $X(K_0)$. The fundamental cycle returned in line 14 consists of at most all the edges in F plus all the edges not in F along the two paths from the endpoints of e to the root of T. Since e is within K_0 but not within K_i, $0 < i \leq q$, its endpoints are both incident to faces with levels less than i_+. Using Lemmas A.4 and A.3, this is at most $(\sqrt{m/2} - 1) + 2(\sqrt{m/2} + 2)$ in the case when $|F| < \sqrt{m/2}$ and $(\sqrt{2m} - 2) + 2(\sqrt{m/2} + 1))$ otherwise. In either case, this is at most $\sqrt{8m}$ when $m \geq 29$.

Suppose line 17 returns the boundary of H_k. By Lemma A.5, all except one edge bounding H_k is in F. F's size is at most $\sqrt{2m} - 2$, so H_k's boundary has at most $\sqrt{2m} - 1 < \sqrt{8m}$ edges.

An overestimate on the edges of a cycle returned in line 19 is the set of all edges bounding any H_k or K_j. By Lemmas A.5 and A.3, this totals $|F| + |X(K_j)| \leq \sqrt{2m} - 2 + \sqrt{m/2} < \sqrt{8m}$ edges. □

ACKNOWLEDGMENTS

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 David Eisenstat and Philip Klein for providing us with their planar graph library.

E.F. was partially supported by NSF grants CCF-0964037, CCF-1409520, and CCF-9640347. C.S. was partially supported by the Swiss National Science Foundation. S.M. was partially supported by National Science Foundation grants CCF-0964037 and CCF-1111109, and by an Israeli Science Foundation grant ISF-749/13.

REFERENCES

Lyudmil Aleksandrov, Hristo Djidjev, Hua Guo, and Anil Maheshwari. 2007. Partitioning planar graphs with costs and weights. *ACM Journal of Experimental Algorithmics* 11, Article 1.5 (Feb. 2007). Announced at ALENEX 2002.

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

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

Punyashloka Biswal, James R. Lee, and Satish Rao. 2010. Eigenvalue bounds, spectral partitioning, and metrical deformations via flows. *Journal of the ACM* 57, 3 (2010), 13:1–13:23. Announced at FOCS 2008.

Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. 2011. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. In *52nd IEEE Symposium on Foundations of Computer Science (FOCS'11)*. 170–179.

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

Camil Demetrescu, Andrew V. Goldberg, and David Johnson. 2006. 9th DIMACS Implementation Challenge—Shortest Paths. (2006). http://www.dis.uniroma1.it/challenge9/download.shtml; accessed October 21, 2012.

Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson. 2008. Implementation challenge for shortest paths. In *Encyclopedia of Algorithms*, Ming-Yang Kao (Ed.). Springer-Verlag New York.

Ralf Diekmann and Robert Preis. 1998. (1998). http://www2.cs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/GRAPHS/FEM2.tar; accessed October 21, 2012.

Hristo N. Djidjev. 1982. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic Discrete Methods* 3, 2 (1982), 229–240.

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

Hristo N. Djidjev and Shankar M. Venkatesan. 1997. Reduced constants for simple cycle graph separation. *Acta Informatica* 34 (1997), 231–243.

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

Lamis M. Farrag. 1998. *Applications of Graph Partitioning Algorithms to Terrain Visibility and Shortest Path Problems*. Master's thesis. School of Computer Science, Carleton University.

Eli Fox-Epstein, Shay Mozes, Phitchaya M. Phothilimthana, and Christian Sommer. 2013. Short and simple cycle separators in planar graphs. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*. Society for Industrial and Applied Mathematics, 26–40.

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

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

John R. Gilbert, Joan P. Hutchinson, and Robert E. Tarjan. 1984. A separator theorem for graphs of bounded genus. *Journal of Algorithms* 5, 3 (1984), 391–407. Announced as TR82-506 in 1982.

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

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

Martin Holzer, Frank Schulz, Dorothea Wagner, Grigorios Prasinos, and Christos D. Zaroliagis. 2009. Engineering planar separator algorithms. *ACM Journal of Experimental Algorithmics* 14 (2009), 5:1.5–5:1.31.

Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. 2011. Improved algorithms for min cut and max flow in undirected planar graphs. In *43rd ACM Symposium on Theory of Computing (STOC'11)*. 313–322.

Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. 2011. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *38th International Colloquium on Automata, Languages and Programming (ICALP'11)*. 135–146.

Ken-ichi Kawarabayashi and Bruce A. Reed. 2010. A separator theorem in minor-closed classes. In *51st IEEE Symposium on Foundations of Computer Science (FOCS'10)*. 153–162.

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

Philip N. Klein and Shay Mozes. 2013. Optimization Algorithms for Planar Graphs. http://www.planarity.org. (Forthcoming). Accessed April 2013.

Philip N. Klein, Shay Mozes, and Christian Sommer. 2013. Structured recursive separator decompositions for planar graphs in linear time. In *45th ACM Symposium on Theory of Computing (STOC'13)*. 505–514.

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

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

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

Jakub Łącki and Piotr Sankowski. 2011. Min-cuts and shortest cycles in planar graphs in $O(n \log\log n)$ time. In *19th European Symposium on Algorithms (ESA'11)*. 155–166.

Gary L. Miller. 1986. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences* 32, 3 (1986), 265–279.

Shay Mozes and Christian Sommer. 2012. Exact distance oracles for planar graphs. In *23rd ACM-SIAM Symposium on Discrete Algorithms (SODA'12)*. 209–222.

Shay Mozes and Christian Wulff-Nilsen. 2010. Shortest paths in planar graphs with real lengths in $O(n \log^2 n/\log\log n)$ time. In *18th Annual European Symposium on Algorithms (ESA'10)*.

Serge A. Plotkin, Satish Rao, and Warren D. Smith. 1994. Shallow excluded minors and improved graph decompositions. In *5th ACM-SIAM Symposium on Discrete Algorithms (SODA'94)*. 462–470.

Bruce A. Reed and David R. Wood. 2009. A linear-time algorithm to find a separator in a graph excluding a minor. *ACM Transactions on Algorithms* 5, 4 (2009), 39:1–39:16. Announced at EuroComb 2005.

Daniel A. Spielman and Shang-Hua Teng. 1996. Disk packings and planar separators. In *12th Symposium on Computational Geometry (SoCG'96)*. 349–358.

Robert E. Tarjan. 1975. Efficiency of a good but not linear set union algorithm. *Journal of the ACM* 22, 2 (April 1975), 215–225. DOI: http://dx.doi.org/10.1145/321879.321884

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

Freek van Walderveen, Norbert Zeh, and Lars Arge. 2013. Multiway simple cycle separators and I/O-efficient algorithms for planar graphs. In *24th ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*. ACM, 901–918.

Karl G. C. von Staudt. 1847. *Geometrie der Lage*. Bauer und Raspe, Nürnberg.

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

Christian Wulff-Nilsen. 2011. Separator theorems for minor-free and shallow minor-free graphs with applications. In *52nd IEEE Symposium on Foundations of Computer Science (FOCS'11)*. 37–46.

Received April 2013; revised May 2016; accepted June 2016