



# Dynamic Embeddings of Dynamic Single-Source Upward Planar Graphs

Ivor van der Hoog  

Technical University of Denmark, Lyngby, Denmark

Irene Parada  

Department of Mathematics, Universitat Politècnica de Catalunya, Barcelona, Spain

Eva Rotenberg  

Technical University of Denmark, Lyngby, Denmark

---

## Abstract

---

A directed graph  $G$  is *upward planar* if it admits a planar embedding where each edge is  $y$ -monotone. Unlike planarity testing, upward planarity testing is NP-hard except in restricted cases, such as when the graph has the single-source property (i.e., each connected component has one source).

In this paper, we present a dynamic data structure for maintaining an upward combinatorial embedding  $\vec{\mathcal{E}}(G)$  of a single-source upward planar graph subject to edge deletions, edge contractions, directed edge insertions across a face, and single-source-preserving vertex splits through specified corners (i.e., the gaps between pairs of consecutive edges that share a vertex and a face). We furthermore support changes to the embedding  $\vec{\mathcal{E}}(G)$  in the form of subgraph flips that mirror or slide the placement of a subgraph that is connected to the rest of the graph via at most two vertices. Updates that are incompatible with the current upward planar embedding are identified and rejected.

All update operations are supported as long as the graph remains upward planar. In addition, we support queries that can tell whether two vertices can be connected with a directed edge while the graph remains single-source (we call these *uplinkability queries*). If a pair of vertices are not uplinkable, we facilitate one-flip-linkable queries: These point to a flip that makes them uplinkable, if any such flip exists. We dynamically maintain a linear-size data structure on  $G$  which supports incidence queries between a vertex and a face, and uplinkability queries for vertex pairs. We support all updates and queries in  $O(\log^2 n)$  worst-case time.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Dynamic graph algorithms; Mathematics of computing  $\rightarrow$  Graphs and surfaces

**Keywords and phrases** dynamic graphs, data structures, computational geometry, graph drawing, graph algorithms, upward planarity

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2024.70

**Related Version** *Full Version:* <https://arxiv.org/abs/2209.14094>

**Funding** This research was supported by Independent Research Fund Denmark grant 2020-2023 (9131-00044B) “Dynamic Network Analysis”.

*Ivor van der Hoog:* This project has additionally received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987.

*Irene Parada:* I. P. is a Serra Hünter Fellow. Partially supported by grant 2021UPC-MS-67392 funded by the Spanish Ministry of Universities and the European Union (NextGenerationEU) and by grant PID2019-104129GB-I00 funded by MICIU/AEI/10.13039/501100011033.

*Eva Rotenberg:* This research was additionally supported by Carlsberg Foundation Young Researcher Fellowship CF21-0302 “Graph Algorithms with Geometric Applications”.



© Ivor van der Hoog, Irene Parada, and Eva Rotenberg;  
licensed under Creative Commons License CC-BY 4.0

32nd Annual European Symposium on Algorithms (ESA 2024).

Editors: Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman; Article No. 70; pp. 70:1–70:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

A directed graph is *upward planar* if it admits a drawing that is both *upward* (every edge has monotonically increasing  $y$ -coordinates) and *planar* (without crossings). Upward planarity is a natural analogy of planarity for directed graphs. We can test if a graph admits a planar embedding in linear time since 1974 [18]. In sharp contrast, testing upward planarity is, in general (for multi-source graphs), NP-hard [13].

We consider a dynamic  $n$ -vertex single-source directed graph  $G$  with an upward embedding  $\vec{\mathcal{E}}(G)$ . An ordered vertex pair  $(u, v)$  is *uplinkable* in  $\vec{\mathcal{E}}(G)$  whenever we may insert the edge  $u \rightarrow v$  across some face without violating upward planarity. We design a data structure that supports updates to the graph (that preserve upward planarity and  $G$  being single-source), local changes to the embedding (flips and slides), and answers queries to whether an edge can be inserted in the current embedding (uplinkability), or inserted after performing a limited type of local changes to the embedding (one-flip uplinkability). We support these queries in  $O(\log^2 n)$  time. The bounds we obtain in this more challenging upward-planar setting match the ones of the planar case [15] and generalise the dynamic data structure for single-source directed graphs with a fixed embedding and outer face [29]. Contrary to [29], we also allow changes to the embedding and edge deletions that disconnect the graph.

For planar graphs, a corresponding dynamic data structure that supports similar queries and changes to the embedding [15] has been proven instrumental for (fully) dynamic planarity testing of a dynamic graph [17]. An additional motivation for this work is the hope that with additional insights and techniques, it would serve as a stepping stone towards a fully-dynamic algorithm for upward planar single source graphs (i.e., facilitating the deletion or insertion of any edge, as long as the graph remains upward planar and component-wise single-source).

**Related work to upward planarity.** Upward planar graphs are spanning subgraphs of planar  $st$ -graphs [2, 21, 27]. While testing upward planarity is NP-complete for general graphs [13], polynomial algorithms exist for several restricted classes. One of the most relevant such results is the linear-time algorithm for single-source directed graphs [6, 7, 19]. Other classes for which upward planarity can be tested in polynomial time are graphs with a fixed (upward) embedding [5], outerplanar graphs [24], and series-parallel graphs [9]. Dynamically, it can be checked in  $O(\log n)$  amortised time whether an embedded single-source upward planar directed graph with a fixed external face can stay as such after an edge insertion or deletion without changing the embedding [29]. There are no existing worst-case results for dynamic upward embeddings and no results for dynamic upward embeddings subject to flips in the embedding. The study of upward planar graphs continues to be a prolific area of research, including recent developments in parameterized algorithms for upward planarity [8], bounds on the page number [20], morphing [23], and extension questions [7, 22]. In particular, in biconnected graphs, it can be tested in  $O(n^2)$  time whether a given upward planar drawing can be extended to an upward planar drawing of an  $n$ -vertex single-source directed graph  $G$  [7]. For  $st$ -graphs, the running time was recently improved to  $O(n \log n)$  [22]. This contrasts the linear-time extension algorithm for the planar undirected case [1].

**Dynamic maintenance of planar (embedded) graphs.** Dynamic maintenance of graphs and their embedding is a well-studied topic in theoretical computer science [3, 4, 10, 11, 12, 14, 15, 16, 17, 28, 30]. In this area, we typically study some graph  $G = (V, E)$  subject to adding or removing edges to the edge set  $E$ . A combinatorial embedding  $\mathcal{E}(G)$  specifies the outer face and for each vertex  $v \in V$  a cyclic ordering of the edges incident to  $v$ .

One famous approach to dynamic maintenance of an embedding is the work by Eppstein [10] who studies maintaining an embedding  $\mathcal{E}(G)$  subject to edge deletions and insertions across a specified face as a crossing free drawing, in  $O(\log n)$  time. Henzinger, Italiano, and La Poutré [14] give a dynamic algorithm for maintaining a plane embedded graph subject to edge deletions and insertions across a face, while supporting queries to whether a pair of vertices presently share a face in the embedding, in  $O(\log^2 n)$  time per operation. Holm and Rotenberg [15] expand on the result of [14] by additionally supporting *flips* (operations which change the combinatorial embedding but not  $G$ ). They show how to support all operations in  $O(\log^2 n)$  time, rejecting operations that would violate the planarity of  $\mathcal{E}(G)$ . Their data structure supports *linkability* queries which, for a pair of vertices, report a sequence of faces across which they are linkable, or singular flips that would make the two vertices linkable.

For dynamic planarity testing (maintaining a bit indicating whether the graph is presently planar) there has been a body of work [3, 4, 11, 12, 16, 17, 25, 26, 28, 30]. The current state-of-the-art algorithm for incremental planarity testing is by Holm and Rotenberg [17] in  $O(\log^3 n)$  worst case time per edge insertion. Here, they crucially rely upon an  $O(\log^2 n)$  fully dynamic algorithm for the dynamic maintenance of a planar embedding  $\mathcal{E}(G)$  [15].

**Contribution and organisation.** Let  $G$  be a single-source upward planar directed graph. We study how to maintain an upward combinatorial embedding  $\vec{\mathcal{E}}(G)$  subject to operations on  $\vec{\mathcal{E}}(G)$ . Updates which would violate upward planarity are recognised as such and rejected. Section 2 contains our preliminaries and formalises our setting and the type of updates and queries that we allow. In Section 3 we present our linear-size dynamic data structure that maintains  $G$  and a combinatorial representation of  $\vec{\mathcal{E}}(G)$  subject to edge insertions across a face, edge deletions, vertex splittings, edge contractions, and “flip”-operations that perform local changes to the embedding, in  $O(\log^2 n)$  time per operation. Our set of operations (see preliminaries) allow us to transform between any two upward embeddings of a single-source biconnected digraph with the same leftmost edge around the source [7, Theorem 3].

A *corner*  $c_u$  in  $\vec{\mathcal{E}}(G)$  is the gap between two edges that are consecutive in the ordering around a vertex  $v$  that share a face  $f$ . An edge insertion specifies two corners  $c_u$  and  $c_v$  that share a face  $f$ , and aims to insert the edge  $c_u \rightarrow c_v$  into  $\vec{\mathcal{E}}(G)$  (the edge directed from  $u$  to  $v$  that is inserted across  $f$  in between the two corners). In Section 3, we give a dynamic data structure that supports edge deletion and edge insertion between corners that share a face. Our data structure can efficiently test whether inserting  $c_u \rightarrow c_v$  violates upward planarity, rejecting updates that would violate upward planarity. In Section 4, we extend this data structure to support *uplinkable queries* in  $O(\log^2 n)$  time; i.e., queries that decide for vertices  $(u, v)$  whether there exists a corner pair  $(c_u, c_v)$  such that  $c_u \rightarrow c_v$  may be inserted without violating upward planarity. Specifically, our algorithm is capable of reporting all suitable corner pairs. In the negative case, Section 4 furthermore describes how to efficiently support queries whether minor changes to the embedding would allow us to insert the edge; namely, the *one-flip uplinkable* query which in  $O(\log^2 n)$  time answers whether there exists at least one flip in  $\vec{\mathcal{E}}(G)$  after which  $u$  and  $v$  are uplinkable, and outputs a flip if one exists.

**Techniques and challenges.** We build on techniques from two sources. Firstly, Holm and Rotenberg [15] show how to dynamically maintain a planar embedded graph, rejecting updates that would make the graph not planar. Secondly, Bertolazzi, Di Battista, Mannino, and Tamassia [6] define the *face-sink* graph of a combinatorially embedded single-source digraphs and prove that a realization as an upward planar drawing exists if and only if the face-sink graph satisfies a specific set of conditions. We combine insights from [15] and [6]:

- We maintain a planar combinatorial embedding  $\mathcal{E}(G)$  of  $G$  with  $O(\log^2 n)$  update time using the data structure in [15], which supports all updates.
- We make  $\mathcal{E}(G)$  into an upward combinatorial embedding  $\vec{\mathcal{E}}(G)$ . We store a data structure over  $\vec{\mathcal{E}}(G)$  to dynamically maintain the face-sink graph (and check the conditions from [6]) in  $O(\log^2 n)$  time. We verify after each update whether  $\vec{\mathcal{E}}(G)$  remains upward planar.

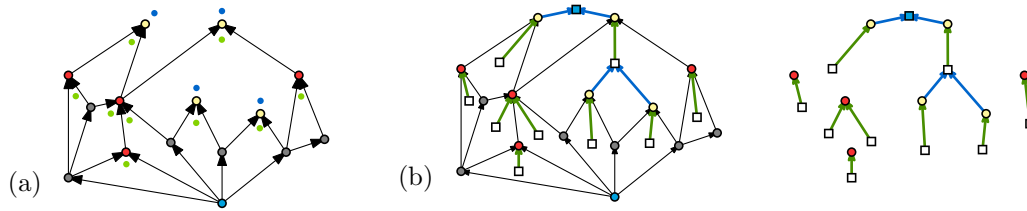
We can dynamically maintain an upward planar combinatorial embedding  $\vec{\mathcal{E}}(G)$  subject to all our updates. We also show that we can answer all queries. Holm and Rotenberg [15] provide a framework for answering these queries in planar graphs. However, we need to build on this framework to be able to apply it to upward planarity. Indeed, for undirected graphs, the edge  $(u, v)$  may be inserted whenever  $(u, v)$  share a face. For upward planar digraphs,  $u$  and  $v$  may share a face even if the edge  $u \rightarrow v$  cannot be inserted whilst preserving upward planarity. This difference creates two problems: First, Holm and Rotenberg specifically assume (and use) that  $(u, v)$  do not share a face before the one-flip linkable query. Thus, we must alter their scheme to operate without such assumption. Second, they may return  $O(n)$  candidate flips after which  $(u, v)$  share a face, from which we need to filter the flips that allow insertion of the directed edge  $u \rightarrow v$ . To achieve this, we show new geometric lemmas that uniquely characterise which faces can be part of a flip to facilitate the insertion of the directed edge  $u \rightarrow v$ . We prove that these faces must form a contiguous subsequence of the output by the data structure of Holm and Rotenberg. Subsequently, we use binary search on the candidate output to identify the sequence of flips that are part of our output.

## 2 Preliminaries

Let  $G = (V, E)$  be a directed graph (*digraph*) where  $V$  is a set of vertices and  $E$  is a set of ordered pairs of vertices. Each pair  $(u, v) \in E$  represents an edge directed from  $u$  to  $v$ . For  $u, v \in V$  we say that  $u < v$  if and only if there exists a directed path from  $u$  to  $v$  in  $G$ . Note that it is not known how to dynamically decide  $u < v$  in poly-logarithmic time. Instead, our algorithms use our current embedding to derive whether  $u < v$  when necessary. We require that  $G$  is *single-source*: each component of  $G$  has a unique vertex with only outgoing edges.

**Combinatorial embeddings.** A *combinatorial embedding*  $\mathcal{E}(G)$  of a connected graph  $G$  (as defined in [15]) specifies for every vertex  $v \in V$  a counter-clockwise ordering of the edges incident to  $v$ . This defines a set of faces  $F$  where for every  $f \in F$  there is a cyclic ordering of vertices incident to  $f$ . We define a *corner* in  $\mathcal{E}(G)$  as a 4-tuple  $(v, f, e_1, e_2)$  where  $v \in V$  is incident to face  $f$ ,  $e_1, e_2 \in E$  are both incident to  $v$  and  $f$ , and are consecutive in the counter-clockwise order around  $v$ . In the degenerate case that  $v$  is a vertex incident to a single edge  $e$  and a face  $f$ , we define the corner  $(v, f, e, e)$ . If  $G$  is not connected, a combinatorial embedding of  $G$  consists of a combinatorial embedding of each connected component.

**Upward (combinatorial) embeddings.** An *upward (combinatorial) embedding*  $\vec{\mathcal{E}}(G)$  is a combinatorial embedding  $\mathcal{E}(G)$  that additionally stores for each corner  $(v, f, e_1, e_2)$  where  $e_1$  and  $e_2$  are directed towards  $v$ , whether the corresponding angle is presently reflex (larger than 180) or convex in the embedding. Also, we require that an upward embedding marks, for each connected component, a face in the corresponding combinatorial embedding as its outer face. Finally, we require that each connected component with outer face  $h$  has exactly one corner  $(v, h, e_1, e_2)$ , where  $v$  is the source and  $e_1$  and  $e_2$  are directed from  $v$ , marked as reflex. An upward embedding  $\vec{\mathcal{E}}(G)$  is upward planar whenever there exists an upward planar drawing  $D(G)$  realising  $\vec{\mathcal{E}}(G)$ .



■ **Figure 1** (a) An upward planar drawing. Sink corners are marked with a dot, which is green for top corners. Critical vertices are red. (b) The face-sink graph is a bipartite graph. If  $\vec{\mathcal{E}}(G)$  is upward planar, it is a forest where the critical vertices and the outer face are the roots.

**Single-source digraphs, corners, and the face-sink graph.** Next, we discuss properties of digraphs and upward embeddings crucial for the remainder of the paper (see Figure 1(a)):

- A *source* (resp. a *sink*) is a vertex with only outgoing (resp. incoming) edges. The *reflex source corner* is the unique reflex corner incident to the source of a component.
- Vertices which are not a source or a sink are *internal* vertices.
- In a *sink corner*  $(v, f, e_1, e_2)$ , the edges  $e_1$  and  $e_2$  are both directed towards  $v$ . In a *source corner*  $(v, f, e_1, e_2)$ , the edges  $e_1$  and  $e_2$  are both directed away from  $v$ .
- A *top corner* is a sink corner where the angle between  $e_1$  and  $e_2$  is convex.
- A *critical vertex* is a (internal) vertex  $v$  incident to at least two incoming edges and at least one outgoing edge.

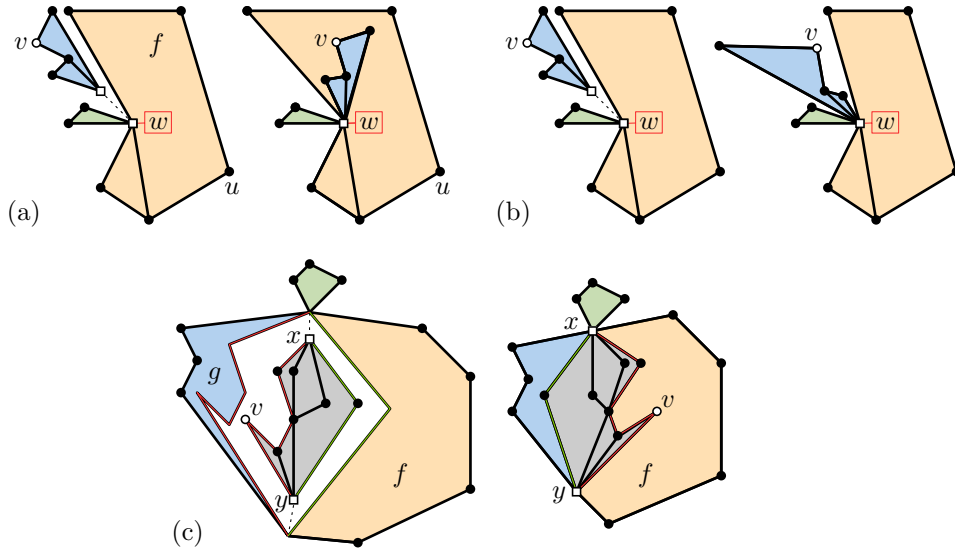
In this paper, we consider single-source digraphs (each connected component has one source). In their upward planar embeddings, in each component the outer face is incident to the source. For an upward embedding  $\vec{\mathcal{E}}(G)$  of a single-source digraph  $G$ , Bertolazzi *et al.* [6] define its face-sink graph  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . We slightly modify their definition to match the above-defined concepts and ensure that  $\mathcal{F}(\vec{\mathcal{E}}(G))$  is a directed graph instead. Formally,  $\mathcal{F}(\vec{\mathcal{E}}(G))$  is a bipartite graph between the vertices  $V$  and faces  $F$  of  $\vec{\mathcal{E}}(G)$  (Figure 1(b)). There is a directed edge from  $v \in V$  to  $f \in F$  whenever they share a sink corner that is *not* a top corner, and a directed edge from  $f$  to  $v$  whenever they share a top corner. They show:

► **Theorem 1** (Theorem 1 in [6], Fact 2+3). *Let  $G$  be a single-source digraph. An upward (combinatorial) embedding  $\vec{\mathcal{E}}(G)$  of  $G$  is upward planar if and only if:*

- $\mathcal{F}(\vec{\mathcal{E}}(G))$  is a forest  $\mathcal{T}^*, \mathcal{T}_1, \dots, \mathcal{T}_m$  where each non-outer face has out-degree 1,
- $\mathcal{T}^*$  does *not* have internal vertices and its root is incident to the unique reflex source corner of  $\vec{\mathcal{E}}(G)$ , and
- $\mathcal{T}_1, \dots, \mathcal{T}_m$  each have as root the unique internal vertex in  $\mathcal{T}_i$ , which is critical.

**Articulation slides, twists, and separation flips.** The most basic dynamic update to an embedded graph, is to alter its embedding. We characterise three important changes to an upward planar embedding that can be used as building blocks for transformations that change it into another upward planar embedding. This is highly motivated by the dynamic updates of adding and removing edges, because the current embedding may not be compatible with the edge we want to add. We will define three embedding changes.

For any (undirected) graph  $G$ , an *articulation vertex*  $w$  is a vertex whose removal separates a connected component  $G'$  of  $G$  into at least two connected components  $C_1, C_2, \dots$ : the *articulation* components. In an embedded planar graph  $\mathcal{E}(G)$ , articulation vertices have at least two corners incident to the same face  $f$ . Given an articulation vertex  $w$ , let  $C_w(v)$



■ **Figure 2** (a) An articulation vertex  $w$  with  $v$  in the blue component  $B$ .  $F_s = (f, w, v)$  slides  $B$  into  $f$ . (b)  $F_t = (w, v)$  mirrors the blue component  $B$ . (c) The square vertices form a separation pair  $(x, y)$ . We show the separation flip  $F_t = (f, g, x, y, v)$  after which  $v$  becomes incident to  $f$ .

be the component containing  $v$  in  $G \setminus w$ . A *separation pair*  $(x, y)$  in a graph is any pair of vertices whose removal separates a connected component into at least two components. In a planar graph,  $(x, y)$  is a separation pair if  $x$  and  $y$  share at least two faces  $f$  and  $g$ .

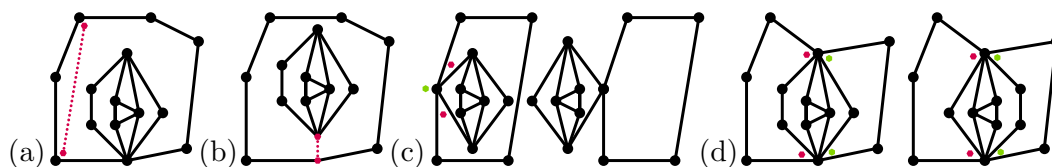
The following operations change the embedding  $\vec{\mathcal{E}}(G)$  while leaving the graph unchanged:

- Given an articulation vertex  $w$  incident to  $f$  and a vertex  $v \neq w$ , for an *articulation slide*  $F_s = (f, w, v)$  we cut the component  $C_w(v)$  from  $w$  and embed  $C_w(v)$  into  $f$  by merging it back at  $w$  (see Figure 2(a)). We intuitively refer to it as *sliding*  $C_w(v)$  into  $f$ .
- Given an articulation vertex  $w$  and a vertex  $v \neq w$ , for an *articulation twist*  $F_t = (w, v)$  we isolate the component  $C_w(v)$  by cutting through  $w$ , mirror its embedding, and merge it back at  $w$  (see Figure 2(b)).
- Given a separation pair  $(x, y)$  sharing faces  $f, g$  and a vertex  $v \neq x, y$ , for a *separation flip*  $F = (f, g, x, y, v)$  we split  $x$  and  $y$  along the corners incident to  $f$  and  $g$ , mirror the embedding of the component containing  $v$ , and merge it back into  $G$  (see Figure 2(c)).

**Update operations.** We dynamically maintain an upward embedding  $\vec{\mathcal{E}}(G)$  (i.e., a combinatorial embedding that specifies for every connected component the outer face, and for every sink corner a Boolean indicating whether the angle is reflex) subject to a list of updates that we specify below. We say that an update violates upward planarity if it cannot be accommodated such that  $\vec{\mathcal{E}}(G)$  remains upward planar. *Combinatorial updates* modify the graph  $G$  (see Figure 3):

- **Insert**( $c_u, c_v$ ) for an ordered pair of corners, inserts  $c_u \rightarrow c_v$  into  $\vec{\mathcal{E}}(G)$ .
- **Delete**( $e$ ) for an edge  $e \in E$ , removes it from  $\vec{\mathcal{E}}(G)$ .
- **Cut**( $c_1, c_2$ ) for two corners  $c_1$  and  $c_2$  incident to a vertex  $v$ , replaces  $v$  by  $v_1$  and  $v_2$  connected by an edge  $(v_1, v_2)$  (choosing the edge's direction that preserves upward planarity). Each of these two vertices becomes incident to a unique consecutive interval of edges incident to  $v$  that is bounded by  $c_1$  and  $c_2$ .
- **Contract**( $e$ ) contracts an edge  $e \in E$ , merging the two endpoints.





■ **Figure 3** Operations: (a) insertion, (b) cut, (c) articulation-slide, and (d) separation-flip.

Recall that  $G$  has one source per connected component. We maintain connected components in separate data structures, which are merged/split with updates that merge/split the respective connected components. We also support *embedding updates* that change  $\vec{\mathcal{E}}(G)$ :

- **Mirror** $(v, c_1, c_2)$  mirrors the flip component containing  $v$  that is separated from  $G$  during  $\text{Cut}(c_1, c_2)$  (inverting the ordering of edges incident to  $v$  between  $c_1$  and  $c_2$ ).
- **Articulation-slide** $(f, w, v)$  executes the articulation slide  $F_s = (f, w, v)$ .
- **Articulation-twist** $(w, v)$  executes the articulation twist  $F_t = (w, v)$ .
- **Separation-flip** $(f, g, x, y, v)$  executes the separation flip  $F = (f, g, x, y, v)$ .
- **Purl** $(C, c_f, h)$  for a connected component  $C$ , and a corner  $c_f$  incident to the source and to a face  $f$ , sets  $f$  to be the outer face, and changes the reflex source corner to be  $c_f$ . We support this only under restricted conditions.

**Queries.** Two corners  $c_u, c_v$  in  $\vec{\mathcal{E}}(G)$  are *uplinkable* in  $\vec{\mathcal{E}}(G)$  whenever inserting the edge  $c_u \rightarrow c_v$  does not violate the upward planarity or the single-sourceness property. A vertex pair  $(u, v)$  is uplinkable whenever uplinkable corners  $(c_u, c_v)$  exist. We support the following queries in  $O(\log^2 n)$  time:

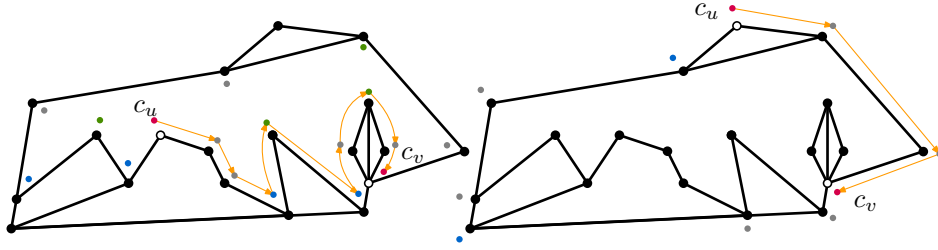
- **UpLinkable** $(u, v)$  identifies all corner pairs  $(c_u, c_v)$  in  $\vec{\mathcal{E}}(G)$  where  $c_u \rightarrow c_v$  is uplinkable. It can report these  $k$  pairs in  $O(k)$  additional time.
- **Slide-UpLinkable** $(u, v)$  identifies all slides  $F_s = (f, w, v)$  and  $F'_s = (f', w, u)$  where, after each articulation slide,  $u \rightarrow v$  is uplinkable in  $f$  (or  $f'$ ).
- **Twist-UpLinkable** $(u, v)$  reports the at most two articulation twists  $F_t = (w, v)$  and  $F'_t = (w, u)$  where, after each articulation twist,  $u \rightarrow v$  are uplinkable.
- **Separation-UpLinkable** $(u, v)$  reports for  $u \rightarrow v$  not uplinkable, a separation flip  $F = (f, g, x, y, v)$  or  $F' = (f', g', x', y', u)$  after which  $u$  and  $v$  are uplinkable (if it exists).

Combining the latter three queries, we achieve the upward-planar equivalent of One-Flip-Linkable, as defined in Holm and Rotenberg [15]:

- **One-Flip-UpLinkable** $(u, v)$  reports whether there exist a face  $f$  where there exists a sequence of flips that all have  $f$  as an argument, after which  $u \rightarrow v$  are uplinkable in  $f$ . Unlike [15], we do not assume that  $u$  and  $v$  do not share a face, and lifting this assumption is indeed important (e.g., in Figure 2(a),  $(u, v)$  share the outer face, but they are only uplinkable after flipping the blue component into  $f$  and then mirroring it in  $f$ ).

**Corner properties.** We now define some more technical concepts.

► **Definition 2.** Let  $c_u, c_v$  be two corners incident to a face  $f \in \vec{\mathcal{E}}(G)$ . If  $f$  is not the outer face, or when  $f$  is the outer face and  $u$  and  $v$  share a connected component in  $G$ , we define a directed path  $\pi(c_u, c_v)$  in  $f$  from  $c_u$  to  $c_v$  as any consecutive sequence of corners incident to  $f$  that starts at  $c_u$  and ends at  $c_v$  (see Figure 4). The edges of the path  $\pi(c_u, c_v)$  are the edges shared between two consecutive corners. Note that the direction of  $\pi(c_u, c_v)$  is independent of the direction of the edges in  $G$ .



■ **Figure 4** A simple polygon where we marked source corners in blue, sink corners in green and internal corners in grey. We show one of the two paths  $\pi(c_u, c_v)$  directed from  $c_u$  to  $c_v$ .

► **Observation 3.** Let  $G$  be a single-source digraph, and let  $\vec{\mathcal{E}}(G)$  be an upward planar embedding. Let  $u$  and  $v$  be vertices that lie in different connected components  $C_u$  and  $C_v$  in  $G$ . The corner pair  $(c_u, c_v)$  is uplinkable if and only if  $c_u$  is not a top corner and  $c_v$  is the reflex source corner.

### 3 Dynamic Upward Planar Embeddings

**Defining our data structure.** Let  $G$  be a single-source upward planar digraph, and let  $\vec{\mathcal{E}}(G)$  be some upward embedding of  $G$ . The foundation of our data structure is the data structure by Holm and Rotenberg [15] for undirected graphs, which can maintain a planar embedding  $\mathcal{E}(G)$  in  $O(\log^2 n)$  subject to all our updates. In addition, we maintain the following new data structure in  $O(\log^2 n)$  time per update. We use it to reject updates that would violate the upward planarity of  $\vec{\mathcal{E}}(G)$ . Our data structure dynamically maintains the face-sink graph  $\mathcal{F}(\vec{\mathcal{E}}(G))$  of  $G$  in  $O(\log^2 n)$  time per update. We store:

- (a) For each connected component  $C$  of  $G$ , an arbitrary spanning tree  $T_C$  of  $C$  stored in a *top tree*  $e(T_C)$ , which is a balanced binary tree over the edges of  $T_C$  (see the full version).
- (b) For each face  $f$  of a connected component  $C$ , a leaf-based balanced binary tree  $T_f$  on the corners incident to  $f$ , in their clockwise ordering around  $f$ . We mark the root-to-leaf path in  $T_f$  to the unique top corner or the unique reflex source corner.
- (c) For every vertex  $v$ , a leaf-based balanced binary tree  $T_v$  of corners incident to  $v$ . We store a Boolean indicating whether  $v$  is critical.
- (d) For every tree  $\mathcal{T}$  in  $\{\mathcal{T}^*, \mathcal{T}_1, \dots, \mathcal{T}_m\}$  of  $\mathcal{F}(\vec{\mathcal{E}}(G))$ , a top tree  $e(\mathcal{T})$  on the edges of  $\mathcal{T}$ .

Each corner and each vertex in  $\vec{\mathcal{E}}(G)$  maintains a pointer to their location in the above data structures. If  $c_x$  is a sink corner or  $x$  is the single source of a connected component, it maintains a Boolean indicating whether its angle is reflex.

► **Theorem 4.** We can dynamically maintain  $\vec{\mathcal{E}}(G)$  and  $\mathcal{F}(\vec{\mathcal{E}}(G))$  subject to *Insert, Delete, Cut, Contract, Mirror, Articulation-slide, Articulation-twist, Separation-flip, and Purl* in  $O(\log^2 n)$  time per update, rejecting updates that violate the upward planarity of  $\vec{\mathcal{E}}(G)$ .

**Proof.** By Holm and Rotenberg [15], we can maintain a combinatorial embedding in  $O(\log^2 n)$  time as long as it remains planar. In addition, we show that we can maintain an upward embedding  $\vec{\mathcal{E}}(G)$  and  $\mathcal{F}(\vec{\mathcal{E}}(G))$  and our data structure in  $O(\log^2 n)$  time. Our data structure then verifies whether  $\vec{\mathcal{E}}(G)$  remains upward planar by testing whether the conditions for Theorem 1 are met. If not, it rejects the update accordingly (undoing all changes to  $\vec{\mathcal{E}}(G)$  in  $O(\log^2 n)$  time). We show how to handle edge insertions first.



**Insert**( $c_u, c_v$ )

We try inserting the edge  $c_u \rightarrow c_v$ . We select an arbitrary edge incident to  $c_u$  and traverse it to the root of its top tree. Doing the same for  $c_v$  allows us to detect in  $O(\log n)$  time whether  $u$  and  $v$  share a connected component. If they do not, we apply Observation 3 to test whether  $c_u \rightarrow c_v$  may be inserted without violating upward planarity in  $O(1)$  time. Else, we reject the update. Similarly, if  $u$  and  $v$  share a connected component, we test whether  $(c_u, c_v)$  share a face in  $O(1)$  time and reject the insertion if not.

Next, we insert  $c_u \rightarrow c_v$  and update our data structure. Through our data structure, we will test the conditions of Theorem 1, verifying whether inserting  $c_u \rightarrow c_v$  violates upward planarity. Note that  $c_u \rightarrow c_v$  deletes  $c_u$  and  $c_v$  and creates four new corners, where at most two can be sink corners (incident to  $v$ ). Sink corners have a Boolean indicating if they are reflex or convex. Since we create at most two sink corners, there are at most four Boolean combinations (in fact, at most three, since we cannot create two new reflex sink corners). We try all combinations as an update to our data structure, and keep an update that does not violate upward planarity (if any such update exists). Having fixed which sink corners are reflex and which are convex, we update the four (a-d) ancillary data structures we store:

**We first assume that  $u$  and  $v$  lie in the same connected component  $C$ .** In this case,  $c_u$  and  $c_v$  share a face  $f$  that gets split into two faces  $f_b$  and  $f_t$ . If  $f$  is marked as the outer face of  $C$  then either  $t_b$  or  $t_f$  needs to become the outer face. We try both combinations.

- (a) Since  $u$  and  $v$  are already connected, the insertion of  $c_u \rightarrow c_v$  does not change the spanning tree  $T_C$  and therefore it does not change the top tree  $e(T_C)$ .
- (b) We remove the corners  $c_u$  and  $c_v$  from  $T_f$ . If any of them were the top corner of  $f$ , we unmark the root-to-leaf path in  $T_f$ . The edge  $c_u \rightarrow c_v$  splits  $f$  into two faces  $(f_b, f_t)$ . The face  $f_b$  (and  $f_t$ ) is incident to a contiguous sequence  $Z$  of corners incident to  $f$ . We can obtain any sequence of corners  $Z$  from  $T_f$  in  $O(\log n)$  time as  $O(\log n)$  balanced subtrees, where each subtree stores a contiguous sequence of  $Z$ .  $O(\log n)$  balanced binary trees, with contiguous domains, may be merged using the standard balanced binary tree merge algorithm to create  $T_{f_b}$  in  $O(\log^2 n)$  total time. We obtain  $T_{f_t}$  by removing  $Z$  from  $T_f$  in  $O(\log^2 n)$  total time. Finally, we insert two new corners into  $T_{f_b}$  (and  $T_{f_t}$ ). If we insert a top corner  $c$ , we mark the root-to-leaf path in  $T_{f_b}$ . If we try to mark two root-to-leaf paths in one tree  $T_{f_b}$ , then  $f_b$  has out-degree two in the face-sink graph and we violate Theorem 1. Thus, we reject the update. Let  $f_t$  be chosen as the outer face. If  $T_{f_t}$  contains a marked root-to-leaf path, then the outer face is incident to a top corner, which violates Theorem 1 – we reject the update.
- (c) We insert the four new corners into  $T_u$  and  $T_v$  in  $O(\log n)$  time. Finally, we update the Booleans of  $u$  and  $v$ . The insertion may result in vertex  $u$  becoming an internal vertex in  $G$ . It may also happen that  $v$  was an internal vertex in  $G$  and after the insertion,  $v$  became incident to a top corner (and hence critical). We test whether  $u, v$  are internal in  $O(\log n)$  additional time and adjust the Booleans accordingly.
- (d) Finally, we update  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . The new faces  $f_b$  and  $f_t$  are each a new node in  $\mathcal{F}(\vec{\mathcal{E}}(G))$  and we delete the node corresponding to the shared face  $f$ . The parent nodes of these faces  $f_b$  and  $f_t$  correspond to top corners in  $T_{f_b}$  and  $T_{f_t}$ , respectively, unless  $f_t$  is the outer face. Given a top corner  $c^*$  in a face  $f'$ , we obtain the vertex  $w$  incident to  $c^*$  in  $O(1)$  time and insert  $f'$  as a child of  $w$  in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . These operations insert  $O(1)$  new edges into trees in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ , which are supported in  $O(\log n)$  time.

In the graph  $\mathcal{F}(\vec{\mathcal{E}}(G))$ , the children of  $f_b$  and  $f_t$  can be obtained by separating the children of  $f$  around two corners in any ordered embedding of  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . This operation is also supported by top trees in  $O(\log n)$  time. Since we only inserted child-to-parent edges,  $\mathcal{F}(\vec{\mathcal{E}}(G))$  is still a forest. We test whether each tree  $\mathcal{T}$  of  $\mathcal{F}(\vec{\mathcal{E}}(G))$  has a unique critical vertex (or the outer face) as its root. Specifically, this operation affects at most two trees of  $\mathcal{F}(\vec{\mathcal{E}}(G))$  and at most two vertices. We check if these constantly many objects still satisfy the conditions of Theorem 1 in  $O(\log n)$  time and reject the update otherwise.

**We next assume that  $u$  and  $v$  lie in different connected components  $C_u$  and  $C_v$ .** In this case,  $c_u$  lies in a face  $f_u$  and  $c_v$  is the reflex source corner (Observation 3). The faces  $f_u$  and  $f_v$  then get merged into a single face  $f_u$ . We delete the reflex Boolean of  $c_v$ , which was required by Observation 3, and update (a)+(b)+(c)+(d):

- (a) We merge  $T_{C_1}$  and  $T_{C_2}$  into a tree  $T_C$  through the edge  $c_u \rightarrow c_v$ . We then merge the top trees  $e(T_{C_1})$  and  $e(T_{C_2})$  in  $O(\log n)$  time.
- (b) By definition, the tree  $T_{f_v}$  has no marked root-to-leaf path. We split  $T_{f_v}$  into two trees around the corner  $c_v$ . We then merge both trees into  $T_{f_u}$  in  $O(\log n)$  time. Finally, we remove  $c_u$  from  $T_{f_u}$  and insert the four new corners. Just as previously, whenever we insert a top corner we mark the root-to-leaf path in  $T_{f_u}$  accordingly and when we try to mark two paths, we reject the update. If  $f$  is not marked as the outer face of  $C_u$  and after the corner insertions,  $T_f$  has no marked root-to-leaf path, then in the face-sink graph  $f$  is a root of a tree without being the outer face, and we violate Theorem 1. Thus, we reject the update.
- (c) This is identical to the case where  $u$  and  $v$  share a connected component.
- (d) Finally, we update  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . Consider the node  $f_v$  in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . Since  $f_v$  was the outer face of  $C_v$ , this node was the root of a tree. By merging  $f_v$  into  $f$ , all children of  $f_v$  become children of  $f_v$  instead. A top tree supports this merge in  $O(\log n)$  time. Since  $v$  was the source of  $C_v$ , no top corners are introduced. It may be that  $c_u$  was a reflex sink corner, where  $c_u$  now becomes an internal vertex, in which case we remove the edge from  $c_u$  to  $f_u$  in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . Since these operations change  $O(1)$  trees in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ , we may test in  $O(\log n)$  time whether  $\mathcal{F}(\vec{\mathcal{E}}(G))$  still satisfies the conditions of Theorem 1.

### Delete( $e$ )

Let  $C$  be the connected component containing  $e = uv$ . We first check if  $e$  was an edge in the spanning tree  $T_C$ . If it was, we risk that deleting  $e$  splits  $C$  into two connected components. The data structure in [15] that we use to store the combinatorial planar embedding supports checking whether this is the case, and restoring  $T_C$  in the negative case, in  $O(\log^2 n)$  time.

If  $e$  does not split  $C$  into two connected components, the delete operation is the immediate inverse of the insertion operation; we need only perform the easy check that this does not create a new source in  $C$ , since such deletions violate the conditions and must be rejected. Thus, we consider the special case where deleting an edge  $e$  splits a connected component  $C$  into two connected components  $C_u$  and  $C_v$ . Observe that, in this case,  $e$  is incident to only one face  $f$ . Afterward, the vertex  $u$  is incident to a face  $f_u$  and the vertex  $v$  is incident to a vertex  $f_v$  where  $f_v$  is the outer face of  $C_v$ . We update our data structure:

- (a) We split the tree  $T_C$  into two trees  $T_{C_1}$  and  $T_{C_2}$ . Thus, we need to obtain the top trees  $e(T_{C_1})$  and  $e(T_{C_2})$ . Top trees support splits in  $O(\log n)$  time (see the full version).
- (b) We split the tree  $T_f$  into fewer than eight subtrees, by cutting at the four corners incident to  $e$  and  $f$ . Each subtree is either contained in  $C_u$ , or in  $C_v$ . All subtrees contained in  $C_u$  get merged in  $O(\log n)$  time to create  $T_{f_u}$  and all subtrees contained in  $C_v$  get

merged in  $O(\log n)$  time to create  $T_{f_u}$ . Finally, we insert a new corner  $c_u$  into  $T_{f_u}$  and a new corner  $c_v$  into  $T_{f_v}$ . If the corner  $c_u$  is a sink corner, then by Observation 3, its angle must be reflex, and we specify its Boolean as such. The corner  $c_v$  must, Observation 3, be the reflex source corner of its connected component (we update its Boolean). If it is not a source corner, we reject the update. If  $f$  was the outer face of  $C$  then  $f_u$  becomes the outer face of  $C_u$ . Otherwise, we test  $f_u$  is incident to a top corner in  $O(\log n)$  time by checking the marked root-to-leaf path in  $T_{f_u}$ . If it is not, then the conditions of Theorem 1 are violated and we reject the update. Similarly, we reject the update whenever  $T_{f_u}$  contains a top corner.

- (c) The trees  $T_u$  and  $T_v$  get updated in  $O(\log n)$  time by deleting  $e$  from them. If  $u$  was a critical vertex, we check whether  $u$  is still a critical vertex (by checking if  $u$  is still incident to an outgoing edge) in  $O(\log n)$  time and update the Boolean accordingly.
- (d) In the face-sink graph, we insert  $f_u$  and  $f_v$  where  $f_v$  is now the root of a tree. The children of  $f$  get split amongst  $f_u$  and  $f_v$ . Since these children are contiguous along  $T_f$ , a top tree supports this operation in  $O(\log n)$  time. If  $u$  was a critical vertex before removing  $e$ , but no longer is a critical vertex after removing  $e$ , we insert the edge from  $u$  to  $f_v$  in the face-sink graph. Since this operation affects at most two trees in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ , we can test whether the face-sink graph still satisfies the conditions of Theorem 1 in  $O(\log n)$  time. Finally, we update the face-sink graph, and mark  $c_v$  as the reflex source corner of  $C_v$ .

### Cut( $c_1, c_2$ ) / Contract( $e$ )

We describe how to perform the cut operation, as contract is its easier inverse. Let  $(c_1, c_2)$  be incident to some vertex  $v$  in a connected component  $C$ . Let  $c_1$  be incident to a face  $f_1$  and  $c_2$  be incident to a face  $f_2$ . It may be that  $f_1 = f_2$ . We assume that  $f_1 \neq f_2$ , since this is the more difficult case. Denote by  $C_t$  and  $C_b$  the two sequences of corners incident to  $v$  that are in between  $c_1$  and  $c_2$ . The Cut( $c_1, c_2$ ) operation creates two new vertices  $v_t$  and  $v_b$  from  $v$ , incident to  $C_t$  and  $C_b$  respectively; creating two new corners  $c_t$  and  $c_b$ . It then connects  $(v_b, v_t)$  with either the edge  $c_b \rightarrow c_t$ , or the edge  $c_t \rightarrow c_b$ , creating four new corners (two incident to  $f_1$  and two incident to  $f_2$ ). We try both options and update our data structure. We then use the face-sink graph  $\mathcal{F}(\vec{\mathcal{E}}(G))$  to test whether  $\vec{\mathcal{E}}(G)$  is still upward planar.

We update aspects (a)+(b)+(c)+(d) of our data structure.

- (a) We remove  $v$  from  $T_C$ , and replace it with the vertices  $v_t$  and  $v_b$  that are connected by an edge. This way, we can update the top tree  $e(T_C)$  in  $O(\log n)$  time.
- (b) We update  $T_{f_1}$  and  $T_{f_2}$ . The cut operation deletes the corners  $c_1$  and  $c_2$  which we may remove from  $T_{f_1}$  and  $T_{f_2}$  in  $O(\log n)$  time. If either of them was a top corner, we update the marked root-to-leaf path in the corresponding tree. We then insert two new corners into  $T_{f_1}$  and two new corners into  $T_{f_2}$ . If a corner is a sink corner, it must specify a Boolean whether it is convex or reflex. We simply try for these four corners all constantly many Boolean combinations and perform the update as such – rejecting updates that violate upward planarity. Through the updated  $T_{f_1}$  and  $T_{f_2}$  we test whether  $f_1$  and  $f_2$  have a unique top corner. If not, we violate the conditions in Theorem 1 and reject the update.
- (c) The trees  $T_{v_b}, T_{v_t}$  are both incident to a sequence of edges that is contiguous in the clockwise order around  $v$ . Given  $(c_1, c_2)$ , we obtain this sequence in  $O(\log n)$  time as  $O(\log n)$  subtrees of  $T_v$ . We merge these  $O(\log n)$  subtrees in  $O(\log^2 n)$  total time using the standard merge of balanced binary trees; obtaining  $T_{v_b}$  and  $T_{v_t}$ . Finally, we insert the edge  $(v_b, v_t)$  into both trees. If the edge  $(v_b, v_t)$  is directed from  $v_b$  to  $v_t$ , it may be that  $v_b$  is a critical vertex. We test this in  $O(\log n)$  time. If the edge  $(v_b, v_t)$  is directed from  $v_t$  to  $v_b$ , it may be that  $v_t$  is a critical vertex. We also test this in  $O(\log n)$  time.

- (d) We update the face-sink graph  $\mathcal{F}(\vec{\mathcal{E}}(G))$  analogous to insertions, connecting  $f_1$  and  $f_2$  to any sink corners in the graph. This updates  $O(1)$  trees in  $\mathcal{F}(\vec{\mathcal{E}}(G))$  which can be done in  $O(\log n)$  time. We then test if the face-sink graph meets the conditions of Theorem 1 in  $O(\log n)$  time by traversing to the root of all  $O(1)$  affected trees.

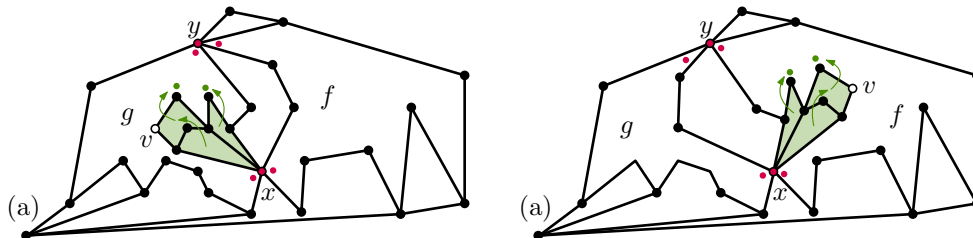
**Articulation-Slide( $f, w, v$ ) / Articulation-Twist( $w, v$ )**

Note that all combinatorial changes incurred by an articulation slide or twist, also occur in a separation flip. Specifically, an articulation slide may alter at most four corners in  $\vec{\mathcal{E}}(G)$ , and causes a path in  $g$  to become a path in  $f$ . An articulation twist selects mirrors a component. A separation flip performs all these three combinatorial changes, twice. Our procedure for the separation flip will specify all the changes to our data structure for these occurrences, and thus for the articulation slide and twist also.

**Separation-flip( $f, g, x, y, v$ )**

Let the face  $g$  share corners  $c_g^x$  and  $c_g^y$  with  $x$  and  $y$ , respectively. Similarly, let  $f$  share corners  $c_f^x$  and  $c_f^y$ . We illustrate the operation by Figure 5. We update aspects (a)-(d):

- (a) Let  $C$  be the connected component that contains  $x, y, v$ . The tree  $e(T_C)$  is a top tree over an embedded arbitrary spanning tree of  $C$ . Moreover, the combinatorial embedding of the spanning tree matches that of  $\vec{\mathcal{E}}(G)$  (i.e., all edges in the spanning tree that are incident to a vertex  $x$  have the same clockwise order as the vertices incident to  $x$  in  $\vec{\mathcal{E}}(G)$ ). This operation selects the vertices  $x$  and  $y$  of the spanning tree, selects a set of edges  $C(x)$  (respectively  $C(y)$ ) that are incident to  $x$  and inverts their order in the embedding. Top trees support this inversion operation in  $O(\log^2 n)$  time [15] in the exact same way as we will support it in (d) two paragraphs ahead.
- (b) The corners  $c_g^x$  and  $c_g^y$  bound a contiguous sequence  $C(1)$  of corners in  $T_g$  that after the flip become incident to  $f$  instead. We obtain this subsequence as at most  $O(\log n)$  binary subtrees of  $T_g$ . We test if  $C(1)$  contains a top corner in  $O(\log n)$  time, by checking the marked root-to-leaf path in  $T_g$ . We merge these  $O(\log n)$  trees into  $T_f$  in  $O(\log^2 n)$  total time and update its marked root-to-leaf path. If we detect that afterwards  $T_f$  is incident to two top corners, we reject the update. We do the same for the corners between  $c_f^x$  and  $c_f^y$ .
- (c) The tree  $T_x$  stores all edges incident to  $x$  in their cyclical order. The separation flip selects a contiguous sequence  $C(x)$  of edges in between  $c_g^x$  and  $c_f^x$  and inverts the order of the edges into  $C(1)$ . We support this operation through the following trick. We maintain two copies of  $T_x$ : one where the leaves are sorted in clockwise order and one where they are sorted in counter-clockwise order. In both copies, the sequence  $C(x)$  consists of at



■ **Figure 5** A separation flip  $F = (f, g, x, y, v)$ . Note that the green subtree under  $g$  in the face-sink graph becomes a subtree under  $f$  instead.

most  $O(\log n)$  subtrees. To invert the order of  $C(x)$  order, we simply interchange these subtrees and rebalance both trees in  $O(\log^2 n)$  total time. We do the same for the tree  $T_y$ . For all other trees in the flip component  $C_2$  separated by  $(x, y)$ , we still have a tree for their clockwise and counter-clockwise order (although they have interchanged) and we hence do not have to update them.

- (d) We update  $\mathcal{F}(\vec{\mathcal{E}}(G))$  as follows: Let  $f$  be contained in  $\mathcal{T}_f$  and  $g$  be contained in  $\mathcal{T}_g$ . All source corners (which are not top corners) in  $C(1)$  receive  $f$  as their parent instead of  $g$ . This operation is *directly* supported by our top trees over  $\mathcal{T}_f$  and  $\mathcal{T}_g$  in  $O(\log n)$  time. The separation flip destroys the corners  $c_g^x, c_g^y, c_f^x, c_f^y$  and replaces them with four new corners. We compute these in  $O(1)$  time and check whether they are sink corners. If so, then we insert the corresponding relation into  $\mathcal{F}(\vec{\mathcal{E}}(G))$  in  $O(\log n)$  time per new corner. Finally, if  $f$  and  $g$  swapped parents in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ , we execute this swap in  $O(\log n)$  time.

**Mirror**( $v, c_1, c_2$ ). The mirror operation requires us to update (d) in the same manner as the separation flip.

### Purl( $C, c_f, h$ )

The Purl operation specifies a connected component  $C$ , a face  $f$ , a corner  $c_f$ , and the outer face  $h$  of  $C$ . We allow this operation only when the following conditions hold (1) the top corner  $c_w$  of  $f$  is incident to a vertex  $w$ , such that there exists a sink corner  $c'_w$  incident to  $w$  and  $f$ , and (2) the corner  $c_f$  is a source corner incident to the source of  $G$ . These conditions hold only if the face-sink graph  $\mathcal{F}(\vec{\mathcal{E}}(G))$  contains an edge from  $f$  to  $w$ , and an edge from  $w$  to  $h$ . We update the aspects of our data structure:

- (a) The spanning tree  $T_C$  remains unchanged and thus  $e(T_C)$  remains unchanged.  
 (b) We set the corner  $c'_w$  to be a top corner and mark the root-to-leaf path in  $T_f$  in  $O(\log n)$  time. We set the corner  $c_w$  to be a reflex corner and unmark the root-to-leaf path in  $T_h$  in  $O(\log n)$  time.  
 (c) By definition,  $w$  was not a critical vertex and thus this aspect remains unchanged.  
 (d) Finally, we replace the path  $h \rightarrow w \rightarrow f$  in  $\mathcal{F}(\vec{\mathcal{E}}(G))$  by  $f \rightarrow w \rightarrow h$ ; satisfying all conditions of Theorem 1. ◀

## 4 Supporting uplinkability queries

We show how we support uplinkability queries. Specifically, for each of our queries we spend  $O(\log^2 n)$  time such that afterwards, we can report in  $O(k)$  time the first  $k$  items of the output. The exception is the Separation-UpLinkable( $u, v$ ) query where in  $O(\log^2 n)$  time we return one separation flip if any valid one exists. We maintain the same data structure as in Section 3 which includes the structure of Holm and Rotenberg [15] maintaining a planar (undirected) graph  $G'$  and its combinatorial embedding  $\mathcal{E}(G')$  subject to linkability queries. They [15] dynamically maintain  $\mathcal{E}(G')$  supporting the following queries in  $O(\log^2 n)$  time:

- **Linkable**( $u, v$ ) returns for two vertices  $u, v$  all corner pairs  $(c_u, c_v)$  where the edge  $(c_u, c_v)$  may be introduced without violating planarity in a data structure  $S$ .  $S$  stores all pairs in their clockwise order around  $u$  and  $v$ . Hence, for any  $i$ , it can return the  $i$ 'th pair in  $S$  in  $O(\log n)$  time and report all pairs up to the  $i$ 'th one in  $O(i)$  time.
- **Slide-Linkable**( $u, v$ ) returns the set of all articulation slides  $F_s = (f, w, v)$  where after  $F_s$ ,  $u$  and  $v$  share a face  $f$ . The vertex  $w$  is the articulation vertex of the slide. This output can, for any  $i$ , return the  $i$ 'th slide (around  $w$ ) in  $O(\log n)$  time and report  $i$  consecutive articulation slides in  $O(i)$  time.

- **Separation-Linkable( $u, v$ )** returns for two vertices  $u, v$  for which  $\text{Linkable}(u, v)$  and  $\text{Slide-Linkable}(u, v)$  are empty, a separation flip after which  $u$  and  $v$  share a face.

**The key difference between linkability and uplinkability.** In a planar embedding  $\mathcal{E}(G)$ , two vertices are linkable across any face they share. In an upward embedding  $\vec{\mathcal{E}}(G)$ , this is not true, which significantly complicates uplinkability queries. To answer  $\text{UpLinkable}(u, v)$  and  $\text{Slide-UpLinkable}(u, v)$  we need to identify the faces across which  $u \rightarrow v$  may be inserted and become uplinkable after a slide, respectively. For  $\text{Separation-UpLinkable}(u, v)$ , we find from all possible separation flips, one making  $u$  and  $v$  uplinkable.

**UpLinkable( $u, v$ ).** reports all corner pairs  $(c_u, c_v)$  where introducing  $c_u \rightarrow c_v$  does not violate upward planarity. The following lemma specifies which corner pairs are uplinkable:

► **Lemma 5.** *Let  $G$  be a single-source digraph and  $\vec{\mathcal{E}}(G)$  be an upward planar embedding. Let  $c_u, c_v$  be two corners of face  $f$  such that  $u$  and  $v$  share a connected component in  $G$ . If  $f$  is the outer face, let  $\pi(c_u, c_v)$  be the path in  $f$  that does not contain the reflex source corner of the connected component. Else, let  $\pi(c_u, c_v)$  be the path in  $f$  that does not contain the top corner of  $f$ . The two corners  $(c_u, c_v)$  in  $f$  are uplinkable if and only if:*

- (i)  $c_u$  is not a top corner and
- (ii) the path  $\pi(c_u, c_v)$  ends with an edge directed to  $v$ .

**Proof.** If  $c_u$  is a top corner, then any edge  $c_u \rightarrow c_v$  violates upward planarity. So let  $c_u$  not be a top corner. We show that  $c_u \rightarrow c_v$  may be inserted across  $f$  without violating upward planarity if and only if Condition (ii) holds. We make a case distinction based on the edges incident to  $c_v$ . For each case, we show whether  $c_u \rightarrow c_v$  may be inserted checking if the face-sink graph satisfies Theorem 1. We write *never / always / depends on Condition (ii)* to summarize whether we can insert  $c_u \rightarrow c_v$ . Assuming that  $c_u$  is not a top corner, the case analysis shows that  $c_u \rightarrow c_v$  may be inserted if and only if Condition (ii) holds.

The edge  $c_u \rightarrow c_v$  splits  $f$  into two faces  $f_t$  and  $f_b$ ; let  $f_b$  be the one incident to  $\pi(c_u, c_v)$ .

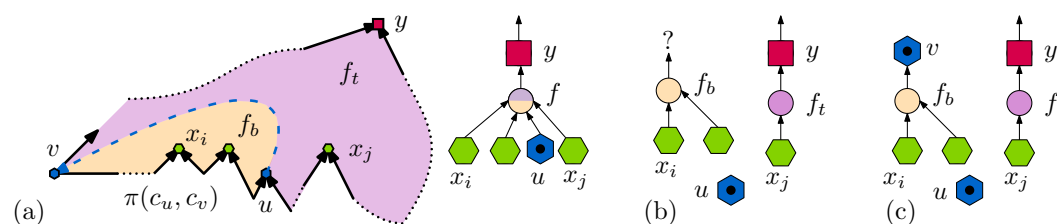
**Case 1:  $c_v$  is incident to two outgoing edges**→**never.** See Figure 6(a+b). If  $f$  is the outer face, assume first that  $c_v$  is the unique reflex source corner of the connected component. Inserting  $c_u \rightarrow c_v$  creates a cycle in  $G$ , violating upward planarity. Else, after the insertion,  $f_b$  and  $f_t$  root two different trees of the face-sink graph. Since neither of them contain a top corner, this contradicts Theorem 1.

If  $f$  is not the outer face, then  $c_v$  is not the top corner of  $f$ . After inserting  $c_u \rightarrow c_v$ ,  $f_b$  cannot be incident to a top corner:  $f_b$  is by construction not incident to the top corner of  $f$  and the insertion in this case creates no additional sink corners. Thus,  $f_b$  becomes a root of a tree in the face-sink graph and, by Theorem 1, it must be the outer face, which is impossible.

**Case 2:  $c_v$  is incident to one outgoing and one incoming edge**→**depends on Condition (ii).** Corner  $c_v$  is split into two corners  $c_t$  and  $c_b$  incident to  $f_t$  and  $f_b$ , respectively.

Consider first the case in which the path  $\pi(c_u, c_v)$  ends with the edge directed to  $v$  (Condition (ii)). After the insertion, all children of  $f$  in the face-sink graph become children of either  $f_t$  or  $f_b$ , with possibly the exception of  $u$ . If  $u$  was a child of  $f$ , then  $c_u$  was a sink corner (not top). Thus,  $u$  becomes a critical vertex. Corner  $c_b$  is a sink corner, and we set its Boolean to convex. Thus,  $c_b$  is the (unique) top corner of  $f_b$ . Face  $f_t$  is either the outer face and incident to the reflex source corner, or it is incident to the top corner of  $f$ . Thus, any subtree rooted at  $f_t$  or  $f_b$  is part of a valid tree in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . See Figure 6(a+c).





■ **Figure 6** (a)  $f$  is split into  $f_b$  and  $f_t$  (orange and purple). We show the original tree in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ . (b) The face-sink graph gets to a graph where the face  $f_b$  becomes parent-less, violating the conditions of Theorem 1. (c) The new face  $f_b$  receives the critical vertex  $v$  as its parent.

Now assume that the path  $\pi(c_u, c_v)$  ends with the edge directed away from  $v$ . In this case  $c_b$  is not a sink corner. Thus,  $f_b$  is not incident to any top corner and also not incident to the reflex source corner of the connected component. This means that, in the face-sink graph,  $f_b$  can neither be the outer face nor have out-degree 1. Thus, it does not satisfy Theorem 1.

**Case 3:  $c_v$  is incident to two incoming edges**→always. First, assume that  $c_v$  is the top corner of  $f$ . By inserting  $c_u \rightarrow c_v$ , we split  $c_v$  into two top corners incident to both  $f_b$  and  $f_t$ . Thus, in the face-sink graph,  $f$  gets replaced by  $f_b$  and  $f_t$  which partition its children and  $\mathcal{F}(\vec{\mathcal{E}}(G))$  is thus still valid. Assume otherwise that  $c_v$  is incident to a sink corner that is not a top corner (i.e., in the face-sink graph, there is a directed edge from  $c_v$  to  $f$ ). The insertion  $c_u \rightarrow c_v$  splits  $c_v$  into two corners  $c_b$  and  $c_t$  incident to  $f_b$  and  $f_t$ , respectively. Both corners are sink corners, and we set  $c_b$  to have a convex angle and  $c_t$  to have a reflex angle. In the face-sink graph, this implies that  $f$  is replaced by the path  $f_b \rightarrow v \rightarrow f_t$  (where  $f_b$  and  $f_t$  partition the children of  $f$ ) and  $\mathcal{F}(\vec{\mathcal{E}}(G))$  still satisfies the conditions of Theorem 1.

**Case 4:  $c_v$  is incident to only one edge, and it is an incoming edge**→always. This case is identical to Case 3. Thus, in  $\mathcal{F}(\vec{\mathcal{E}}(G))$ ,  $f$  is replaced by the path  $f_b \rightarrow v \rightarrow f_t$  (and  $f_b$  and  $f_t$  partition the children of  $f$ ) and  $\mathcal{F}(\vec{\mathcal{E}}(G))$  still satisfies the conditions of Theorem 1.

**Case 5:  $c_v$  is incident to only one edge, and it is an outgoing edge**→never. Here  $v$  is the unique source of  $G$ . Inserting  $c_u \rightarrow c_v$  creates a directed cycle, violating upward planarity. ◀

Lemma 5 enables uplinkability queries. Let  $k$  be the number of output faces.

► **Lemma 6.** *We can support the  $UpLinkable(u, v)$  query in  $O(\log^2 n + k)$  time.*

**Proof.** We use (a) from our data structure to test if  $u$  and  $v$  share a connected component in  $G$ . If not, we use Observation 3 to test whether  $(u, v)$  are uplinkable in  $O(1)$  time (testing if  $v$  is the unique source,  $c_v$  is the reflex source corner, and checking if  $c_u$  is not a top corner). In the remainder, we assume that  $u$  and  $v$  share a component.

By the definition of  $Linkable(u, v)$  in [15], we obtain in  $O(\log^2 n)$  time all corner pairs  $S = \{(c_u, c_v)\}$  across which  $u$  and  $v$  are linkable, sorted around  $u$  and  $v$ . For all corner pairs  $(c_u, c_v) \in S$  where  $(c_u, c_v)$  share a face  $f$ , by Lemma 5,  $u \rightarrow v$  may be inserted into  $\vec{\mathcal{E}}(G)$  if and only if Conditions (i)+(ii) hold. For any given  $(c_u, c_v)$  sharing a face  $f$ , we can test these conditions in  $O(\log n)$  time. Indeed, we may obtain any path  $\pi(c_u, c_v)$  as  $O(\log n)$  subtrees in  $T_f$  and check if any root of the subtree lies on the marked path to the top corner (or the reflex source corner). Note that the top corners  $c'_u$  incident to  $u$  (that violate Condition (i)) are contiguous in the cyclical ordering around  $u$ . So, the set  $S_1 \subseteq S$  of corner pairs that fulfil

Condition (i) is a contiguous subset of  $S$ . The corners  $c'_v$  incident to  $v$  that are incident to at least one edge incoming to  $v$  are contiguous in the cyclical ordering around  $v$ . Hence, the set  $S_2 \subseteq S$  of corner pairs that fulfil Condition (ii) must also be a contiguous subset of  $S$ .

We may obtain the set  $S^* = S_1 \cap S_2$  in  $O(\log n)$  time by performing binary search over  $S$  (using the above  $O(1)$  time testing algorithm at each step). By Holm and Rotenberg [15], we can report the first  $k$  elements in this output in  $O(k)$  additional time. Thus, if  $v \not\prec u$  then we may report the first  $k$  corners  $(c_u, c_v)$  where  $c_u \rightarrow c_v$  is uplinkable in  $O(\log^2 n + k)$  time. ◀

### Supporting more queries

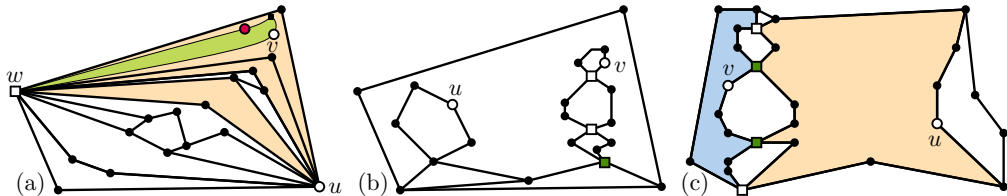
The proof for the remaining queries can be found in the full version. Here, we highlight the main approach and challenges for answering these queries (see Figure 7):

**Slide-UpLinkable( $u, v$ ).** We find (an implicit representation of) all slides  $F_s = (f, w, v)$  around  $w$  (where  $u$  and  $v$  are in different components of  $G \setminus \{w\}$ ) after which  $u$  and  $v$  share a face, as a set  $\Omega_v$  that is cyclically ordered around  $w$  in  $O(\log^2 n)$  time. We identify the set  $\Omega'_v \subseteq \Omega_v$  of slides that do not violate upward planarity. Specifically, we prove that  $\Omega'_v$  is a contiguous subset. We then identify the subset  $\Omega_v^* \subseteq \Omega'_v$  of slides which are part of our output. We show that the faces in  $\Omega'_v$  meeting all conditions of Lemma 5 are a contiguous subset, and we output  $\Omega_v^*$  accordingly. The procedure for slides  $F'_s = (f, w', u)$  is similar.

**Twist-UpLinkable( $u, v$ ).** By using a subroutine of Twist-Linkable( $u, v$ ), we identify an implicit representation of the set  $\Lambda_v$  of twists  $F_t = (w, v)$  in  $O(\log^2 n)$  time. We show that  $u \rightarrow v$  are either uplinkable after all of  $\Lambda_v$  or none of  $\Lambda_v$ . Using Lemma 6 we test whether after an arbitrary twist in  $\Lambda_v$   $u$  and  $v$  are uplinkable. We do the same for twists  $F'_t = (w', u)$ .

**Separation-UpLinkable( $u, v$ ).** By using a variant of Separation-Linkable( $u, v$ ) we consider all separation flips  $F = (f, g, x, y, v)$  where  $u$  is incident to the face  $f$ , and  $v$  to  $g$ . Of these flips, we obtain the unique flip  $F^* = (f, g, x^*, y^*, v)$  where  $x^*$  and  $y^*$  are closest to  $v$ , and the subgraph containing  $v$  is minimal with respect to inclusion. We prove in the full version that if performing  $F^*$  violates upward planarity, then any such  $F$  must violate upward planarity. We show the same statement holds for uplinkability of  $u \rightarrow v$ . We perform  $F^*$ , and use Lemma 5 to test whether we may now insert  $u \rightarrow v$ , thus deciding the question.

**One-Flip-UpLinkable( $u, v$ ).** We show that if there exists faces  $f$  and  $g$ , and any sequence of flips, slides, or twists (as above) involving only  $f$  and/or  $g$  after which  $u \rightarrow v$  may be inserted, then this sequence has constant size. If any such sequence exists, we output it in  $O(\log^2 n)$  time. Upward planarity is more complicated than planarity in this regard, as strictly more than one flip may be necessary when the vertices already share a face.



■ **Figure 7** (a) For articulation slides, the subset  $\Omega_v^*$  where, afterwards,  $u \rightarrow v$  may be inserted is shown in yellow. (b) There exists a twist  $F_t = (w, v)$  for every square vertex  $w$ . We only need to consider the last such  $w$  (green). (c) There exists a separation flip  $F$  for each pair of squares. We show that we only need to consider the green pair  $(x^*, y^*)$ .

## References

- 1 Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Transactions on Algorithms*, 11(4):32:1–32:42, 2015. doi:10.1145/2629341.
- 2 Giuseppe Di Battista and Roberto Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61:175–198, 1988. doi:10.1016/0304-3975(88)90123-5.
- 3 Giuseppe Di Battista and Roberto Tamassia. On-line graph algorithms with SPQR-trees. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 443 of *LNCS*, pages 598–611. Springer, 1990. doi:10.1007/BFb0032061.
- 4 Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996. doi:10.1137/S0097539794280736.
- 5 Paola Bertolazzi, Giuseppe Di Battista, Giuseppe Liotta, and Carlo Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 12(6):476–497, 1994. doi:10.1007/BF01188716.
- 6 Paola Bertolazzi, Giuseppe Di Battista, Carlo Mannino, and Roberto Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM Journal on Computing*, 27(1):132–169, 1998. doi:10.1137/S0097539794279626.
- 7 Guido Brückner, Markus Himmel, and Ignaz Rutter. An SPQR-tree-like embedding representation for upward planarity. In *Proc. 27th International Symposium on Graph Drawing and Network Visualization (GD)*, pages 517–531. Springer, 2019. doi:10.1007/978-3-030-35802-0\_39.
- 8 Steven Chaplick, Emilio Di Giacomo, Fabrizio Frati, Robert Ganian, Chrysanthi N. Raftopoulou, and Kirill Simonov. Parameterized algorithms for upward planarity. In *Proc. 38th International Symposium on Computational Geometry (SoCG)*, volume 224 of *LIPICs*, pages 26:1–26:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.SoCG.2022.26.
- 9 Walter Didimo, Francesco Giordano, and Giuseppe Liotta. Upward spirality and upward planarity testing. *SIAM Journal on Discrete Mathematics*, 23(4):1842–1899, 2009. doi:10.1137/070696854.
- 10 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 599–608. ACM/SIAM, 2003. URL: <https://dl.acm.org/doi/10.5555/644108.644208>.
- 11 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification. I. Planary testing and minimum spanning trees. *Journal of Computer and System Sciences*, 52(1):3–27, 1996. doi:10.1006/jcss.1996.0002.
- 12 Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. Fully dynamic planarity testing. In *Proc. 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 495–506. ACM, 1992. doi:10.1145/129712.129761.
- 13 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.
- 14 Monika Rauch Henzinger and Johannes A. La Poutré. Certificates and fast algorithms for biconnectivity in fully-dynamic graphs. In *Proc. 3rd Annual European Symposium on Algorithms (ESA)*, volume 979 of *LNCS*, pages 171–184. Springer, 1995. doi:10.1007/3-540-60313-1\_142.
- 15 Jacob Holm and Eva Rotenberg. Dynamic planar embeddings of dynamic graphs. *Theory of Computing Systems*, 61(4):1054–1083, 2017. doi:10.1007/s00224-017-9768-7.
- 16 Jacob Holm and Eva Rotenberg. Fully-dynamic planarity testing in polylogarithmic time. In *Proc. 52nd Annual ACM-SIGACT Symposium on Theory of Computing (STOC)*, pages 167–180. ACM, 2020. doi:10.1145/3357713.3384249.

- 17 Jacob Holm and Eva Rotenberg. Worst-case polylog incremental SPQR-trees: Embeddings, planarity, and triconnectivity. In *Proc. 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2378–2397. SIAM, 2020. doi:10.1137/1.9781611975994.146.
- 18 John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974. doi:10.1145/321850.321852.
- 19 Michael D. Hutton and Anna Lubiw. Upward planar drawing of single-source acyclic digraphs. *SIAM Journal on Computing*, 25(2):291–311, 1996. doi:10.1137/S0097539792235906.
- 20 Paul Jungeblut, Laura Merker, and Torsten Ueckerdt. A sublinear bound on the page number of upward planar graphs. In *Proc. 2022 ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 963–978. SIAM, 2022. doi:10.1137/1.9781611977073.42.
- 21 David Kelly. Fundamentals of planar ordered sets. *Discrete Mathematics*, 63(2-3):197–216, 1987. doi:10.1016/0012-365X(87)90008-2.
- 22 Giordano Da Lozzo, Giuseppe Di Battista, and Fabrizio Frati. Extending upward planar graph drawings. *Computational Geometry*, 91:101668, 2020. doi:10.1016/j.comgeo.2020.101668.
- 23 Giordano Da Lozzo, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Vincenzo Roselli. Upward planar morphs. *Algorithmica*, 82(10):2985–3017, 2020. doi:10.1007/s00453-020-00714-6.
- 24 Achilleas Papakostas. Upward planarity testing of outerplanar dags. In *Proc. DIMACS International Workshop on Graph Drawing (GD)*, volume 894 of *LNCS*, pages 298–306. Springer, 1994. doi:10.1007/3-540-58950-3\_385.
- 25 Mihai Patrascu. Lower bounds for dynamic connectivity. In *Encyclopedia of Algorithms - 2008 Edition*. Springer, 2008. doi:10.1007/978-0-387-30162-4\_214.
- 26 Mihai Patrascu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 546–553. ACM, 2004. doi:10.1145/1007352.1007435.
- 27 C. R. Platt. Planar lattices and planar graphs. *Journal of Combinatorial Theory, Series B*, 21(1):30–39, 1976. doi:10.1016/0095-8956(76)90024-1.
- 28 Johannes A. La Poutré. Alpha-algorithms for incremental planarity testing (preliminary version). In *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC)*, pages 706–715. ACM, 1994. doi:10.1145/195058.195439.
- 29 Aimal Rextin and Patrick Healy. Dynamic upward planarity testing of single source embedded digraphs. *The Computer Journal*, 60(1):45–59, 2017. doi:10.1093/comjnl/bxw064.
- 30 Jeffery R. Westbrook. Fast incremental planarity testing. In *Proc. 19th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 623 of *LNCS*, pages 342–353. Springer, 1992. doi:10.1007/3-540-55719-9\_86.