Decremental SPQR-trees for Planar Graphs

Jacob Holm, Giuseppe F Italiano, Adam Karczmarz, Jakub Łącki, Eva Rotenberg

Decremental SPQR-trees for Planar Graphs

Jacob Holm[1]
University of Copenhagen, Denmark
jaho@di.ku.dk
http://orcid.org/0000-0001-6997-9251

Giuseppe F. Italiano[2]
University of Rome Tor Vergata, Italy
giuseppe.italiano@uniroma2.it
https://orcid.org/0000-0002-9492-9894

Adam Karczmarz[3]
University of Warsaw, Poland
a.karczmarz@mimuw.edu.pl
https://orcid.org/0000-0002-2693-8713

Jakub Łącki[4]
Google Research, USA
jlacki@google.com
https://orcid.org/0000-0001-9347-0041

Eva Rotenberg
Technical University of Denmark, Denmark
erot@dtu.dk
http://orcid.org/0000-0001-5853-7909

Abstract

We present a decremental data structure for maintaining the SPQR-tree of a planar graph subject to edge contractions and deletions. The update time, amortized over $\Omega(n)$ operations, is $O(\log^2 n)$.

Via SPQR-trees, we give a decremental data structure for maintaining 3-vertex connectivity in planar graphs. It answers queries in $O(1)$ time and processes edge deletions and contractions in $O(\log^2 n)$ amortized time. This is an exponential improvement over the previous best bound of $O(\sqrt{n})$ that has stood for over 20 years. In addition, the previous data structures only supported edge deletions.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms, Theory of computation → Graph algorithms analysis, Theory of computation → Data structures design and analysis

Keywords and phrases Graph embeddings, data structures, graph algorithms, planar graphs, SPQR-trees, triconnectivity.

[1] Jacob Holm is supported by Mikkel Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.
[2] Giuseppe F. Italiano is partially supported by the Italian Ministry of Education, University and Research under Project AMANDA (Algorithmics for MAssive and Networked DAta).
[3] Adam Karczmarz is supported by the grants 2014/13/B/ST6/01811 and 2017/24/T/ST6/00036 of the Polish National Science Center.
[4] When working on this paper Jakub Łącki was partly supported by the EU FET project MULTIPLEX no. 317532 and the Google Focused Award on "Algorithms for Large-scale Data Analysis" and Polish National Science Center grant number 2014/13/B/ST6/01811. Part of this work was done while Jakub Łącki was visiting the Simons Institute for the Theory of Computing.

1 Introduction

A graph algorithm is called *dynamic* if it is able to answer queries about a given property while the graph is undergoing a sequence of updates, such as edge insertions and deletions. It is *incremental* if it handles only insertions, *decremental* if it handles only deletions, and *fully dynamic* if it handles both insertions and deletions. In designing dynamic graph algorithms, one is typically interested in achieving fast query times (either constant or polylogarithmic), while minimizing the update times. The ultimate goal is to perform fast both queries and updates, i.e., to have both query and update times either constant or polylogarithmic. So far, the quest for obtaining polylogarithmic time algorithms has been successful only in few cases. Indeed, efficient dynamic algorithms with *polylogarithmic* time per update are known only for few problems, such as dynamic connectivity, 2-connectivity, minimum spanning tree and maximal matchings in undirected graphs (see, e.g., [6, 24, 25, 28, 36, 52, 54, 56]). On the other hand, some dynamic problems appear to be inherently harder. For example, the fastest known algorithms for basic dynamic problems, such as reachability, transitive closure, and dynamic shortest paths have only *polynomial* times per update (see, e.g., [9, 10, 11, 38, 49, 51, 55]).

A similar situation holds for planar graphs where dynamic problems have been studied extensively, see e.g. [3, 14, 16, 18, 20, 21, 26, 33, 42, 43, 44, 45, 53]. Despite this long-time effort, the best algorithms known for some basic problems on planar graphs, such as dynamic shortest paths and dynamic planarity testing, still have polynomial update time bounds. For instance, for fully dynamic shortest paths on planar graphs the best known bound per operation is[5] $\widetilde{O}(n^{2/3})$ amortized [19, 33, 35, 39], while for fully dynamic planarity testing the best known bound per operation is $O(\sqrt{n})$ amortized [16].

In the last years, this exponential gap between polynomial and polylogarithmic bounds has sparkled some new exciting research. On one hand, it was shown that there are dynamic graph problems, including fully dynamic shortest paths, fully dynamic single-source reachability and fully dynamic strong connectivity, for which it may be difficult to achieve subpolynomial update bounds. This started with the pioneering work by Abboud and Vassilevska-Williams [2], who proved conditional lower bounds based on popular conjectures. Very recently, Abboud and Dahlgaard [1] proved polynomial update time lower bounds for dynamic shortest paths also on planar graphs, again based on popular conjectures.

On the other hand, the question of improving the update bounds from polynomial to polylogarithmic, has, for several other dynamic graph problems, received much attention in the last years. For instance, there was a very recent improvement from polynomial to polylogarithmic bounds for decremental single-source reachability (and strongly connected components) on planar graphs: more precisely, the improvement was from $O(\sqrt{n})$ amortized [42] to $O(\log^2 n \log\log n)$ amortized [32] (both amortizations are over sequences of $\Omega(n)$ updates). Other problems that received a lot of attention are fully dynamic connectivity and minimum spanning tree in general graphs. Up to very recently, the best worst-case bound for both problems was $O(\sqrt{n})$ per update [15]: since then, much effort has been devoted towards improving this bound (see e.g., [36, 37, 47, 48, 57]).

In this paper, we follow the ambitious goal of achieving polylogarithmic update bounds for dynamic graph problems. In particular, we show how to improve the update times from polynomial to polylogarithmic for another important problem on planar graphs: decremental 3-vertex connectivity. Given a graph $G = (V, E)$ and two vertices $x, y \in V$ we say that x and y are 2-vertex connected (or, as we say in the following, *biconnected*) if there are at least two

[5] Throughout the paper, we use the notation $\widetilde{O}(f(n))$ to hide polylogarithmic factors.

vertex-disjoint paths between x and y in G. We say that x and y are 3-vertex connected (or, as we say in the following, *triconnected*) if there are at least three vertex-disjoint paths between x and y in G. The decremental planar triconnectivity problem consists of maintaining a planar graph G subject to an arbitrary sequence of edge deletions, edge contractions, and query operations which test whether two arbitrary input vertices are triconnected. We remark that decremental triconnectivity on planar graphs is of particular importance. Apart from being a fundamental graph property, a triconnected planar graph has only one planar embedding, a property which is heavily used in graph drawing, planarity testing and testing for isomorphism [30, 31, 34]. Furthermore, our extended repertoire of operations, which includes edge contractions, contains all operations needed to obtain a graph minor, which is another important notion for planar graphs.

While polylogarithmic update bounds for decremental 2-edge and 3-edge connectivity, and for decremental biconnectivity on planar graphs have been known for more than two decades [20], decremental triconnectivity on planar graphs presents some special challenges. Indeed, while connectivity cuts for 2-edge and 3-edge connectivity, and for biconnectivity have simple counterparts in the dual graph or in the vertex-face graph (see Section 2 for a formal definition of vertex-face graph), triconnectivity cuts (separation pairs, i.e., pairs of vertices whose removal disconnects the graph) have a much more complicated structure in planar graphs. Roughly speaking, maintaining 2-edge and 3-edge connectivity cuts in a planar graph under edge deletions corresponds to maintaining respectively self-loops and cycles of length 2 (pairs of parallel edges) in the dual graph under edge contractions. On the other side, maintaining biconnectivity and triconnectivity cuts in a planar graph under edge deletions corresponds to maintaining, respectively, cycles of length 2 and cycles of length 4 in the vertex-face graph. While detecting cycles of length 2 boils down to finding duplicates in the multiset of all edges, detecting cycles of length 4 under edge contractions is far more complex. We believe that this is the reason why designing a fast solution for decremental triconnectivity on planar graphs has been an elusive goal, and the best bound known of $O(\sqrt{n}\,)$ per update [17] has been standing for over two decades.

Our results and techniques. In this paper, we show how to solve the decremental triconnectivity problem on planar graphs in constant time per query and $O(\log^2 n)$ amortized time per edge deletion or contraction, over any sequence of $\Omega(n)$ deletions and contractions. This is an exponential speed-up over the previous $O(\sqrt{n}\,)$ long-standing bound [17]. To obtain our bounds, we also need to solve decremental biconnectivity on planar graphs in constant time per query and $O(\log^2 n)$ amortized time per edge deletion or contraction. (A better $O(\log n)$ amortized bound can be obtained if no contractions are allowed [26]). Our results are achieved with the help of two new tools, which may be of independent interest.

The first tool is an algorithm for efficiently detecting and reporting cycles of length 4 as they arise in a dynamic planar embedded graph subject to edge contractions and insertions. The algorithm works for a graph with bounded face-degree, i.e, where each face is delimited by at most some constant number of edges. Specifically, given a plane embedded graph with bounded face-degree subject to edge-contractions and edge-insertions across a face, after each dynamic operation we can report all edges that lie on a length-4 cycle because of this dynamic operation. The total running time is $O(n \log n)$. One of the challenges that we face is that a planar graph may have as many as $\Omega(n^2)$ distinct cycles of length 4. Still, we give a surprisingly simple algorithm for solving this problem. The difficulty of the algorithm lies in the analysis — in fact, this analysis is the most technically involved part of this paper.

The second tool is a new data structure that maintains the SPQR-tree [12] of a planar

graph, while the graph is updated with edge deletions and edge contractions, in $O(\log^2 n)$ amortized time per operation. While incremental algorithms for maintaining the SPQR tree were known for more than two decades [12, 13], to the best of our knowledge no decremental algorithm was previously known.

Organization of the paper. The remainder of the paper is organized as follows. In Section 2, we introduce notation and definitions that we later use. Then, in Section 3 we present a high-level overview of our results. Section 4 presents our new algorithm for maintaining an SPQR-tree during edge deletions and contractions.

Due to space constraints, an algorithm for detecting cycles of length 4 under contractions, which is a key tool in maintaining an SPQR tree is described in Appendix B. Moreover, the detailed discussion of how to use the SPQR-trees to maintain information about triconnectivity is deferred to Appendix A. Finally, the proofs omitted from Section 4 are given in Appendix C.

2 Preliminaries

Throughout the paper we use the term *graph* to denote an undirected *multigraph*, that is we allow the graphs to have parallel edges and self-loops. Formally, each edge e of such a graph is a pair $(\{u, w\}, \mathrm{id}(e))$ consisting of a pair of vertices and a unique *integer identifier* used to distinguish between the parallel edges. For simplicity, in the following we skip the identifier and use just uw to denote one of the edges connecting vertices u and w. If the graph contains no parallel edges and no self-loops, we call it *simple*.

Given a graph G, we use $V(G)$ to denote the vertices, and $E(G)$ to denote the edges of G. For any $X \subseteq V(G)$ let $G[X]$ denote the subgraph $(X, \{(\{u, v\}, l) \in E(G) \mid u, v \in X\})$ of G *induced by* X.

The *components* of a graph G are the minimal subgraphs $H \subseteq G$ such that for every edge $uv \in E(G)$, $u \in V(H)$ if and only if $v \in V(H)$. The components of a graph partition the vertices and edges of the graph. A graph G is *connected* if it consists of a single component. For a positive integer k, a graph is *k-vertex connected* if and only if it is connected, has at least k vertices, and stays connected after removing any set of at most $k-1$ vertices. The *local vertex connectivity* of a pair of vertices u, v, denoted $\kappa(u, v)$, is the maximal number of internally vertex-disjoint u, v-paths. By Menger's Theorem [46], G is k-vertex connected if and only if $\kappa(u, v) \geq k$ for every pair of non-adjacent vertices u, v. We say that u, v are (locally) k-vertex connected if $\kappa(u, v) \geq k$. We follow the common practice of using *biconnected* as a synonym for 2-vertex connected and *triconnected* as a synonym for 3-vertex connected. An articulation point v of G is a vertex whose removal disconnects G. Thus a graph is biconnected if and only if it has no articulation points.

Let G be a graph and $e \in E(G)$. We use $G - e$ to denote the graph obtained from G by removing e. If e is not a self-loop, we use G/e to denote the graph obtained by contracting e. A *cycle* C of length $|C| = k$ in a graph G is a cyclic sequence of edges $C = e_1, e_2 \ldots, e_k$ where $e_i = u_i u_{i+1}$ for $1 \leq i < k$ and $e_k = u_k u_1$. A cycle is *simple* if $\mathrm{id}(e_i) \neq \mathrm{id}(e_j)$ and $u_i \neq u_j$ for $i \neq j$. We sometimes abuse notation and treat a cycle as a set of edges or a cyclic sequence of vertices. Note that this definition allows cycles of length 1 (a self-loop) or 2 (a pair of parallel edges).

Let G be a planar embedded graph. For each component H of G, let H^* denote the dual graph of H, defined as the graph obtained by creating a vertex for each face in the embedding of H, and an edge e^* (called the *dual edge* of e), connecting the two (not necessarily distinct) faces that e is incident to. Let G^* denote the graph obtained from G by taking the dual of

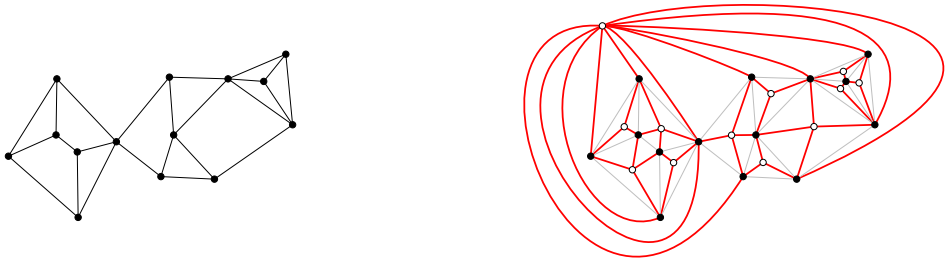

Figure 1 Left: a plane embedded graph. Right: the corresponding vertex-face graph (red) and the underlying graph (gray).

each component.

Each face f in a planar embedded graph is bounded by a (not necessarily simple) cycle called the *face cycle* for f. We call the length of this cycle the *face-degree* of f. We call any other cycle a *separating cycle*.

Let G be a connected planar embedded multigraph with at least one edge. Define the set $E^\diamond(G)$ of *corners*[6] of G to be the the set of ordered pairs of (not necessarily distinct) edges (e_1, e_2) such that e_1 immediately precedes e_2 in the clockwise order around some vertex, denoted $v(e_1, e_2)$. Note that if $(e_1, e_2) \in E^\diamond(G)$, then $(e_2^*, e_1^*) \in E^\diamond(G^*)$. We denote by $G^\diamond$ the *vertex-face* graph[7] of G (see Figure 1). This is a plane embedded multigraph with vertex set $V(G) \cup V(G^*)$, and an edge between $v(e_1, e_2)$ and $v(e_2^*, e_1^*)$ for each corner $(e_1, e_2) \in E^\diamond(G)$. Abusing notation slightly, we can write $G^\diamond$ as $= (V(G) \cup V(G^*), E^\diamond(G))$. We use the following well-known facts about the vertex-face graph:

1. $G^\diamond$ is bipartite and planar, with a natural embedding given by the embedding of G.
2. The vertex-face graphs of G and G^* are the same: $G^\diamond = (G^*)^\diamond$.
3. There is a one-to-one correspondence between the edges of G and the faces of $G^\diamond$ (in the natural embedding, each face of $G^\diamond$ contains exactly one edge of G interior, see Fig 1).
4. $(G^\diamond)^*$ (also known as the *medial graph*) is 4-regular.
5. $G^\diamond$ is simple if and only if G is loopless and biconnected (See e.g. [8, Theorem 5(i)]).
6. $G^\diamond$ is simple, triconnected and has no separating 4-cycles if and only if G is simple and triconnected (See e.g. [8, Theorem 5(iv)]).

If v is an articulation point in G or has a self-loop, then in any planar embedding of G there is at least one face f whose face cycle contains v at least twice. Any such f is either an articulation point or has a self-loop in G^*, and v and f are connected by (at least) two edges in $G^\diamond$.

The dynamic operations on G correspond to dynamic operations on G^* and $G^\diamond$. Deleting a non-bridge edge e of G corresponds to contracting e^* in G^*, that is $(G - e)^* = G^*/e^*$. Similarly, contracting an edge e corresponds to deleting the corresponding edge from the dual, so $(G/e)^* = G^* - e^*$. Finally, deleting a non-bridge edge or contracting an edge corresponds to adding and then immediately contracting an edge across a face of $G^\diamond$ (and removing two duplicate edges).

The useful concept of a separation is well-defined, even for general graphs:

[6] For alternative definitions, see e.g. [27] and [50]. The latter uses the name *angles* for what we call corners.

[7] A.k.a. the *vertex-face incidence graph* [7], the *angle graph* [50], and the *radial graph* [5].

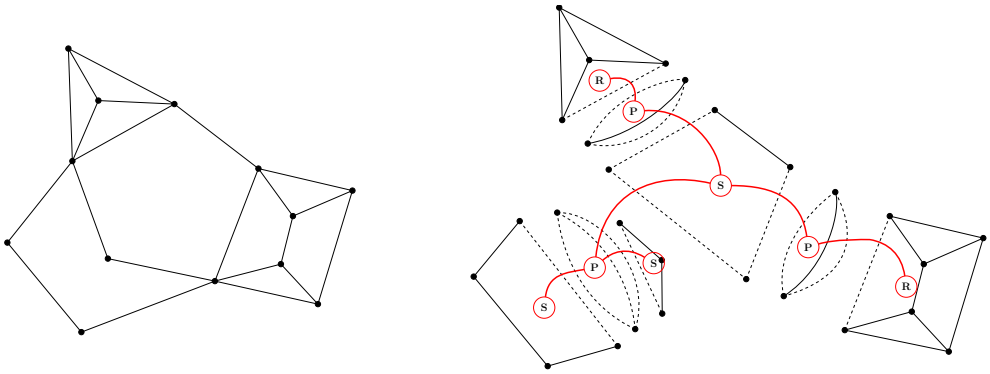

Figure 2 A biconnected graph and its SPQR tree. See Definition 3.

▶ **Definition 1.** Given a graph $G = (V, E)$, a *separation* of G is a pair of vertex sets (V', V'') such that the induced subgraphs $G' = G[V'], G'' = G[V'']$ cover G, and $V' \setminus V''$ and $V'' \setminus V'$ are both nonempty. A separation is *balanced* if $\max\{|V'|, |V''|\} \leq \alpha |V|$ for some fixed constant $\frac{1}{2} \leq \alpha < 1$. If (V', V'') is a separation of G, the set $S = V' \cap V''$ is called a *separator* of G. A separator S is *small* if $|S| = O(\sqrt{|V|})$, and it is a *cycle separator* if the subgraph of G induced by S is Hamiltonian.

3 Overview of Our Approach

Our data structure for decremental triconnectivity in planar graphs consists of two main ingredients. Before describing them, we need a few definitions. We recall that a graph G that is biconnected but not triconnected has at least one separation pair, i.e., a pair of vertices that can be removed to disconnect G:

▶ **Definition 2** (Hopcroft and Tarjan [29, p. 6])**.** Let $\{a, b\}$ be a pair of vertices in a biconnected multigraph G. Suppose the edges of G are divided into equivalence classes $E_1, E_2, \ldots, E_k$, such that two edges which lie on a common path not containing any vertex of $\{a, b\}$ except as an end-point are in the same class. The classes E_i are called the *separation classes* of G with respect to $\{a, b\}$. If there are at least two separation classes, then $\{a, b\}$ is a *separation pair* of G unless (i) there are exactly two separation classes, and one class consists of a single edge, or (ii) there are exactly three classes, each consisting of a single edge (these two exceptions actually make it easier to state some properties related to separation pairs).

Note that *separation pair*, which is a pair of vertices, should not be confused with *separation* (see Definition 1), which is a pair of vertex sets.

Our first ingredient for decremental triconnectivity is an algorithm for efficiently detecting separation pairs in planar graphs. The second ingredient is the maintenance of the SPQR-tree [12] for each biconnected component of a graph G under edge deletions and contractions. The SPQR-tree captures the structure of all separating pairs, and can be defined as follows:

▶ **Definition 3.** The SPQR-tree for a biconnected multigraph $G = (V, E)$ with at least 3 edges is a tree with nodes labeled S, P, or R, where each node x has an associated *skeleton graph* $\Gamma(x)$ with the following properties:

- For every node x in the SPQR tree, $V(\Gamma(x)) \subseteq V$.
- For every edge (x, y) in the SPQR tree, $V(\Gamma(x)) \cap V(\Gamma(y))$ is a separation pair $\{a, b\}$ in G, and there is a virtual edge ab in each of $\Gamma(x)$ and $\Gamma(y)$.

- For every node x in the SPQR tree, every edge in $\Gamma(x)$ is either in E or a *virtual edge* corresponding to an edge (x, y) in the SPQR-tree.
- For every edge $e \in E$ there is a unique node x in the SPQR-tree such that $e \in E(\Gamma(x))$.
- If x is an S-node, $\Gamma(x)$ is a simple cycle with at least 3 edges.
- If x is a P-node, $\Gamma(x)$ consists of a pair of vertices with at least 3 parallel edges.
- If x is an R-node, $\Gamma(x)$ is a simple triconnected graph.
- No two S-nodes are neighbors, and no two P-nodes are neighbors.

It turns out (see e.g. [12]) that the SPQR-tree for a biconnected graph is unique. The (skeleton graphs associated with) nodes of the SPQR-tree are sometimes referred to as the triconnected *components* of G.

Detecting separating 4-cycles. A 4-cycle is a simple cycle of length 4. We say that a 4-cycle in a planar embedded graph G is a *face 4-cycle* if it is a cycle bounding a face of G, and a *separating 4-cycle* otherwise. As we show in Appendix C, there is a one-to-one correspondence between separation pairs in G and separating 4-cycles in the vertex-face graph $G^{\diamond}$.

Since no two parallel edges can lie on the same 4-cycle, and no self-loop can be contained in a 4-cycle, we can assume the input graph is simple. However, when we contract edges, new parallel edges and self-loops may arise. To handle this, we could detect and remove parallel edges, but it turns out that both the algorithm and the analysis become simpler if we keep (most of) the additional edges, as long as no two parallel edges are consecutive in the circular ordering around both their endpoints.

If G has a face bounded by two edges, we can *simplify* the graph by deleting one of them. For our purposes we do not really care which one is deleted, but we need a rule that is consistent. For presentational purposes, we assume that we always keep the edge e with larger $\mathrm{id}(e)$.

This motivates the following definition of a quasi-simple graph[8]:

▶ **Definition 4.** A plane embedded graph is *quasi-simple* if the dual of each non-simple component has minimum degree 3. Given a plane embedded graph G and a set of vertices X, we define the subgraph of G *quasi-induced* by X to be the unique quasi-simple subgraph of G with vertex set X and the maximum total sum of $\mathrm{id}(e)$ values. Let $d_X(v)$ denote the degree of v in the subgraph quasi-induced by $X \cup \{v\}$.

Roughly speaking, a quasi-simple graph is obtained from a plane embedded multigraph by merging parallel edges that lie next to each other in the circular orderings around both their endpoints.

We build a structure for 4-cycle detection by recursively using balanced separators, and by detecting, for each separator, the cycles that cross the separator. Detecting 4-cycles that cross a separator is not trivial, and our analysis introduces a complicated potential function which reflects how well connected the non-separator vertices are with the separator, that is, how many neighbors on the separator they have. At the same time, we make sure that all the work done can be paid with the decrease in the potential. Our analysis exploits the fact that for a planar graph with separator S, at most $O(|S|)$ vertices have more than 4 neighbors in S.

The recursive use of separators can be sketched as follows: Let S be a small balanced separator in $G = (V, E)$ that induces a separation (V_1, V_2), that is, $V_1 \cap V_2 = S$ and

[8] In [40] these graphs are called *semi-strict*.

$V_1 \cup V_2 = V$. Moreover, let $n = |V|$. We observe that each 4-cycle is fully contained in V_1 or V_2, or consists of two paths of length 2 that connect vertices of S. This motivates the following recursive approach. We compute a separator S of $O(\sqrt{n})$ vertices and then find all paths of length 2 that connect vertices of S. Since the size of S is $O(\sqrt{n})$, there are only $O(n)$ pairs of vertices of S, and for each pair of vertices, we can easily check if the two-edge paths connecting them form any separating 4-cycles. It then remains to find the 4-cycles that are fully contained in either V_1 or V_2, which can be done recursively. Because S is a balanced separator, the recursion has $O(\log n)$ levels.

This algorithm can be made dynamic under contractions and edge insertions that respect the embedding of G. Contractions are easy to handle, as they preserve planarity. Moreover, a separator S of a planar graph can be easily updated under contractions. Namely, whenever an edge uw is contracted, the resulting vertex belongs to the separator iff any of u and w did. Insertions that preserve planarity, however, are in general harder to accommodate. To handle this we introduce a new type of separators that we call *face-preserving* separators, which (like cycle-separators) always exist when the face-degree is bounded. These are still preserved by contractions, but also ensure that any edge across a face can be inserted.

All in all, there are $O(\log n)$ levels of size $O(n)$ each, where each level handles insertions and contractions in constant time, leading to a total of $O(n \log n)$ time. The details of this construction and the proof of the below theorem can be found in Appendix B.

▶ **Theorem 5.** *Let G be an n-vertex quasi-simple plane embedded graph with bounded face degree. There exists a data structure that maintains G under contractions and embedding-respecting insertions, and after each update operation reports edges that become members of some separating 4-cycle. It runs in $O(n \log n)$ total time.*

Maintaining SPQR trees. The main challenge in maintaining an SPQR-tree is handling the case when an edge within a triconnected component is deleted. First of all, the data structure should be able to detect whether or not the component is still triconnected.

For any triconnected component Γ of G, we maintain a 4-cycle detection structure for the corresponding vertex-face graph $\Gamma^\diamond$. A separating 4-cycle in $\Gamma^\diamond$ corresponds to a separation pair in Γ, which would witness that Γ is no longer triconnected. The deletion or contraction of the edge e in the triconnected component Γ of G corresponds to an (embedding-respecting) insertion and immediate contraction of an edge in $\Gamma^\diamond$. This way by detecting 4-cycles in $\Gamma^\diamond$, we can detect when the corresponding triconnected component falls apart.

However, this is not the only challenge. If Γ does indeed cease to be triconnected, the SPQR-tree of $(\Gamma - e)$ (or (Γ/e) when doing a contraction) is a path $\mathcal{P}$. This is where we need the 4-cycle structure to output the edges contained in separating 4-cycles. Those edges correspond to a set of corners N of G. We use those corners to guide a search, which helps identify the non-largest components of the SPQR-path $\mathcal{P}$. More specifically, if a vertex v now belongs to two distinct triconnected components, there are two corners in N that separate the edges incident to v into two groups of edges, each belonging to a distinct triconnected component. We can afford to build a 4-cycle detection structure for $\Gamma'^\diamond$ for any non-largest triconnected component Γ' on the path from scratch. To obtain the data structure representing the largest component, we delete or contract the corresponding edges from Γ while updating $\Gamma^\diamond$. Since an edge only becomes part of a structure built from scratch when its triconnected component size has been halved, this happens only $O(\log n)$ times per edge, so the total time used for rebuilding is $O(n \log^2 n)$. The second logarithmic factor comes from rebuilding the data structure for 4-cycle detection, that takes $O(n \log n)$ time to initialize and process any number of operations.

Finally, since no two S-nodes can be neighbors and no two P-nodes can be neighbors, some S- or P-nodes in $\mathcal{P}$ may have to be merged with their (at most 2) neighbors of the same type outside $\mathcal{P}$. To handle this step efficiently, we keep the SPQR-tree rooted in an arbitrary node. While merging the skeleton graphs of two S- or P-nodes can be done in constant time, what can be more costly is updating the parent pointers in the children of the merged nodes. Hence, we move the children of the node with fewer children to the other node. This way, each node changes parent at most $O(\log n)$ times before it is deleted or split. The total number of distinct SPQR-nodes that exist throughout the lifetime of the data structure is $O(n)$, so the total time used for maintaining the parent pointers is $O(n \log n)$.

Since SPQR-trees are only defined for biconnected graphs, another challenge is to maintain SPQR-trees for each biconnected component, even as the decremental update operations cause the biconnected components to fall apart. We recall here that the structure of the biconnected components of a connected graph can be described by a tree called the *block-cutpoint tree* [23, p. 36], or *BC-tree* for short. This tree has a vertex for each biconnected component (block) and for each articulation point of the graph, and an edge for each pair of a block and an articulation point that belongs to that block. If the tree is rooted arbitrarily at any block, each non-root block has a unique articulation point separating it from its parent.

To handle updates, we notice that the SPQR-tree points to the fragile places where the graph is about to cease to be biconnected: An edge deletion in an S-node will break up a block in the BC-tree into path, and an edge contraction in a P-node breaks a block in the BC-tree into a star. Upon such an update, we remove the aforementioned S- or P-node from the SPQR-tree, breaking it up into an SPQR-forest. Each tree corresponds to a new block in the BC-tree. They form a path (or a star), and the ordering along the path, as well as the articulation points, can be read directly from the SPQR-tree.

On the other hand, in order to even know which SPQR tree to modify during an update, we can search in the BC-tree for the right SPQR-structure in which to perform the operation.

Bi- and triconnectivity. Finally, we use SPQR-trees to facilitate triconnectivity queries. First of all, vertices need to be biconnected in order to be triconnected. To facilitate biconnectivity queries, it is enough that each vertex v knows the name of the block $B(v)$ closest to the root in the BC-tree that contains it, and each block b knows the name of the vertex separating it from the parent $p(b)$. Then, any two vertices u and w are biconnected if and only if one of the following occur: $B(u) = B(v)$, or $u = p(B(v))$, or $v = p(B(u))$.

The information we maintain for triconnectivity is similar, using the SPQR-tree. Namely: each non-root node x in the SPQR-tree stores the *virtual edge* (see Definition 3) that separates it from its parent. Each vertex v stores (a pointer to) the node $C(v)$ closest to the root that contains it, and, in a special case, at most two other nodes that are the children of $C(v)$. Queries are handled similarly as above.

The main challenge is to handle updates. Note that the change to the SPQR-tree may involve both the split and merge of nodes. In particular, we have one split and up to several merges when a triconnected component falls apart into an SPQR-path. However, upon a merge, we can afford to update the information regarding vertices in the non-largest components, costing only an additive $\log n$ to the amortized running time. Similarly, upon a split, we update any information that relates to vertices in the non-largest components only.

The total running time is thus $O(n \log n + f(n))$, where $f(n)$ is the running time for maintaining the SPQR-tree. The following theorem is proven in Appendix A.

▶ **Theorem 6.** *There is a data structure that can be initialized on a planar graph G on n vertices in $O(n \log n)$ time, and supports any sequence of k edge deletions or contractions in*

total time $O((n+k)\log^2 n)$, while supporting queries to pairwise triconnectivity in worst-case constant time per query.

4 Decremental SPQR-trees

In this section, we use the data structure of Theorem 5 (described in Appendix B) to maintain an SPQR-tree (see Definition 3) for each biconnected component of G with at least 3 edges under arbitrary edge deletions and contractions. We start with some useful facts.

▶ **Lemma 7.** *For any pair of edges in a biconnected graph G, their corresponding faces of $G^\diamond$ are separated by a 4-cycle (v_1, f_1, v_2, f_2) if and only if they belong to different separation classes with respect to v_1, v_2 in G and with respect to f_1, f_2 in G^*.*

▶ **Lemma 8.** *Let G be a biconnected graph. If a 4-cycle $C = (v_1, f_1, v_2, f_2)$ in $G^\diamond$ is a separating cycle, then v_1, v_2 is a separation pair of G and f_1, f_2 is a separation pair of G^*.*

▶ **Lemma 9.** *Let G be a loopless biconnected plane graph and u, w be a separation pair in G. Consider the set of edges E_x incident to $x \in \{u, w\}$. Then, the edges of E_x belonging to each separation class of u, w are consecutive in the circular ordering around both u and w.*

▶ **Lemma 10.** *Let G be a triconnected plane graph and $e = uw \in E(G)$. Assume that $G - e$ is not triconnected. Then, the SPQR-tree of $G - e$ is a path H (we call it an SPQR-path). Moreover, given all edges that lie on 4-cycles in $(G - e)^\diamond$, we can compute all nodes of H except for the largest one in time that is linear in their size.*

For a planar graph, there is a nice duality, as proven by Angelini et al. [4, Lemma 1]. Define the dual SPQR-tree as the tree obtained from the SPQR-tree by interchanging S- and P-nodes, and taking the dual of the skeletons.

▶ **Lemma 11** (Angelini et al [4])**.** *The SPQR-tree of G^* is the dual SPQR-tree of G.*

Let G be a connected plane graph. Since $(G^\diamond)^*$ is 4-regular, $G^\diamond$ is quasi-simple and has bounded face-degree. Furthermore, any edge deletion or contraction in G that leaves G connected, corresponds to an edge insertion and immediate contraction in $G^\diamond$. Thus by Theorem 5 we can maintain a data structure for G under connectivity-preserving edge deletions and contractions, that after each update operation reports the corners that become part of a separating 4-cycle in $G^\diamond$.

Algorithm 1 Removing an edge e from a P-node x of T

```
1:  function removeP(e, x, T)
2:      remove e from Γ(x)
3:      if Γ(x) has two edges then
4:          if Γ(x) has no virtual edges then
5:              delete T
6:          else if Γ(x) has one virtual edge then
7:              y := the only neighbor of x
8:              e_x := the virtual edge in Γ(y) corresponding to x
9:              replace e_x by the non-virtual edge of Γ(x)
10:             remove x from T
11:         else if Γ(x) has two virtual edges then
12:             {y, z} := neighbors of x in T
13:             remove x from T, making y and z neighbors in T
14:             if y and z are S-nodes then
15:                 merge y and z into one node
```

In the algorithm we maintain one SPQR-tree for each biconnected component with at least 3 edges. We now describe how these trees are updated upon edge deletions. The procedures, depending on the type of the SPQR-tree node are given as Algorithms 1, 2 and 3. Note that the lines 4 and 5 in Algorithm 2 only introduce notation, that is the values of the variables are not computed. The proofs of correctness can be found in Appendix C.

Algorithm 2 Removing an edge e from an R-node x of T

```
1:  function removeR(e, x, T)
2:      remove e from Γ(x)
3:      if Γ(x) has a separation pair then
4:          X' := SPQR-path representing
                  Γ(x)
5:          x_big := the node of X' st. Γ(x_big)
                  has the most edges
6:          compute all nodes of X' \ x_big
7:          remove and contract edges of
                  Γ(x) to obtain Γ(x_big)
8:          replace x in T by X' (connect
                  each child of x to the cor-
                  rect node of X')
9:          for each S- or P-node z ∈ X' do
10:             for each neighbor z' ∉ X' do
11:                 if z, z' are same type then
12:                     merge z with z'
```

Algorithm 3 Removing an edge e from an S-node x of T

```
1:  function removeS(e, x, T)
2:      remove e from Γ(x)
3:      remove x from T
4:      for each edge e' in Γ(x) do
5:          Make a new BC-node z
6:          if e' is a virtual edge then
7:              y := neighbor of x in T corre-
                    sponding to e'
8:              Make the tree containing y the
                SPQR-tree for the new BC-node
9:              if y is a P-node then
10:                 removeP(y, e', T)
11:             else
12:                 removeR(y, e', T)
```

We can now prove the main theorem of this section. Note that, as in the block-cutpoint tree, we root each SPQR-tree in an arbitrary vertex.

▶ **Theorem 12.** *There is a data structure that can be initialized on a simple planar graph G on n vertices in $O(n \log n)$ time, and supports any sequence of edge deletions or contractions in total time $O(n \log^2 n)$, while maintaining an explicit representation of a rooted SPQR-tree*

for each biconnected component with at least 3 *edges, including all the skeleton graphs for the triconnected components. Moreover, during updates, the total number of times a node of an SPQR-tree changes its parent is* $O(n \log n)$.

Proof. We first partition the graph into biconnected components, and, as sketched in Section 3, maintain the block-cutpoint tree explicitly. Thus, given two vertices u, v, we can in $O(1)$ time access the biconnected component containing both of them, along with its auxiliary data. Now, for each biconnected component C_i, we compute the SPQR-tree T. This can be done in linear time due to [22]. We also root each SPQR-tree in an arbitrary node, and keep the trees rooted as they are updated.

For each node x of T we maintain the graph $\Gamma(x)$. Each virtual edge of $\Gamma(x)$ has a pointer to the neighbor of x it represents. Moreover, for each R-node r, we keep a data structure of Theorem 5 for detecting separating 4-cycles in the vertex-face graph $(\Gamma(r))^\diamond$. By Lemma 8, any separating 4-cycle in $(\Gamma(r))^\diamond$ corresponds to a separation pair in $\Gamma(r)$. Since r is an R-node, there are no separating 4-cycles to begin with, but some may appear after an update.

Since the total size of the R-components is n, it follows from Theorem 5 that the entire construction time is $O(n \log n)$.

Deletion. When an edge e is removed we find the node x of the SPQR-tree, such that e is a non-virtual edge in x. Then, we proceed according to Algorithms 1, 2 and 3.

Whenever an edge fg is deleted from an R-node r, we update the corresponding 4-cycle detection structure for $(\Gamma(r))^\diamond$. We first insert the dual edge $(fg)^*$ in the vertex-face graph, and then contract along that edge. This allows us to detect whether $\Gamma(r)$ has any separation pairs after each edge deletion.

Let us now analyze the running time. When processing an edge deletion, the following changes can take place in a SPQR-tree (all other changes can be handled in $O(1)$ time):

- an R-node is split into multiple nodes,
- two P-nodes or S-nodes are merged,
- an S- or P- node is deleted.

Note, a P- or S-node can never get split. So, though each edge may at first belong to nodes that are split, once it becomes a part of a P- or S-node, its node only participates in merges.

When two S- or P-nodes are merged, we can merge their skeleton graphs in constant time. These skeleton graphs have only two common nodes, and their lists of adjacent edges can be merged in constant time thanks to Lemma 9. When nodes are merged, we also have to update the parent pointers of their children. To bound the number of these updates, we merge the node with fewer children into the node with more. Thus, the number of parent updates caused by these merges is $O(n \log n)$, and so is the impact on the running time.

A similar analysis applies to the case when an R-node r is split into an SPQR-path. By Lemma 10, we can compute all but the largest node of the SPQR-path in linear time. Since the size of the skeleton graph in each of these nodes is at most half the size of $\Gamma(r)$, each edge takes part in this computation at most $O(\log n)$ times. For every new R-nodes, we also initialize their associated data structures for detecting 4-cycles. We charge the running time of each data structure to this initialization. From Theorem 5 we get that recomputing all the nodes and data structures takes $O(n \log^2 n)$ total time.

Taking care of the largest component of the SPQR-path is even easier, as we can simply reuse the skeleton graph of r and its associated data structure for detecting 4-cycles. To update the skeleton graph, we use the following lemma.

▶ **Lemma 13.** *If G is triconnected, $e \in E(G)$, and x is an R-node in the SPQR-tree for $G - e$, then there exists a sequence of $\big|E(G)\big| - \big|E(\Gamma(x))\big|$ edge deletions and contractions that transform $G - e$ into $\Gamma(x)$ while keeping the graph connected at all times.*

After an R-node r is split into a SPQR-path H we also need to update the parent pointers in the children of r. However, the number of children to update is at most the number of edges in the non-largest components of the SPQR-path. As we have argued, the total number of such edges across all deletions is $O(n \log n)$.

Contraction. The contraction of an edge of the embedded planar graph G corresponds to the deletion of an edge of its dual graph, G^*. By Lemma 11, the SPQR-tree of G^* is the dual SPQR-tree of G. Thus, if the edge was in a P-node of the SPQR-tree, its contraction is handled like the deletion of an edge in a S-node, and vice versa.

If the contracted edge e belongs to an R-node, that R node may expand to a path in the SPQR-tree (because deletion in G^* may expand an R-node into a path). In the vertex-face graph, we may find all edges participating in new separating 4-cycles, corresponding to separating corners of the graph. To find the new components, we simply apply Lemma 10 to the dual graph and proceed analogously to a deletion. ◀

A Decremental Triconnectivity

To answer triconnectivity queries, we maintain a rooted SPQR-decomposition (see e.g. [12, 22]) of each biconnected component of the planar graph.

Now it follows from the definition that pair of vertices in a biconnected graph is triconnected if and only if there exists a P or R component in the SPQR-tree containing them both. By associating a constant amount of information with every vertex in G and every node in the SPQR-tree, we can answer triconnectivity queries in constant time:

▶ **Definition 14.** A *triconnectivity query structure* for a biconnected graph consists of a rooted SPQR-tree, and the following additional information:

- For each node x in the SPQR-tree except the root, a pointer $e(x)$ to the virtual edge that separates it from its parent.
- For each vertex v, a pointer $C(v)$ to the node containing v that is closest to the root.
- For each vertex v such that $C(v)$ points to an S-node x, a set $D(v)$ of pointers to the at most 2 children of x that contain v.

▶ **Lemma 15.** *Given the triconnectivity query structure described in Definition 14, we can answer triconnectivity for any pair of vertices in constant time.*

Proof. Given vertices u and v. If $C(u) = C(v)$ and $C(u)$ is not an S-node, then u and v are triconnected. If $C(u) = C(v)$ is an S-node, u and v are triconnected if and only if they share a virtual edge in $\Gamma(C(u))$, which happens if and only if $D(u) \cap D(v)$ is non-empty. If $C(u) \neq C(v)$, then u and v are triconnected if and only if either u is an endpoint of $e(C(v))$, or, v is an endpoint of $e(C(u))$. ◀

Given Theorem 5, we have the tools ready for maintaining triconnectivity:

▶ **Theorem 6.** *There is a data structure that can be initialized on a planar graph G on n vertices in $O(n \log n)$ time, and supports any sequence of k edge deletions or contractions in total time $O((n+k)\log^2 n)$, while supporting queries to pairwise triconnectivity in worst-case constant time per query.*

Proof. For each vertex v and for each SPQR-node x, we associate the information $e(x), C(v), D(v)$ described in Definition 14.

Query. To answer a triconnected query (u, v), we first ask if (u, v) are biconnected. Otherwise, they cannot be triconnected. If they are, we get the SPQR-tree associated with their common biconnected component and use Lemma 15. This answers the query in $O(1)$ worst case time.

Updates. Our data structure for SPQR trees already maintain $e(x)$, so the main difficulty is in maintaining $C(v)$ and $D(v)$ for each vertex. Let x be the value of $C(v)$ before the change, let x' the new value, and suppose $x \neq x'$.

If x and x' are both R-nodes, $|E(\Gamma(x'))| < \frac{1}{2}|E(\Gamma(x))|$ so we are already using $\Omega(|E(\Gamma(x'))|) = \Omega(|V(\Gamma(x'))|)$ time to rebuild $\Gamma(x')^\diamond$. We can thus afford to update $C(v)$ for all $v \in V(\Gamma(x'))$.

If x is an R-node and x' is not, then $C(c)$ was split into $k > 1$ new nodes. In this case we are already using $\Omega(k)$ time maintaining the SPQR tree, so we can spend an additional $O(k)$ time on updating $C(v)$ for the $O(k)$ vertices from $V(\Gamma(x))$ whose new $C(v)$ is not an R-node.

If x is a P-node, then it has to be the root (since $C(v) = x$), so this can happen for at most 2 vertices per update and we can easily afford that.

If x is an S-node, then either the biconnected component was split into $k > 1$ new components and we can afford to spend $O(k)$ time on updating $C(v)$ for the $k-2$ vertices in S that were pointing to x. Or x was merged into another S-node. The total cost is linear in the total number of times some node changes parent due to such a merge, which is $O(n \log n)$.

Finally for each node x' that has a new parent p in the SPQR-tree, if p is an S-node, $e(x')$ has the two vertices whose $D(v)$ need to be changed, and this can be done in constant time. The total number of times this happens is $O(n \log n)$. ◀

B Detecting 4-Cycles Under Edge Contractions and Insertions

In this section we give an algorithm for detecting 4-cycles (simple cycles of length 4) in a planar embedded graph that undergoes contractions and edge insertions that respect the embedding. We say that a 4-cycle in a planar embedded graph G is a *face 4-cycle* if it is a cycle bounding a face of G, and a *separating 4-cycle* otherwise. For our purposes, only the separating 4-cycles are interesting, but we note in passing that new face 4-cycles are easy to detect under edge insertions and contractions:

▶ **Observation 16.** *An embedding-respecting edge insertion creates two new faces, and we may check in constant time whether each of them has degree 4 or not. An edge contraction affects degrees of only two faces (the two incident to the contracted edge), and we may check in constant time whether their new degree is 4 or not.*

The main goal of this section is to prove the following theorem.

▶ **Theorem 5.** *Let G be an n-vertex quasi-simple plane embedded graph with bounded face degree. There exists a data structure that maintains G under contractions and embedding-respecting insertions, and after each update operation reports edges that become members of some separating 4-cycle. It runs in $O(n \log n)$ total time.*

In order to detect 4-cycles, we use planar separators. In fact, in order to maintain our data structure dynamically, we need something a little bit stronger.

▶ **Definition 17.** Given a planar graph G, a separation (A, B) of G is said to be *face-preserving* if for any face f of G, all vertices of f belong to A or all vertices of f belong to B.

For instance, given a cycle separator K, we can form a face-preserving separation (A, B) such that $A \cap B = K$. Namely, K corresponds to a Jordan curve dividing the plane into two parts, S_A, S_B, where every face lies entirely in one part. Define A by all vertices incident to faces on S_A, and B similarly. Then, $A \cup B = G$, and $A \cap B = K$.

In our algorithm we need to maintain separations under edge insertions and contractions. Let (A, B) be a separation in G. When an edge is inserted, we do not modify the separation. When an edge uw is contracted into a vertex x, we obtain a new separation (A', B') as follows. If $u \in A$ or $w \in A$, we set $A' = (A \setminus \{u, w\}) \cup \{x\}$. Otherwise, $A' = A$. The set B' is defined analogously. Thanks to this convention, we obtain the following.

▶ **Lemma 18.** *Let (A, B) be a face-preserving separation in G. Let G' be the result of an embedding-respecting edge insertion or edge contraction, and let A', B' be the vertices corresponding to A and B in G'. Then (A', B') is a face-preserving separation in G', and $|A \cap B| - 1 \leq |A' \cap B'| \leq |A \cap B|$.*

Proof. If an edge is inserted that respects the embedding, it is inserted into some face f. By definition at least one of A, B contain all vertices on f, and in particular it also contains all the vertices of the two new faces that appear in G'. Since $A = A'$ and $B = B'$ in this case, the result follows.

If an edge uv is contracted, the resulting graph G' has the same faces as G, and the separation (A', B') is clearly face-preserving. If $u, v \in A \cap B$, then $|A' \cap B'| = |A \cap B| - 1$. Otherwise uv has an endpoint outside $A \cap B$. Without loss of generality, we can assume that $u \in A \setminus B$. In that case, $v \in A$ and $B = B'$ and it follows that $|A' \cap B'| = |A \cap B|$. ◀

▶ **Definition 19.** Given a graph G, a *separator tree* is a binary tree where each node x is associated with an induced subgraph H_x of G, such that for some constant $n_0 > 0$:

- If x is the root, $H_x = G$.
- If $|V(H_x)| > n_0$ then x has children y, z such that $(V(H_y), V(H_z))$ is a balanced separation of H_x with a small separator $S_x = V(H_y) \cap V(H_z)$.
- If $|V(H_x)| \leq n_0$ then x is a leaf.

A separator tree is a *cycle separator tree* if S_x is a cycle separator, and it is *face preserving* if $(V(H_y), V(H_z))$ is face-preserving, for all nodes x with children y and z.

▶ **Lemma 20.** *Given a planar graph with bounded face degree, we can in $O(n \log n)$ time build a face-preserving separator tree where each node x explicitly stores S_x and H_x. This tree has height $O(\log n)$, and uses $O(n \log n)$ space.*

We construct the tree in three steps. First, we take our graph G and make a triangulation $G^\triangle$. Then, referring to a result by Klein, Mozes, and Sommer [41], we make a cycle separator tree for $G^\triangle$. Finally, we can transform the cycle separator tree for $G^\triangle$ to a face preserving separator tree for G:

Proof. Let G be a graph with maximum face-degree k and let $G^\triangle$ be a triangulation of G. Then using the algorithm from [41, Theorem 3], we can in linear time compute a cycle separator tree for $G^\triangle$. Since the cycle separator tree is balanced, it has height $h \in O(\log n)$. Since the children of each node partition the faces of $G^\triangle$ contained in the node, and contain at most a constant number of other faces (called "holes"), the total number of faces (and hence vertices) in graphs associated with depth i nodes is $O(n)$ for each $0 \leq i < h$. Thus the total size of all these graphs in the cycle separator tree is $O(n \cdot h) = O(n \log n)$.

While this cycle separator tree indeed is face preserving for $G^\triangle$, it may be not face preserving for G because we may have edges $e \in G^\triangle \setminus G$ in the cycle separator. Luckily, for each such edge, we can fix the problem by adding the at most k vertices of the crossed face to the separator as follows:

For each $H_x^\triangle$ in the cycle separator tree for $G^\triangle$, we can construct H_x by starting with $V(H_x^\triangle)$, adding the remaining vertices of each face of G that is crossed by an edge in $E(H_x^\triangle) \setminus E(G)$, and taking the induced subgraph of G. By construction, $\left|H_x^\triangle\right| \leq |H_x| \leq O(k \left|H_x^\triangle\right|)$. Furthermore, if x is parent to y, z in the separator tree then $V(H_x) = V(H_y) \cup V(H_z)$. Clearly, $(V(H_y), V(H_z))$ is a balanced separation of H_x, and, by construction, it is face-preserving. What remains to be shown is that the size of the corresponding separator is small.

Given a node x in the separator tree with children y, z we compute $S_x = V(H_y) \cap V(H_z)$. Note that each vertex in $S_x \setminus S_x^\triangle$ must belong to a face of G that is crossed by an edge in $S_x^\triangle$, so also $\left|S_x^\triangle\right| \leq |S_x| \leq O(k \left|S_x^\triangle\right|)$. Thus the H_x and S_x form a face-preserving separator tree for G.

The total time to explicitly construct the face-preserving separator tree and all the associated graphs is $O(kn \log n)$. ◀

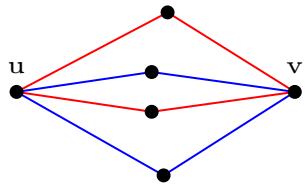

Figure 3 4 paths all participate in separating 4-cycles.

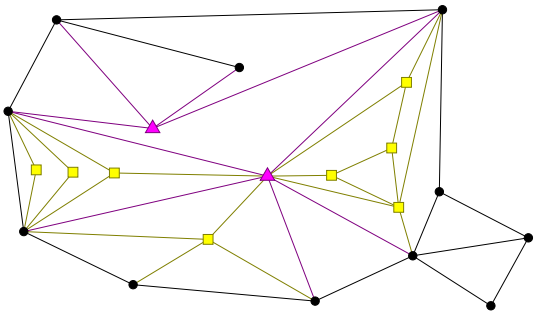

Figure 4 The sets M (magenta), Y (yellow), and K (black) from Lemma 23.

▶ **Lemma 21.** *Let G be a graph and (A, B) be a separation of G. Then, any 4-cycle either has exactly one vertex in $A \setminus B$, one vertex in $B \setminus A$, and the remaining two vertices in $A \cap B$, or the 4-cycle is completely contained in at least one of A or B.*

Proof. From Definition 1, for each edge e of G, both endpoints of e are in A or B. Thus, an edge that has one endpoint in $A \setminus B$ has its other endpoint in A. Using these facts, the lemma follows by simple case analysis. ◀

It follows that we can use the separator tree to detect 4-cycles as follows. For each leaf x of the separator tree, the graph H_x has constant size, so 4-cycles inside H_x can be detected in constant total time. In any other node, we have a graph with a separator K, and we need to dynamically detect 4-cycles that cross K under edge contractions and embedding-respecting edge insertions. Referring to Lemma 21 above, we only need to detect the two halves of a 4-cycle, that is, length-2 paths between the vertices of K.

▶ **Lemma 22.** *Let G be a plane embedded graph on n vertices and K be a set of vertices of G of size $|K| = O(\sqrt{n})$. Assume that G undergoes edge contractions and insertions respecting the embedding. There exists a data structure that after each update operation can report the edges of G that become members of separating 4-cycles whose two opposing vertices lie on K. Its total running time is $O(n + k)$, where k is the total number of edge contractions and insertions.*

We now proceed with the description of the data structure of the above lemma. First note that if a pair of vertices is connected by at least 4 paths of length two, all edges on those paths lie on separating 4-cycles (see Figure 3). Thus, if we can keep, for every pair of vertices of K, the list of all length 2 paths between them, we need to check at most 2 existing paths when a new path arrives, and then report at most 8 edges (4 new length-2 paths) that now belong to separating 4-cycles.

To report every edge only once, we also keep a Boolean flag for each edge that indicates whether it has been reported before, and check that flag before reporting.

We thus only need to argue that we can detect all the length-2 paths between K-vertices that arise in the graph under contractions and edge insertions, in $O(n+k)$ total time. We do that by constructing a potential function Φ that is initially $O(n)$, remains nonnegative, and drops at each operation proportionally to the amount of work done. We start by partitioning the vertices into 3 sets, that we need to treat differently.

▶ **Lemma 23.** *Given a planar graph $G = (V, E)$ and a vertex set $K \subseteq V$, let M denote the vertices $m \in V \setminus K$ that have $d_K(m) \geq 4$ (see Definition 4), and let Y denote $V \setminus (M \cup K)$. Then Y, M and K form a partition of V, and $|M| \leq |K| - 2$.*

Proof. The partition property follows trivially from the definition. Consider the maximal quasi-simple bipartite subgraph H of G with bipartition (M, K). By definition, each $m \in M$ has at least 4 neighbors in the subgraph of G quasi-induced by $K \cup \{m\}$. Thus for each $m \in M$, $d_H(m) \geq 4$, and so $4|M| \leq E(H)$. Since H is bipartite and quasi-simple, by Euler's formula we have $E(H) \leq 2(|M|+|K|)-4$. Combining the two we get $4|M| \leq E(H) \leq 2(|M|+|K|)-4$, which implies $|M| \leq |K| - 2$. ◀

The aim of the data structure is to notice when new common neighbors of pairs of vertices in K appear. The idea is that since K has size $O(\sqrt{|V|})$, there are only $O(|V|)$ pairs of vertices of K. For each such pair, we maintain a doubly-linked list of all length-2 paths between them, and for each edge we maintain a doubly-linked list of all such paths it participates in.

When an edge uw is inserted, new length-2 paths can only appear if $u \in K$ and/or $w \in K$. In this case the number of candidate paths to check is bounded by $d_K(u) + d_K(w)$, and this can be done in constant time per path.

When an edge uw is contracted, new length-2 paths between vertices of K may appear in other ways. For example, we have new paths between neighbors of u contained in K and neighbors of w in K. Other cases are possible if u or w belongs to K.

We now define a potential function that decreases by at least the number of candidate paths after each contraction. We can decide if each candidate path is an actual length-2 path between vertices in K in constant time.

It is defined in stages:

$$\Phi_q(X) = \sum_{v \in X} d_V(v)$$

$$\Phi_v(X) = 4|X| - \frac{1}{2}\sum_{v \in X} d_X(v) = 4|V(G_X)| - |E(G_X)|$$

$$\Phi_s(X) = 63(\Phi_v(X))^2 - \sum_{v \in X}(d_X(v))^2$$

$$\Phi = 6\Phi_v(V) + 3\Phi_q(Y \cup M) + \Phi_s(M \cup K)$$

▶ **Lemma 24.** *The potential Φ is initially $O(n)$ and remains nonnegative. The embedding-respecting insertion or contraction of an edge uv decreases Φ by at least the number of candidate paths.*

Before we proceed with the proof, we give a simple observation concerning quasi-simple graphs, which follows from Euler's formula and the fact that quasi-simple graphs of at least 3 vertices have faces of degree at least 3.

▶ **Observation 25.** *In a quasi-simple planar graph with $n \geq 3$ vertices, the number of edges is at most $3n - 6$.*

Proof of Lemma 24. To see the first statement, note that

$$1 \leq |K| \leq |M \cup K| \leq \Phi_v(M \cup K) \leq 4|M \cup K| \leq 4(2|K| - 2) = 8(|K| - 1)$$

where the inequality $|M \cup K| \leq \Phi_v(M \cup K)$, stems from $\Phi_v(M \cup K) - |M \cup K| = 3|V(G_{M \cup K})| - |E(G_{M \cup K})| \geq 0$, which is true by Observation 25. The last inequality is because there cannot be more than $|K| - 2$ vertices in M by Lemma 23. By a similar argument, we see that $\Phi_v(V) \geq 0$, and, being a sum of nonnegative terms, so is $\Phi_q(Y \cup M) \geq 0$.

We may thus realize that Φ is always positive:

$$\begin{aligned}\Phi \geq \Phi_s(M \cup K) &\geq 63(\Phi_v(M \cup K))^2 - \sum_{v \in M \cup K} (d_{M \cup K}(v))^2 \\ &\geq 63\,|M \cup K|^2 - \left(\sum_{v \in M \cup K} d_{M \cup K}(v)\right)^2 \\ &\geq 63\,|M \cup K|^2 - (6\,|M \cup K| - 12)^2 \\ &= 27\,|M \cup K|^2 + 144(|M \cup K| - 1) \\ &\geq 0\end{aligned}$$

Furthermore, note that Φ is initiated at $O(n)$. $\Phi_q(Y \cup M) \leq E$ which is $O(n)$ as the graph is planar. $\Phi_v(M \cup K) \leq 8|K| = O(\sqrt{n})$, and thus $(\Phi_v(M \cup K))^2 = O(n)$.

Before continuing with the second statement in the theorem, we consider how the different terms of Φ behave during changes.

Define $\Delta\Phi$, $\Delta\Phi_q$, $\Delta\Phi_v$ and $\Delta\Phi_s$ as the increases of the respective values Φ, Φ_q, Φ_v, Φ_s resulting from a change.

First we observe, that $\Phi_q(Y \cup M)$ has the following properties when contractions occur:

1. $\Delta\Phi_q(Y \cup M) \leq -2$ when a pair of vertices in $Y \cup M$ are contracted.
2. $\Delta\Phi_q(Y \cup M) \leq -d_V(v)$ when a vertex $v \in Y \cup M$ is contracted with a vertex in K.
3. $\Delta\Phi_q(Y \cup M) \leq 0$ when a pair of vertices in K are contracted.

Furthermore, we observe that for any $X \subseteq V$, $\Phi_v(X)$ has the following properties when changes occur:

1. $\Delta\Phi_v(X) \leq 0$ when a vertex of degree ≥ 4 is added to X.
2. $-4 \leq \Delta\Phi_v(X) \leq -1$ when a vertex of degree ≤ 3 is deleted from X.
3. $\Delta\Phi_v(X) = -1$ when an edge is added to G_X.
4. $-3 \leq \Delta\Phi_v(X) \leq -1$ when contracting any edge and reducing to a quasi-simple graph. ($\Delta\Phi_v(X) = -(3 - \eta)$, where $0 \leq \eta \leq 2$ is the number of additional edges deleted).

All the statements have similar proofs, so take for instance statement 4. Here, we decrease the first term by 4 but increase the second by $\eta + 1$, and thus, the resulting change is between -3 and -1.

When an edge uv is inserted, it can only create new length-2 paths between vertices of K if at least one of its ends is in K. Suppose without loss of generality that $u \in K$. Then we have the following cases for where v is before the insertion:

$v \in Y$: Then v had at most 3 neighbors in K before uv was added, and thus at most 3 candidate paths need to be checked. In this case $\Phi_v(V)$ drops by one, $\Phi_q(Y \cup M)$ increases by one, and $\Phi_s(M \cup K)$ is unchanged. Thus Φ drops by 3.

$v \in M$: In this case there are $d_K(v)$ new candidate paths, $\Phi_v(V)$ drops by one and $\Phi_q(Y \cup M)$ increases by one just like before. However, now $\Phi_v(M \cup K)$ drops by one, so $\Phi_v(M \cup K)^2$ drops by $2\Phi_v(M \cup K) - 1$ and so $\Phi_s(M \cup K)$ drops by $63(2\Phi_v(M \cup K) - 1) - (2d_{M \cup K}(u) - 1) - (2d_{M \cup K}(v) - 1) \geq 63(2\,|M \cup K| - 1) - (2(6\,|M \cup K| - 12) - 1) - (2(6\,|M \cup K| - 12) - 1) \geq 102\,|M \cup K| + 225$, which is much larger than $d_K(v)$.

$v \in K$ In this case there are $d_K(u) + d_K(v)$ new candidate paths, $\Phi_v(V)$ drops by one and $\Phi_q(Y \cup M)$ is unchanged. However, as in the previous case $\Phi_v(M \cup K)$ drops by one so $\Phi_s(M \cup K)$ drops by at least $102\,|M \cup K| + 225$, which is much larger than $d_K(u) + d_K(v)$.

Finally we consider the case where an edge uv is contracted. To check the lemma, one simply has to check the different combinations of which partition the two elements and the result belong to:

(Y, Y) merge to Y: $\Phi_q(Y \cup M)$ drops by at least 2, and the other terms in the potential are unchanged, so Φ drops by at least 6. Since the result v is in Y, there are at most 3 paths of length 2 in $G_{K\cup\{v\}}$ with v as a middle vertex, and at most 2 of them are new.

(Y, Y) merge to M: The product of the degrees, and therefore the number of candidate paths is at most 9. $\Delta\Phi_v(M \cup K) \leq 0$ and $\Delta\Phi_q(Y \cup M) \leq -2$, and the term $\sum_{v\in M\cup K}(d_{M\cup K}(v))^2$ drops by the sum of degrees squared. Thus, $\Delta\Phi \leq -2+0-8 < -9$, and we are done.

(M, Y)-merge Suppose $u \in M$ and $v \in Y$. Let w be the node that u, v are contracted to, it will be an M-node.

We call an edge (v, k) important if it participates in a new length 2 path between different vertices of K. Each important edge incident to $v \in Y$ will become part of the graph quasi-induced by $M \cup K$, and therefore cause a drop in $\Phi_v(M \cup K)$. This drop is enough to pay for all new paths containing that edge.

(K, Y)-merge Suppose $u \in K$ and $v \in Y$. Let w be the node that u, v are contracted to, it will be an K-node.

There are two types of new length-two (K, K) paths that arise: Paths having w as the middle vertex, and paths having w as an end vertex.

The paths with w as a middle vertex are accounted for just like the previous case.

Each new path having w as an end vertex must have a neighbor of v as middle vertex. There are (less than) $d_V(v)$ of these neighbors. Since $\Delta\Phi_q(Y \cup M) \leq -d_V(v)$ we can afford to look at each of them, and pay for at most 2 new paths for each.

Now consider a neighbor m of v that is middle vertex of some new path. If $m \in Y$ (after the contraction), then there is at most 2 new paths involving m, and the drop in $\Phi_q(Y \cup M)$ pays for them. If $m \in M \cup K$, then $d_{M\cup K}(m)$ has increased, and $\Phi_s(M \cup K)$ drops appropriately.

(M, M)-merge Suppose $u, v \in M$ are merged to the new vertex w. Note that $w \in M$. Let $X = \{x_1, \ldots, x_k\}$ be the set of common neighbors of u, v in $G_{M\cup K}$ that lose an edge when quasi-simplifying after the contraction, and note that $0 \leq \eta \leq 2$. Let $\Phi_v = \Phi_v(M \cup K)$, then

$$\Delta(\Phi_v^2) = (\Phi_v + \Delta\Phi_v)^2 - \Phi_v^2 = (\Phi_v - (3-\eta))^2 - \Phi_v^2 = (3-\eta)^2 - 2(3-\eta)\Phi_v$$

Let $a = d_{M\cup K}(u)$, $b = d_{M\cup K}(v)$, and for $i \in \{1, \ldots, \eta\}$ let $c_i = d_{M\cup K}(x_i)$. Then

$$a, b, c_i \leq a + b + \sum_{i=1}^{\eta} c_i \leq \sum_{y\in M\cup K} d_{M\cup K}(y) \leq 6\,|M \cup K| - 12 \leq 6\Phi_v - 12$$

And finally

$$\begin{aligned}
\Delta\Phi_s(M \cup K) &\leq 63\Delta(\Phi_v^2) \\
&\quad - \left(\left((a+b-(\eta+2))^2 + \sum_{i=1}^{\eta}(c_i-1)^2\right) - \left(a^2+b^2+\sum_{i=1}^{\eta}c_i^2\right)\right) \\
&= 63((3-\eta)^2 - 2(3-\eta)\Phi_v) \\
&\quad - \left(2ab + (\eta+2)^2 - 2(\eta+2)a - 2(\eta+2)b - \sum_{i=1}^{\eta}(2c_i-1)\right) \\
&= 63((3-\eta)^2 - 2(3-\eta)\Phi_v) \\
&\quad - 2ab - ((\eta+2)^2+\eta) + \left(2(\eta+2)a + 2(\eta+2)b + \sum_{i=1}^{\eta}2c_i\right) \\
&\leq 63((3-\eta)^2 - 2(3-\eta)\Phi_v) \\
&\quad - 2ab - ((\eta+2)^2+\eta) + \left(2(\eta+2) + 2(\eta+2) + \sum_{i=1}^{\eta}2\right)(6\Phi_v - 12) \\
&= 63((3-\eta)^2 - 2(3-\eta)\Phi_v) \\
&\quad - 2ab - ((\eta+2)^2+\eta) + (6\eta+8)(6\Phi_v - 12) \\
&= -2ab + \begin{cases} 467 - 330\Phi_v & \text{if } \eta = 0 \\ 74 - 168\Phi_v & \text{if } \eta = 1 \\ -195 - 6\Phi_v & \text{if } \eta = 2 \end{cases} \\
&\leq -2ab - 6\Phi_v \qquad \text{for } \Phi_v \geq 2 \qquad (1)
\end{aligned}$$

In particular, $\Delta\Phi \leq -ab = -d_{M\cup K}(u) \cdot d_{M\cup K}(v) \leq -d_S(u) \cdot d_S(v)$, as desired.

(K, M)-merge: Suppose $u \in K$ and $v \in M$. Let w be the node that u, v are contracted to, it will be an K-node.

There are two types of new length-two (K, K) paths that arise: Paths having w as the middle vertex, and paths having w as an end vertex.

The paths with w as a middle vertex are accounted for just like the previous case.

Each new path having w as an end vertex must have a neighbor of v as middle vertex. There are (less than) $d_V(v)$ of these neighbors. Since $\Delta\Phi_q(Y \cup M) \leq -d_V(v)$ we can afford to look at each of them, and pay for at most 2 new paths for each.

Now consider a neighbor m of v that is middle vertex of some new path. If $m \in Y$ (after the contraction), then there is at most 2 new paths involving m, and the drop in Φ_q pays for them.

Let M be the set of neighbors of v in $M \cup K$ that is middle of some new path. The number of such paths is at most $|E(G_{M\cup K})| \leq 3\,|M \cup K| - 6 \leq 3\Phi_v(M \cup K) - 6$, since each must contain a unique edge from $G_{M\cup K}$. And the $-6\Phi_v(M \cup K)$ pays for them.

(K, K)-merge: Suppose $u, v \in K$. Let w be the node that u, v are contracted to, it will be an K-node.

There are two types of new length-two (K, K) paths that arise: Paths having w as the middle vertex, and paths having w as an end vertex.

The paths with w as a middle vertex are accounted for just like the previous two cases.

Each new path having w as an end vertex was already an (K, K) path with either u or v before the merge. For each of u, v there is at most $|K| - 1$ such pairs that (may) need to

be updated, so the total cost of updating these is less than $2(|K| - 1) \leq 2\Phi_v$. The $-6\Phi_v$ can pay for these updates. ◀

As a result, Lemma 22 follows, and we are finally ready to prove Theorem 5.

Proof of Theorem 5. Given a planar graph G with bounded face-degree, we build a face-preserving separator tree as in Lemma 20 in $O(n \log n)$ time. For each internal vertex of the tree, we may detect new 4-cycles crossing the separator due to Lemma 22. The leaves have size at most $n_0 = O(1)$, and we can detect 4-cycles in the leaves in $O(1)$ time.

We can distinguish between face 4-cycles and separating 4-cycles. Namely, we can choose to report only when an edge first lies on any 4-cycle, or when it first lies on a separating one, as described in Lemma 22.

An edge insertion in the graph H_x needs to be duplicated in each child of x that contains both vertices. However, the drop in the Φ potentials for each node is large enough to pay for each cascading insertion. Whenever a contraction in the graph H_x for a node x of the separator tree introduces a new edge between two separator vertices, that edge may need to be added to the subtree containing the other side of the separation, but again that is paid for. In general, if we update graphs closer to the root first, the changes only propagate down and every change is paid for by a corresponding drop in the potential. ◀

C Omitted proofs from Section 4

▶ **Lemma 7.** *For any pair of edges in a biconnected graph G, their corresponding faces of $G^\diamond$ are separated by a 4-cycle (v_1, f_1, v_2, f_2) if and only if they belong to different separation classes with respect to v_1, v_2 in G and with respect to f_1, f_2 in G^*.*

Proof. Let C be the 4-cycle. Consider a path L in G containing edges e_1 and e_2. Consider the set of faces F in $G^\diamond$ that are incident to a vertex on L. L does not cross v_1, v_2 if and only if F is completely contained on one side of C, which happens if and only if e_1 and e_2 are not separated by C. An identical argument can be made about f_1, f_2 in G^* ◀

▶ **Lemma 8.** *Let G be a biconnected graph. If a 4-cycle $C = (v_1, f_1, v_2, f_2)$ in $G^\diamond$ is a separating cycle, then v_1, v_2 is a separation pair of G and f_1, f_2 is a separation pair of G^*.*

Proof. If C is a separating cycle, there is at least two faces on either side. By Lemma 7 there are thus at least two different separation classes with respect to v_1, v_2 (or f_1, f_2). If there exactly 2 separation classes, each consists of at least two edges. If there are exactly 3 classes, at least one of them consists of at least two edges. ◀

▶ **Lemma 9.** *Let G be a loopless biconnected plane graph and u, w be a separation pair in G. Consider the set of edges E_x incident to $x \in \{u, w\}$. Then, the edges of E_x belonging to each separation class of u, w are consecutive in the circular ordering around both u and w.*

Proof. The proof is by contradiction. Assume that the circular order of some 4 edges incident to u is e_1, e_2, e_3, e_4. Moreover, assume that only e_1 and e_3 belong to the same separation class. From Definition 2 there is a path that begins with e_1 and ends with e_3 that does not contain u or w as its internal point. Thus, this path is a cycle C that does not go through w. Hence, every path from either e_2 or e_4, that ends in w and does not contain u as an internal point, has to go through C. This contradicts the fact that e_2 and e_4 are in different separation classes than e_1 and e_3. Clearly, the same argument applies to w. ◀

▶ **Lemma 26.** *Let G be a loopless biconnected plane graph. Let F be a subset of edges of G, such that F is a separation class for some pair u, w of vertices of G. Then there exists a 4-cycle (possibly non-separating) in $G^\diamond$ that separates the set of faces that correspond to F from all other faces.*

Proof. Throughout the proof by separation class we mean one of separation classes defined by u and w. Clearly, each separation class has to have an edge incident to u or w (otherwise, since the graph is connected, it would not be maximal). In fact, since G is biconnected, each separation class has edges incident both to u and w. If a separation class had edges incident only to one of the two vertices, this vertex would be an articulation point.

Denote the edges incident to u in circular order by $e_1, \ldots, e_k$. For convenience, let $e_{k+1} := e_1$ and $e_0 := e_k$. Moreover, assume that $e_i \in F$ iff $a \leq i \leq b$, where $1 \leq a \leq b \leq k$. Note that by Lemma 9, a and b are well-defined (for some way of breaking the circular ordering into a sequence $e_1, \ldots, e_k$).

Let f_1 be the face that comes in the circular order between e_{a-1} and e_a and f_2 be the face that comes between e_b and e_{b+1}. Note that $f_1 \neq f_2$.

We now show that there is a 4-cycle in $G^\diamond$ that contains u, w, f_1 and f_2. To that end, we prove that both u and w lie on f_1. Indeed, the cycle bounding f_1 is simple and contains edges from two separation classes. Thus, it has to contain both u and w. Similarly, both u and w lie on f_2.

This implies that $G^\diamond$ contains a 4-cycle C_F going through u, w, f_1 and f_2, and by construction, C_F separates the faces corresponding to F from all other edges. ◀

▶ **Lemma 27.** *Let G be loopless biconnected graph. If v_1, v_2 is a separation pair in G, then there exists a separation pair f_1, f_2 in G^* such that (v_1, f_1, v_2, f_2) is a separating cycle in $G^\diamond$.*

Proof. If every separation class with respect to v_1, v_2 consists of a single edge, then G consists of two vertices connected by multiple edges and the lemma is trivial. It suffices to use the fact that since v_1, v_2 is a separation pair, there are at least 4 edges in G. Otherwise there is a separation class F with at least two edges, such there are at least two edges in $E(G) \setminus F$. Now apply Lemma 26, to get a delimiting cycle C in $G^\diamond$ that separates faces corresponding to F from all other faces. Denote the vertices of C by v_1, f_1, v_2, f_2. Since both F and $E(G) \setminus F$ are nontrivial (both contain more than one edge), C is a separating cycle in $G^\diamond$. It then follows from Lemma 8 that f_1, f_2 form a separation pair in G^*. ◀

▶ **Lemma 13.** *If G is triconnected, $e \in E(G)$, and x is an R-node in the SPQR-tree for $G - e$, then there exists a sequence of $\big|E(G)\big| - \big|E(\Gamma(x))\big|$ edge deletions and contractions that transform $G - e$ into $\Gamma(x)$ while keeping the graph connected at all times.*

Proof. For each neighbor y of x in the SPQR-tree T we proceed as follows. Let a, b be the separation pair corresponding to the edge in T between x and y. Consider all nodes reachable from y in T with a path that does not contain x. Let D be the set of non-virtual edges in all these nodes. While there is an edge e in D that is not a self-loop and not an edge between a and b, contract it. Then if there are any self-loops delete them. When all edges in D go between a and b, delete edges until there is only one left. ◀

▶ **Lemma 10.** *Let G be a triconnected plane graph and $e = uw \in E(G)$. Assume that $G - e$ is not triconnected. Then, the SPQR-tree of $G - e$ is a path H (we call it an SPQR-path). Moreover, given all edges that lie on 4-cycles in $(G - e)^\diamond$, we can compute all nodes of H except for the largest one in time that is linear in their size.*

Proof. Let us first prove that H is indeed a path. Since $G - e$ is biconnected, there exist two internally vertex-disjoint paths between u and w. No separation pair in $G - e$ can have both vertices on the same of these paths, since otherwise it would be a separation pair in G. Moreover, observe that each separation pair defines at most two separation classes that consist of more than one edge (otherwise, it is also a separation pair in G). Thus, we can split $G - e$ into two subgraphs by using an arbitrary separation pair in $G - e$. By repeating the same reasoning on both subgraphs, we get that H is a path. Observe that u and w belong to the nodes at the opposite ends of H.

Note that since we know the edges belonging to 4-cycles in $(G - e)^\diamond$, by using Lemma 7 we also know all separation pairs in $G - e$. We now describe how to compute all components of H (i.e. the skeleton graphs stored in the nodes) except the largest one. Consider an algorithm that starts from one end of H and discovers the components one by one, each in linear time. Observe that if each component of H has size at most $|E(G)|/2$, we can afford to detect all components of H, without affecting the total running time. However, to prepare for the opposite case, we need to do the search in parallel, starting from both ends of H. Let y be the largest component of H. Observe that only one search can start exploring edges of y. As soon as the other search reaches y, we have discovered all separation pairs (that is why we need to know all separation pairs upfront), and both searches can stop. Thus, one of the searches only uses time that is at most the total size of all non-largest components of H. Since the other search runs in parallel, it runs in the same asymptotic time.

To complete the proof it remains to describe how the search procedure works. Recall that by Lemma 9, for each separation pair u, w of $G - e$, the edges belonging to each separation class come in consecutive order around u and w. Observe that the edges of the 4-cycles of $(G - e)^\diamond$ correspond to the corners of $G - e$ that lie between edges belonging to distinct separation classes. Thus, once we mark these corners in $G - e$, we can run a DFS-search that, once started from an edge belonging to a skeleton graph of an S- or R-node, explores all edges of this graph (and only those).

Observe moreover, that from our earlier analysis it follows that the endpoints u and w of e do not belong to any separation pair. This implies that both u and w are contained in S- or R- nodes. Let us focus on the search starting from u. Note that it discovers the entire component containing u and the separation pairs that separates it from the rest of the SPQR-tree. If the skeleton graph of this component is a path connecting the vertices of the separation pair, we have found an S-node. Otherwise, we have found an R-node.

Now assume we have found some prefix of the SPQR-path that ends at a separation pair a, b. If there is an edge ab (this edge comes next in the circular ordering after the edges we have visited, so it is easy to find), the next node on the SPQR-path is a P-node. After we have processed all edges between a and b, we insert a virtual edge between a and b and continue the search starting from this edge in a similar way to the search that has discovered the first node on the SPQR-path. Clearly, the algorithm runs in linear time. ◀

We now show that algorithms 1, 2, 3 are correct. In each proof, the goal is to show that after the procedure the tree T satisfies Definition 3.

▶ **Lemma 28.** *Algorithm 1 is correct.*

Proof. If after the edge deletion, $\Gamma(x)$ sill has at least 3 edges, then clearly T is a valid SPQR-tree. Otherwise, $\Gamma(x)$ has exactly two edges and we consider three cases. Recall that the number of virtual edges in a node is equal to the number of the node's neighbors in the SPQR-tree. If $\Gamma(x)$ has no virtual edges, then x is the only node of T, and thus this biconnected component now only has 2 edges, so we should delete the entire SPQR-tree.

If $\Gamma(x)$ has exactly one virtual edge, x has exactly one neighbor. In this case, simply x represents one edge of the graph, so it has to be merged with its only neighbor and the virtual edge in the neighbor becomes non-virtual. If $\Gamma(x)$ has two virtual edges we remove x and the neighbors of x become neighbors. Note that the neighbors of x cannot be P-nodes. Thus, unless x has two neighboring S-nodes, we obtain a valid SPQR-tree. In the remaining case, it is easy to see that the two S-nodes can be merged into one S-node. ◀

▶ **Lemma 29.** *Algorithm 2 is correct.*

Proof. If after removing the edge, $\Gamma(x)$ is triconnected, clearly the tree is a valid SPQR-tree. Otherwise, by Lemma 10, $\Gamma(x)$ is represented by a SPQR-path. It is easy to see that after replacing x by the SPQR-path X', we obtain a valid SPQR-tree, unless there are two neighboring S-nodes or P-nodes. Since the SPQR-path is a SPQR-tree, each such pair contains exactly one node z from the SPQR-path. If it is a P node, it can not be the end of the path, and so has at least 2 virtual edges to its neighbors on the path, and at most one more virtual edge to a neighbor z' outside the path. If it is an S-node it may have up to 2 virtual edges to a neighbor z' outside the path. Clearly, if z and z' have the same type, they can be merged into one node, and this yields a valid SPQR-tree. ◀

▶ **Lemma 30.** *Algorithm 3 is correct.*

Proof. Observe that after removing an edge $e = uw$, each vertex of $\Gamma(x)$ distinct from u and w is an articulation point. Thus, each neighbor of x now belongs to a different biconnected component. Thus, we update T by deleting x, which breaks T into a piece for each neighbor y. For each piece we create a new BC-node z.

Each non-virtual edge of $\Gamma(x)$ becomes a biconnected component by itself, so we could simply ignore these edges from now on. For each virtual edge of $\Gamma(x)$, we delete the corresponding edge in the neighbor of x using an appropriate function. From Lemmas 28 and 29 it follows that the SPQR-trees are updated correctly. ◀

References

1. Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 477–486, 2016.
2. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.
3. Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1199–1218, 2012.
4. Patrizio Angelini, Thomas Bläsius, and Ignaz Rutter. Testing mutual duality of planar graphs. *Int. J. Comput. Geometry Appl.*, 24(4):325–346, 2014.
5. Dan Archdeacon and R Bruce Richter. The construction and classification of self-dual spherical polyhedra. *Journal of Combinatorial Theory, Series B*, 54(1):37 – 63, 1992.
6. Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. *SIAM J. Comput.*, 44(1):88–113, 2015.

7 Graham R. Brightwell and Edward R. Scheinerman. Representations of planar graphs. *SIAM Journal on Discrete Mathematics*, 6(2):214–229, 1993.

8 Gunnar Brinkmann, Sam Greenberg, Catherine Greenhill, Brendan D. Mckay, Robin Thomas, and Paul Wollan. Generation of simple quadrangulations of the sphere. *Discrete Math.*, 305(1-3):33–54, December 2005.

9 Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Jakub Łącki, and Nikos Parotsidis. Decremental single-source reachability and strongly connected components in $\widetilde{O}(m\sqrt{n})$ total update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 315–324, 2016.

10 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004.

11 Camil Demetrescu and Giuseppe F. Italiano. Mantaining dynamic matrices for fully dynamic transitive closure. *Algorithmica*, 51(4):387–427, 2008.

12 Giuseppe Di Battista and Roberto Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996.

13 Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996.

14 Krzysztof Diks and Piotr Sankowski. Dynamic plane transitive closure. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 594–604, 2007.

15 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.

16 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification I: Planarity testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996.

17 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator-based sparsification II: Edge and vertex connectivity. *SIAM J. Comput.*, 28(1):341–381, 1998. Announced at STOC '93.

18 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992.

19 Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72(5):868–889, 2006.

20 Dora Giammarresi and Giuseppe F. Italiano. Decremental 2- and 3-connectivity on planar graphs. *Algorithmica*, 16(3):263–287, 1996. Announced at SWAT 1992.

21 Jens Gustedt. Efficient union-find for planar graphs and other sparse graph classes. *Theor. Comput. Sci.*, 203(1):123–141, 1998.

22 Carsten Gutwenger and Petra Mutzel. *A Linear Time Implementation of SPQR-Trees*, pages 77–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

23 Frank Harary. *Graph Theory*. Addison-Wesley Series in Mathematics. Addison Wesley, 1969.

24 Monika Rauch Henzinger and Mikkel Thorup. Sampling to provide or to bound: With applications to fully dynamic graph algorithms. *Random Struct. Algorithms*, 11(4):369–379, 1997.

25 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

26 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, Eva Rotenberg, and Piotr Sankowski. Contracting a Planar Graph Efficiently. *ArXiv e-prints*, June 2017. `https://arxiv.org/abs/1706.10228v1` Accepted for ESA 2017.

27 Jacob Holm and Eva Rotenberg. Dynamic planar embeddings of dynamic graphs. *Theory of Computing Systems*, Apr 2017.

28 Jacob Holm, Eva Rotenberg, and Christian Wulff-Nilsen. Faster fully-dynamic minimum spanning forest. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, Sept. 14-16, 2015, Proceedings*, pages 742–753, 2015.

29 John E. Hopcroft and Robert Endre Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, 1973.

30 John E. Hopcroft and Robert Endre Tarjan. A V log V algorithm for isomorphism of triconnected planar graphs. *J. Comput. Syst. Sci.*, 7(3):323–331, 1973.

31 John E. Hopcroft and J. K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1974, Seattle, Washington, USA*, pages 172–184, 1974.

32 Giuseppe F. Italiano, Adam Karczmarz, Jakub Łącki, and Piotr Sankowski. Decremental single-source reachability in planar digraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1108–1121, 2017.

33 Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 313–322, 2011.

34 Goossen Kant. Algorithms for drawing planar graphs. 2001.

35 Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 338–355, 2012.

36 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142, 2013.

37 Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Faster worst case deterministic dynamic connectivity. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 53:1–53:15, 2016.

38 Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 81–91, 1999.

39 Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, BC, Canada, January 23-25, 2005*, pages 146–155, 2005.

40 Philip N. Klein and Shay Mozes. Optimization algorithms for planar graphs, 2017.

41 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514, 2013.

42 Jakub Łącki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Trans. Algorithms*, 9(3):27:1–27:15, 2013.

43 Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the Steiner tree. In

Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015, pages 11–20, 2015.

44 Jakub Łącki and Piotr Sankowski. Min-cuts and shortest cycles in planar graphs in $O(n \log \log n)$ time. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 155–166, 2011.

45 Jakub Łącki and Piotr Sankowski. Optimal decremental connectivity in planar graphs. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 608–621, 2015.

46 Karl Menger. Zur allgemeinen kurventheorie. *Fund. Math.*, 10:96–115, 1927.

47 Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, Las Vegas, and $O(n^{1/2 - \epsilon})$-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1122–1129, 2017.

48 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Proceedings of the 58th Annual Symposium on Foundations of Computer Science, FOCS 2017*, 2017. To appear.

49 Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008.

50 Pierre Rosenstiehl. Embedding in the plane with orientation constraints: The angle graph. *Annals of the New York Academy of Sciences*, 555(1):340–346, 1989.

51 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *45th Symposium on Foundations of Computer Science FOCS 2004, 17-19 October 2004, Rome, Italy, Proceedings*, pages 509–517, 2004.

52 Shay Solomon. Fully dynamic maximal matching in constant update time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334, 2016.

53 Sairam Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings*, pages 372–383, 1993.

54 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350, 2000.

55 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 112–119, 2005.

56 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1757–1769, 2013.

57 Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1130–1143, 2017.