


On Dynamic $\alpha + 1$ Arboricity Decomposition and Out-Orientation

Aleksander B. G. Christiansen ✉

Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Denmark

Jacob Holm ✉

Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Eva Rotenberg ✉ 

Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Denmark

Carsten Thomassen ✉

Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Denmark

Abstract

A graph has arboricity α if its edges can be partitioned into α forests. The dynamic arboricity decomposition problem is to update a partitioning of the graph's edges into forests, as a graph undergoes insertions and deletions of edges. We present an algorithm for maintaining partitioning into $\alpha + 1$ forests, provided the arboricity of the dynamic graph never exceeds α . Our algorithm has an update time of $\tilde{O}(n^{3/4})$ when α is at most polylogarithmic in n .

Similarly, the dynamic bounded out-orientation problem is to orient the edges of the graph such that the out-degree of each vertex is at all times bounded. For this problem, we give an algorithm that orients the edges such that the out-degree is at all times bounded by $\alpha + 1$, with an update time of $\tilde{O}(n^{5/7})$, when α is at most polylogarithmic in n . Here, the choice of $\alpha + 1$ should be viewed in the light of the well-known lower bound by Brodal and Fagerberg which establishes that, for general graphs, maintaining only α out-edges would require linear update time.

However, the lower bound by Brodal and Fagerberg is non-planar. In this paper, we give a lower bound showing that even for planar graphs, linear update time is needed in order to maintain an explicit three-out-orientation. For planar graphs, we show that the dynamic four forest decomposition and four-out-orientations, can be updated in $\tilde{O}(n^{1/2})$ time.

2012 ACM Subject Classification Theory of computation \rightarrow Dynamic graph algorithms

Keywords and phrases Dynamic graphs, bounded arboricity, data structures

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.34

Funding *Aleksander B. G. Christiansen*: Partially supported by the VILLUM Foundation grant 37507 “Efficient Recomputations for Changeful Problems”.

Jacob Holm: Partially supported by the VILLUM Foundation grant 16582, “BARC”

Eva Rotenberg: Partially supported by Independent Research Fund Denmark grants 2020-2023 (9131-00044B) “Dynamic Network Analysis” and 2018-2023 (8021-00249B) “AlgoGraph”, and the VILLUM Foundation grant 37507 “Efficient Recomputations for Changeful Problems”.

Carsten Thomassen: Partially supported by Independent Research Fund Denmark grant 2018-2023 (8021-00249B) “AlgoGraph”.

Acknowledgements We thank Irene Parada for helpful discussions.

1 Introduction

Dynamic graphs have been the subject of much study. Here one is typically interested in maintaining some property of or information about the graph, as edges of the graph are inserted and deleted. Sometimes one studies more restricted classes of graphs with more structure



© Aleksander B. G. Christiansen, Jacob Holm, Eva Rotenberg, and Carsten Thomassen;
licensed under Creative Commons License CC-BY 4.0

47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).

Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 34; pp. 34:1–34:15

Leibniz International Proceedings in Informatics



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

in the hopes of improving algorithmic performance. The existence of efficient algorithms for testing planarity of a fully-dynamic graph [17, 24] motivates efforts to dynamically maintain well-known properties of planar graphs, such as e.g. bounded out-degree orientations and colourings.

The *arboricity* of a graph G is the smallest number α such that α forests can cover the edges of G . Planar graphs have arboricity at most 3. If a forest is rooted arbitrarily, and all of the edges in the forest are oriented towards the roots, then every vertex has out-degree 1. In particular, this implies that the edges of a planar graph resp. graphs of arboricity α can be oriented such that no vertex has out-degree more than 3 resp. α . We call an orientation of a graph such that no vertex has more than k out-edges a *k-bounded out-orientation*. Since there exists planar graphs on n vertices with more than $2n$ edges, it follows that there exists planar graphs for which a 3-bounded out-orientation is the lowest out-orientation possible.

In light of the above, we ask the question: can one efficiently maintain a 3-bounded out-orientation of a dynamic planar graph? Here the aim is to be more efficient than the fastest static algorithm – running in linear time – as one could construct a dynamic algorithm by statically recomputing a new solution after every update.

It turns out that it is not possible to improve upon this; at least not if we want to maintain an *explicit* orientation. Here, one has to store the orientation of every edge in memory opposed to an *implicit* orientation, where one is allowed to compute the orientation of an edge at query-time. More precisely we show that any algorithm maintaining an explicit 3-bounded out-orientation of a dynamic planar graph must spend $\Omega(n)$ update time, even amortised. For the broader class of graphs with bounded arboricity, Brodal & Fagerberg [9] showed that any algorithm maintaining an explicit α -bounded out-orientation must spend $\Omega(n)$ update time, even amortised. However for $\alpha = 3$ their example is far from planar. Our lower bound shows that the same bounds indeed hold for planar graphs.

In light of this negative result, it is natural to ask if one can do better if a little slack on the number of out-edges is allowed. We show that this is indeed the case, as we provide an algorithm maintaining a 4-bounded out-orientation of a dynamic planar graph with $\tilde{O}(\sqrt{n})$ amortised update time. In fact, this generalises to maintaining an $(\alpha + 1)$ -bounded out-orientation of a dynamic graph whose arboricity never exceeds α through-out the entire update sequence. Here the algorithm has an amortised update-time of $\tilde{O}(n^{5/7})$. Here \tilde{O} hides α and $\log n$ factors, so one should think of $\alpha = O(1)$.

An *arboricity decomposition* of a graph is a decomposition of the graph's edges into forests. We show how to dynamically maintain a 4-arboricity decomposition of a dynamic planar graph with $\tilde{O}(\sqrt{n})$ amortised update time. This also generalises to graphs of bounded degree α : here we present an algorithm maintaining an $(\alpha + 1)$ arboricity decomposition with an amortised update-time of $\tilde{O}(n^{3/4})$.

The presented algorithms all follow the same idea: we show how to update the out-orientation or arboricity decomposition in such a way that 1) every update only uses very little of the extra slack provided by having one more out-edge or one more forest available and 2) the update time scales with the amount of extra slack used. Combining these two properties allow us to truncate the update algorithm if it runs for too long and instead statically recompute an optimal solution.

1.1 Results

We consider dynamic graphs on n vertices undergoing a sequence of updates such that the arboricity of the graph at all times is bounded by α . We refer to such a sequence of updates as an α preserving sequence of updates.

We show the following theorem for maintaining an $\alpha + 1$ out-orientation.

► **Theorem 1.** *Given an initially empty dynamic graph undergoing an arboricity α preserving sequence of updates and a static black box algorithm that computes an α -bounded out-degree orientation of a graph with n vertices and arboricity α in time $S(\alpha, n)$, the algorithm will maintain an $(\alpha + 1)$ -bounded out-degree orientation with an amortised insertion time of $O(\sqrt{\alpha} \cdot S(\alpha, n))$, and a worst-case deletion time of $O(1)$.*

Specifying a black box algorithm gives the following corollary:

► **Corollary 2.** *Given an initially empty dynamic planar graph undergoing edge insertions and deletions, there is an algorithm that maintains a 4-bounded out-degree orientation with an amortised insertion time of $O(\sqrt{n})$, and a worst-case deletion time of $O(\log n)$.*

Proof. Chrobak & Eppstein gave a linear-time algorithm for computing a 3-bounded out-degree orientation in planar graphs [12]. Now applying Theorem 1 yields the result. ◀

For bounded arboricity graphs, applying the fastest static algorithm [31] gives an amortised update time of $\tilde{O}(n^{5/7})$. We show the following theorem for maintaining an $\alpha + 1$ arboricity decomposition:

► **Theorem 3.** *Given an initially empty dynamic graph undergoing an arboricity α preserving sequence of updates and a static black box algorithm that computes an α arboricity decomposition of a graph with n vertices and arboricity α in time $S(\alpha, n)$, the algorithm will maintain an $(\alpha + 1)$ arboricity decomposition with an amortised insertion time of $O(\alpha\sqrt{S(\alpha, n)} \log n)$, and a worst-case deletion time of $O(\log n)$.*

Specifying black box algorithms gives the following corollaries:

► **Corollary 4.** *Given an initially empty dynamic planar graph undergoing edge insertions and deletions, then there exists an algorithm maintaining a 4-arboricity decomposition with an amortised insertion time of $O(\sqrt{n} \log n)$, and a worst-case deletion cost of $O(\log n)$.*

Proof. Chrobak & Eppstein also showed how to compute a 3-forest partition in linear time in planar graphs [12]. ◀

For bounded arboricity graphs, applying the fastest static algorithm [18, 19] gives an amortised update time of $\tilde{O}(n^{3/4})$. Finally, we show a lower bound for maintaining explicit 3-bounded out-orientations in dynamic planar graphs:

► **Theorem 5.** *Let A be an algorithm explicitly maintaining a 3-bounded out-degree orientation of an n -vertex planar graph under insertion and deletion of edges. Then there exists a sequence of updates taking $\Omega(n)$ amortised time per update.*

Note that some results rely on ideas similar to those that appeared in the master's thesis by A. B. G. Christiansen [10].

1.2 Related Work

Dynamic Planar Graphs. Dynamic planar graphs have been studied both in the incremental (edge-insertion only) [13, 25, 35, 38] and fully-dynamic (insertion/deletion) setting [16, 17, 20, 24, 26].

34:4 On Dynamic $\alpha + 1$ Arboricity Decomposition and Out-Orientation

■ **Table 1** Overview of dynamic algorithms for maintaining out-orientations. This table is inspired by a similar table in [11].

Reference	Out-degree	Update time	α preserving seq.
Brodal & Fagerberg [9]	$2(\alpha + 1)$	$\tilde{O}(\log n)$ am.	yes
Kopelowitz et al. [27]	$\beta\alpha + \log_\beta n$	$\tilde{O}(\beta^2 + \beta \log n)$	no
He et al. [22]	$O(\alpha\sqrt{\log n})$	$O(\sqrt{\log n})$ am.	yes
Berglin & Brodal [5]	$O(\alpha + \log n)$	$O(\log n)$	no
Henzinger et al. [23]	40α	$O(\log^2 n)$ am.	no
Kowalik [29]	$O(\alpha \log n)$	$O(1)$ am.	yes
Christiansen & Rotenberg [11]	$\alpha + 2$	$\tilde{O}(\log^3(n))$	yes
New (Thm. 1)	$\alpha + 1$	$\tilde{O}(n^{5/7})$	yes

Bounded Out-Orientations. For the more general class of graphs with bounded arboricity, the out-orientation problem is well studied – both from a dynamic view-point [9, 27, 37, 29, 5] and a static view-point [18, 34, 6, 1]. For dynamic arboricity-bounded graphs, Brodal & Fagerberg gave an algorithm maintaining a $2(\alpha + 1)$ -bounded out-orientation with amortised $O(\alpha + \log n)$ update time. There also exists an algorithm [11] for maintaining an $(\alpha + 2)$ -bounded out-orientation with worst-case update time in $O(\text{poly}(\log n, \alpha))$. See Table 1 for an overview.

From a static view-point, the fastest algorithms have running time $\tilde{O}(m^{10/7})$ [31] and $\tilde{O}(m\sqrt{n})$ [30]. Furthermore, Kowalik [28] also gave an algorithm computing a $(\alpha + 1)$ out-orientation in $\tilde{O}(m)$ time. Specialising to the planar case, where the graphs are assumed to be planar at all times, the lowest out-degree one can get in the dynamic setting while still achieving sublinear update time becomes 5 with update time in $O(\text{poly}(\log n))$. In the static case, Chrobak & Eppstein showed how to compute a 3 out-orientation in linear time [12].

Arboricity Decompositions. There has been a lot of work dedicated to computing an arboricity decomposition [18, 19, 14, 34]. The state-of-the-art for computing exact arboricity decompositions run in $\tilde{O}(m^{3/2})$ time [18, 19]. There also exists approximation algorithms. There is a 2-approximation algorithm [3, 15] as well as an algorithm for computing an $\alpha + 2$ arboricity decomposition in near-linear time [7]. From the dynamic side, Bannerjee et al. [4] give a dynamic algorithm for maintaining the current arboricity. The algorithm has a near-linear update time. They also provide a lower bound of $\Omega(\log n)$ for dynamically maintaining arboricity. Henzinger et al. [23] provide a dynamic algorithm for maintaining a $2\alpha'$ arboricity decomposition, given access to any black box dynamic α' out-degree orientation algorithm. Combining this technique with the results from Table 1 yields different trade-offs. Finally there also exists an algorithm maintaining an $(\alpha + 2)$ arboricity decomposition with $O(\text{poly}(\log(n), \alpha))$ update-time [11].

Specialising to the planar case, the last of the algorithms above yields a sublinear update time dynamic algorithm for computing a 5 arboricity decomposition. In the static case, Chrobak & Eppstein also showed how to compute a 3-forest partition in linear time [12].

1.3 Paper Outline

In Section 2 we recall preliminaries. Section 3 is dedicated to Theorem 1. In Section 4 we show Theorem 3. Finally in Section 5, we prove Theorem 5.

2 Preliminaries & Notation

We follow standard graph-terminology. We say that a graph $G = (V, E)$ is k -degenerate if every subgraph of G has a vertex of degree at most k . If the edges of G receives an orientation, we let the *out-edges* of a vertex v be the edges oriented away from v . The *out-degree* of v is then the number of out-edges incident to v .

Nash-Williams showed the following density formula for graphs of arboricity α :

► **Theorem 6** ([33, 32]). *Let G be a graph with no loops. Then*

$$\alpha(G) = \left\lceil \max_{J \subset G, |V(J)| \geq 2} \frac{|E(J)|}{|V(J)| - 1} \right\rceil$$

A plane graph is a planar graph together with an embedding into the plane such that no edges cross. For plane graphs we have the usual notion of faces. A *triangulation* is a maximal planar subgraph (wrt. the edges). If the triangulation is equipped with an embedding, we call it a *plane triangulation*. For planar graphs, we have the following well-known theorem:

► **Theorem 7** (Euler's Theorem). *If G is a connected plane graph with n vertices, m edges and f faces, then $n - m + f = 2$. In particular, if G is a planar triangulation, then $m = 3n - 6$.*

3 Low Out-orientations

In this section, we present a dynamic algorithm that maintains an $(\alpha + 1)$ -bounded out-degree orientation. Suppose we are given an initially empty dynamic graph G on n vertices undergoing an arboricity α preserving sequence of updates. Then the algorithm has an update time of $\tilde{O}(\sqrt{S(\alpha, n)})$ provided that it is given access to a black box static algorithm for computing an α -bounded out-degree orientation in $O(S(\alpha, n))$ time.

The idea behind the algorithm is the following: suppose we are given a graph G , an edge $e = uv \in G$ and an $\alpha + 1$ orientation of $G - e$. Then we can extend the orientation of $G - e$ to an $\alpha + 1$ orientation of G in the following way. Find a minimal oriented path P beginning at u that ends at a vertex w with out-degree at most α . We will refer to such a path as an *augmenting path*. After finding P , we reorient all edges along the path. Finally, we make uv an out-edge of u . This yields an $\alpha + 1$ orientation of G since the only vertex to have its out-degree increased is w , and w was specifically chosen to have out-degree $< \alpha + 1$.

In order to find P , we look for w by doing a breadth-first-search (BFS) beginning from u . In Lemma 8, we show that such a BFS always succeeds and that it takes time proportional to the number of vertices visited. Finally, we show that if the BFS visits too many vertices, we have witnessed enough updates to be able to truncate the search and recompute an α orientation.

The Algorithm. Whenever an edge is deleted, we can simply delete it in the graph, so we focus on insertions. Suppose $e = uv$ is inserted. Either u has out-degree $< \alpha + 1$ and $P = u$ is an augmenting path or u has out-degree $\alpha + 1$ in which case, we push u 's neighbours to a queue Q and mark them as visited. Now we can recursively visit vertices in Q by pushing the un-visited neighbours of vertices with degree $\alpha + 1$ to Q , and terminating with an augmenting path if we locate a vertex with out-degree $< \alpha + 1$. In the end of the update algorithm, we mark all visited vertices as un-visited. The search for P has the following properties:

► **Lemma 8.** *Suppose that $G - uv$ is given an $(\alpha + 1)$ -bounded out-degree orientation, and we begin a search from u . Then after visiting t vertices the following holds:*

1. *If no vertex with out-degree $< \alpha + 1$ has been visited, then Q is not empty.*
2. *The algorithm has spent $O(\alpha \cdot t)$ -time.*

Proof. We begin by showing that 1) holds. Suppose for contradiction that the algorithm has visited all vertices pushed to Q without finding a vertex with out-degree $< \alpha + 1$. Let J be the set of visited vertices. Since the algorithm pushes all new out-neighbours of a vertex v to Q , when visiting v , it must be the case that all out-edges of vertices in J go to other vertices in J . Since the algorithm did not find a vertex with out-degree $< \alpha + 1$, all vertices of J have out-degree $\alpha + 1$, and as a result, there must be at least $(\alpha + 1)|J|$ edges between vertices in J . But then, $\frac{|E(J)|}{|J|-1} = \frac{(\alpha+1)|J|}{|J|-1} > \alpha + 1 > \alpha$, contradicting Theorem 6.

Now, we show that 2) also holds. After visiting t vertices, the algorithm has traversed every out-edge out of the visited vertices when pushing new out-neighbours to Q . The time needed to perform these steps is $O(1)$ per visited vertex and $O(1)$ per out-edge leaving a visited vertex. Since every visited vertex has out-degree $\alpha + 1$ by assumption, the total time spent in the search is $O(\alpha \cdot t)$. ◀

► **Remark 9.** Lemma 2 in [9] implies that P has length $O(\alpha \log(n))$.

Now we make the following observation:

► **Observation 10.** *Inserting an edge e into G increases the number of vertices with out-degree $\alpha + 1$ by at most one.*

Proof. The only vertex that has its out-degree increased during an insertion is the final vertex of the augmenting path. ◀

In particular this means that if we visit t vertices when searching for P , then we must have witnessed at least t insertions, since the last time we statically re-computed an α -bounded out-orientation. As such, we arrive at Theorem 1 (restated below for convenience):

► **Theorem 11** (Theorem 1). *Given an initially empty dynamic graph undergoing an arboricity α preserving sequence of updates and a static black box algorithm that computes an α -bounded out-degree orientation of a graph with n vertices and arboricity α in time $S(\alpha, n)$, the algorithm will maintain an $(\alpha + 1)$ -bounded out-degree orientation with an amortised insertion time of $O(\sqrt{\alpha \cdot S(\alpha, n)})$, and a worst-case deletion time of $O(1)$.*

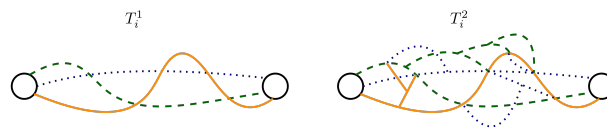
Proof. The algorithm clearly maintains an $(\alpha + 1)$ -bounded out-degree orientation. All that remains is to analyse the amortised cost of running the black box algorithm. To this end, say a vertex is *bad*, if it has out-degree $\alpha + 1$. Suppose that we have witnessed i insertions since the last time that the static black-box algorithm was run, and suppose furthermore that inserting e_{i+1} causes the insertion search to visit $\sqrt{S(\alpha, n)/\alpha}$ bad vertices. Then Observation 10 implies that $i \geq \sqrt{S(\alpha, n)/\alpha}$, and so setting aside $O(\sqrt{\alpha \cdot S(\alpha, n)})$ credit for each insertion, ensures that at least $O(S(\alpha, n))$ credit is stored, when the black box algorithm is run. Furthermore, by Lemma 8 each truncated search spends no more than $O(\alpha \cdot \sqrt{S(\alpha, n)/\alpha})$ -time, yielding a total amortised insertion time in $O(\sqrt{\alpha \cdot S(\alpha, n)})$, as claimed. To delete an edge, we may remove it from the graph in constant time (assuming that we are given a pointer to the edge as part of the query). ◀

4 Low Arboricity Decompositions

In this section, we present a dynamic algorithm that maintains an $(\alpha + 1)$ -arboricity decomposition. The setup is the same as in the previous section: we assume that we are given an initially empty dynamic graph G on n vertices undergoing an arboricity α preserving sequence of updates. The algorithm has an update time of $\tilde{O}(\sqrt{S(\alpha, n)})$ provided that it is given access to a black box static algorithm for computing an α -arboricity decomposition in $O(S(\alpha, n))$ time.

The idea behind this algorithm is the same as in the previous section. We give an algorithm based on a search procedure with slow update time, and arrive at the final algorithm via truncation of the search. The main difficulty of turning the algorithm from the previous section into an algorithm maintaining an arboricity decomposition is that it is not clear how to direct a search to obtain a sequence of forest alterations that allows one to accommodate the insertion of an edge.

We will now present how to accommodate the edge insertion using techniques that are inspired by [18] and [21]. To this end, suppose we are given a graph G , an edge $e = uv \in G$ and an $\alpha + 1$ arboricity decomposition of $G - e$. We are then interested in extending this to an $\alpha + 1$ arboricity decomposition of G . If e cannot be added to forest F_i , it is because u and v are connected by a path T_i^1 in F_i . If this is the case for all i , then we must move one of the edges in T_j^1 for some j to be able to put e into F_j . If this is not possible for any j , it is because all vertices on T_j^1 sit in the same tree in all forests. So for all j we can let T_j^2 be the smallest tree in F_j spanning the vertices $\bigcup_{i=1}^{\alpha+1} V(T_i^1)$. Continuing like this yields trees T_i^r (see Figure 1). We cannot continue this construction indefinitely since we cannot partition



■ **Figure 1** An illustration of the first two layers of the forests. The forests are represented by different colours.

G 's edges into $\alpha + 1$ spanning trees without contradicting the fact that G has arboricity α . Hence, there must exist a largest k for which T_j^k has no movable edges for all choices of j . For an edge $e' \neq e$, we let the *rank* $r(e')$ of e' be the smallest r such that $e' \in T_i^r$ for some i that we denote $t(e')$. We set $r(e) = 0$. Furthermore, we define $T^r = \bigcup_i T_i^r$. Similar to [18, 21], we can now define an *augmenting sequence* of edges that we can move between forests to accommodate an insertion of e :

► **Definition 12.** Suppose G is a graph, $e_0 \in E(G)$ an edge in G , and that we are given an $\alpha + 1$ arboricity decomposition $F_1, \dots, F_{\alpha+1}$ of $G - e_0$. Then an *augmenting sequence of edges* is a sequence of edges e_0, e_1, \dots, e_k satisfying the following conditions:

1. For all $i \geq 0$, the rank of e_i satisfies $r(e_i) \leq i$.
2. For all $i \geq 1$, if $e_{i-1} = xy$, then x and y sit in different trees in $F_{t(e_i)} - e_i$.
3. There exists some $j \neq k$ such that $F_j \cup e_k$ is a forest.

Given such a sequence of edges e_1, \dots, e_k , we can extend the arboricity decomposition of $G - e$ to contain e . Indeed, we move e_k from $F_{t(e_k)}$ to F_j . Now e_1, \dots, e_{k-1} is an augmenting sequence. Eventually, e can be inserted into $F_{t(e_1)}$.

We shall search for an augmenting sequence differently than in [18]. Here, when visiting an edge e' belonging to forest F_i , for $j \neq i$ they en-queue the entire fundamental cycle in $F_j \cup e_i$. However, this cycle could possibly have length $\Omega(n)$. Instead, we visit the paths blocking e' one edge at a time, only en-queueing the visited edge. This ensures that at all times, the time spent searching for an augmenting sequence is proportional to the number of vertices that we have processed. This property is key in ensuring that we can afford to truncate our search and run a static algorithm, if we do not identify an augmenting sequence fast.

The algorithm. Similar to the previous section, we first describe a slow, search-based algorithm, and then we obtain a faster algorithm via truncation. We accommodate edge deletions by simply deleting the edge from the dynamic forest data structure, it resides in. To handle an insertion of an edge $e = uv$, we construct a short augmenting sequence as follows. Beginning with e , we process an edge by trying to insert it into all forests. If this is not possible, u and v must be connected by a path in F_i for all i . For each i , beginning with $i = 1$, we then try to move every edge on this path in F_i . If we fail to do so, it means that there is no augmenting sequence ending with an edge in $T^1 = \bigcup_i T_i^1$. Now we search for an augmenting sequence ending with an edge in T^2 . If we fail to locate such an augmenting sequence, we try to find one ending with an edge in T^3 and so on. To do so for T^r , having already tried all edges in T^{r-1} , we try to move all edges blocking an edge in T^{r-1} i.e. we try to move edges blocking the previously explored edges. When we try to move such an edge, but fail to do so, it is because the endpoints of these edges are connected by blocking paths in all forests. Conceptually we would like to push these blocking paths to a queue as we encounter them, in order to remember them. This is relevant, since we might have to visit them later to try to locate an augmenting sequence. However, these paths might have length $\Omega(n)$, and, similar to the last section, we would like the time spent during a search to depend on the number of visited vertices. Hence, in practice, we traverse the blocking paths one by one. Whenever we visit an edge f on a blocking path in a forest F_j , we try to move it to another forest. If this is not possible, we push f to a queue Q , store that f is blocking e by saving a value $b(f) = e$, and mark both endpoints of f as visited in F_j . Now we can always recover the edges on the paths blocking f one-by-one using standard dynamic forest data-structures. When we are done trying to move edges from the previous set of blocking paths, we recursively try to move the un-processed edges blocking $e' = \text{pop}(Q)$. Note that whenever a blocking path reaches a previously visited edge, we terminate that blocking path and push no further edges of it to Q . The nature of the search ensures that the blocking paths form trees in each forest, and therefore no blocking edge is left unvisited by this early termination. Indeed, one of the endpoints of the blocking path was already in the tree, and so as soon as the path connects to this tree we can be certain that the rest of the edges on the path have been processed. If we encounter an edge g that we can move to a new forest, we extract an augmenting sequence by moving the edge and then recursively moving the edge it was blocking. We have the following claim:

▷ **Claim 13.** Let e and g be as above. We have that $e = b^{r(g)}(g), \dots, b(b(g)), b(g), g$ is an augmenting sequence, where $b^k(g)$ is the edge obtained by following the blocked edge k steps back.

Proof. Observe that we process all edges of rank i before processing any edges of rank $i + 1$. In particular, we only visit edges of rank $i + 1$ when processing edges of rank i . Therefore, for all processed edges h , we have that $r(b(h)) = r(h) - 1$. Hence, $b^{r(g)}(g) = e$, as e is the only edge of rank 0. It is now easy to verify that the conditions of Definition 12 hold. ◁

By storing each vertex in a data structure for maintaining dynamic forests for instance top-trees [2] or link-cut trees [36], we can determine whether a blocking path exists, and if it does, we can traverse it spending $O(\log n)$ time to find the path and $O(1)$ time per visited edge on the path. Thus the search satisfy the following properties:

► **Lemma 14.** *Suppose that we have an $\alpha + 1$ arboricity decomposition of $G - uv$, and we begin a search to extend it to an $\alpha + 1$ arboricity decomposition of G . Then after visiting t vertices the following holds:*

1. *If no moveable edge has been located, then Q is not empty.*
2. *The algorithm has spent $O(\alpha^2 \cdot t \cdot \log n)$ -time.*

Proof.

1. Let E denote the set of edges, visited by the algorithm. Every time we unsuccessfully try to move an edge $e = xy$, we conceptually push all edges on the unique x to y path in F_i to Q for all i . Thus, the blocking paths necessarily form trees in each forest. In particular, this means that if Q is empty, the graph formed by all the visited edges, $J = G[E]$, will be a tree if restricted to F_i for all i . Indeed, suppose it is not true, and that the restriction of J to F_j is not a tree. Then we must have visited some first edge leaving the component, containing u , of the restriction of J to F_j for which we did not explore a blocking path in F_j , contradicting the fact that Q is empty. This means, in particular, that $|E(J)| \geq (\alpha + 1)(|V(J)| - 1)$, and hence J contradicts Theorem 6, as in the proof of Lemma 8.
2. Since the algorithm has visited t vertices, it cannot have visited more than $\alpha \cdot t$ edges. Otherwise, the t visited vertices form a subgraph with a density contradicting Theorem 6. Each time the algorithm visits an edge, it spends only $O(\alpha \log n)$ -time. Indeed, it takes $O(\alpha \log n)$ -time to check if the edge can be moved and $O(1)$ -time to push it to Q . Finally, we can update pointers to the blocked edge with only $O(1)$ overhead per visit. Since the algorithm has visited no more than $\alpha \cdot t$ edges, 2) follows. ◀

Furthermore, we make the following observations:

► **Observation 15.** *If the search phase is unsuccessful after visiting t vertices, then every vertex visited by the algorithm during the search phase must belong to the same tree in every forest F_i . In particular, this means that $|E(F_{\alpha+1})| \geq t - 1$.*

Proof. We prove the observation by contraposition. Suppose that we initially tried to insert uv . If at some point we try to move an edge $e = xy \in F_j$ with endpoints belonging to different trees in F_k , $k \neq j$, then e could be moved to F_k rendering the search phase successful. In particular, this means that if $u \in T \subset F_i$ and there is some other vertex w in a different tree T' in F_i that we also visit, then at some point, we must have tried to move an edge e' with only one endpoint in T . But then e' could have been moved to F_i , contradicting the fact that the search was unsuccessful. ◀

► **Observation 16.** *Augmenting along an augmenting sequence e_0, e_1, \dots, e_k increases the size of the forest that e_k is moved to by one. All other forests remain the same size.*

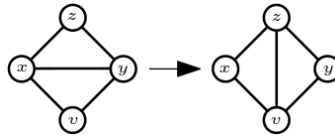
Proof. We prove the statement by induction on k . The induction basis is clear, so we move to the induction step. After moving e_k from $F_{t(e_k)}$ to some forest F_i , the size of F_i is increased by one, the size of $F_{t(e_k)}$ is decreased by one, and all other forests have the same size. Now e_0, e_1, \dots, e_{k-1} is an augmenting sequence, so we can apply induction on it. This sequence increases the size of $F_{t(e_k)}$ by exactly one, and all other forests remain the same size. Hence, we arrive at the statement. ◀

Since a deletion never increases the size of a forest, the two above observations together imply that choosing the truncation $t = \sqrt{\frac{S(\alpha, n)}{\alpha^2}} + 1$ means that we witness at least $\sqrt{\frac{S(\alpha, n)}{\alpha^2}}$ insertions between any two runs of the static black box algorithm. Since we can extend the black box algorithm to maintain the dynamic forests using only $O(\log n)$ overhead per operation, the total time needed for each run is in $O(S(\alpha, n) \log n)$. Combining this with Lemma 14, we find that setting aside $O(\alpha \sqrt{S(\alpha, n)} \log n)$ per insertion yields a total amortised update time in $O(\alpha \sqrt{S(\alpha, n)} \log n)$. Hence:

► **Theorem 17** (Identical to Theorem 3). *Given an initially empty dynamic graph undergoing an arboricity α preserving sequence of updates and a static black box algorithm that computes an α arboricity decomposition of a graph with n vertices and arboricity α in time $S(\alpha, n)$, the algorithm will maintain an $(\alpha + 1)$ arboricity decomposition with an amortised insertion time of $O(\alpha\sqrt{S(\alpha, n)}\log n)$, and a worst-case deletion time of $O(\log n)$.*

5 Dynamic 3-Out Orientations in planar graphs

In this section, we show that any dynamic algorithm maintaining a 3-bounded out-orientation of a dynamic planar graph can be forced to spend $\Omega(n)$ update time. The idea is to first consider a different problem: let G be a plane triangulation. Then G has an outer face xyz . Every vertex incident to the outer face is an *outer* vertex. All vertices that are not outer vertices are *inner* vertices. A 3-orientation of G is then an orientation of all edges incident to inner vertices such that all inner vertices have out-degree 3. Given a plane triangulation, we have the notion of a *diagonal flip*: a diagonal flip is the action of removing an edge xy incident on faces xyz and vxy and replacing it with the edge vz (see Figure 2). Note that such a flip is only possible, if the edge vz does not already exist. The *flip distance* between two undirected graphs is then the minimum number of diagonal flips needed to go from one one graph to the other. We show that any algorithm explicitly maintaining a 3-orientation



■ **Figure 2** A diagonal flip.

of a dynamic plane triangulation under diagonal flips of inner edges, can be made to spend linear update time. This is done by constructing two planar graphs of constant flip distance, but with unique 3-orientations differing on $\Omega(n)$ edges. By slightly generalising and by considering a graph containing multiple copies of this construction, we arrive at the explicit lower bound for maintaining 3-orientations in dynamic planar graphs.

3-Orientations. Brehm [8] showed that a plane triangulation has a unique 3-orientation if and only if it is *stacked* which is equivalent to being 3-degenerate. In fact this can be slightly generalised - and we will use this slight generalisation later on, when dealing with graphs where no embedding is specified. The proof of the generalisation is similar, so we only provide a sketch:

► **Lemma 18.** *Let G be a planar triangulation with a 3-bounded out-degree orientation O . Let x, y, z form a triangle in G , and let H be a component of $G - \{x, y, z\}$, such that the out-degree $d_G^+(v) = 3$ for all $v \in H$. Then the restriction of any 3-bounded out-degree orientation of G to all edges incident to H is unique if and only if $G[V(H) \cup \{x, y, z\}]$ is 3-degenerate.*

Proof (Sketch). A counting argument shows that all edges between H and x, y, z are oriented away from H . Indeed, $G[H \cup \{x, y, z\}]$ is planar and so contains at most $3(|H| + 3) - 6$ edges. 3 of these go between x, y, z , so the remaining $3|H|$ edges must be out-edges of vertices in H .

Next, we show that the restriction of any 3-bounded out-degree orientation O to H is unique iff it is acyclic. Indeed, suppose it is not acyclic. Then it has a directed cycle. Reversing the orientation along this cycle creates a new 3-bounded out-degree orientation

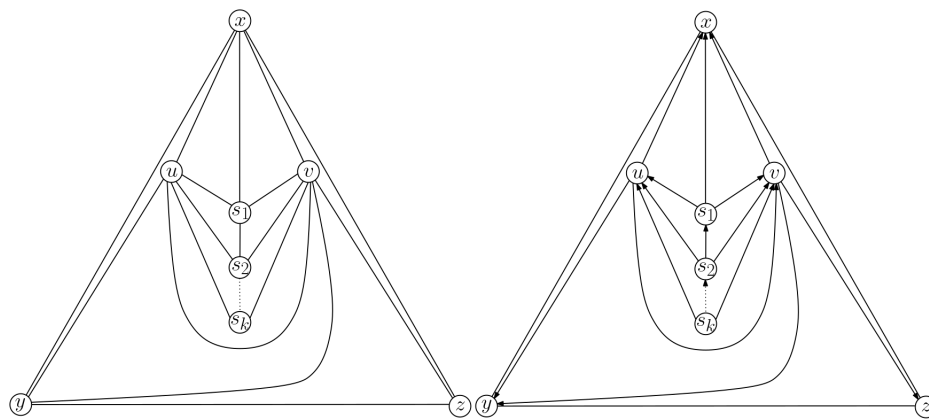
with a different restriction. The other direction is as follows: suppose that there exists an orientation for which the restriction to H is acyclic, but that this restriction is not unique. Comparing two such restrictions gives an edge which is oriented differently in the two orientations. Now, since every point has out-degree 3 an endpoint cannot be incident to only one such edge, so there is a new edge - oriented differently by the two orientations - that one can follow. Continuing like this eventually gives you a directed cycle in H as, by above, one never reaches x, y, z . This is a contradiction.

The Lemma then follows by noting that having an acyclic 3-bounded out-degree orientation is equivalent to being 3-degenerate. Indeed, beginning at an arbitrary vertex in H and following incoming edges backwards ensures that one ends up in a source in H . This source has degree at most 3. We can remove this vertex and apply induction to see that any subgraph not containing this vertex also has a vertex of degree at most 3. The other direction follows, since the 3-degeneracy implies that in any non-empty subgraph $S \subset H$ one can always find a vertex $v \in S$ of degree 3 in $G[S \cup \{x, y, z\}]$. Beginning from H one can remove such a vertex and orient its incident edges so that it becomes a source. Continuing like this never creates cycles and therefore yields an acyclic 3-bounded out-degree orientation. ◀

As noted earlier, we give the lower bound by first considering explicit 3-orientations in plane graphs. We have the following lemma:

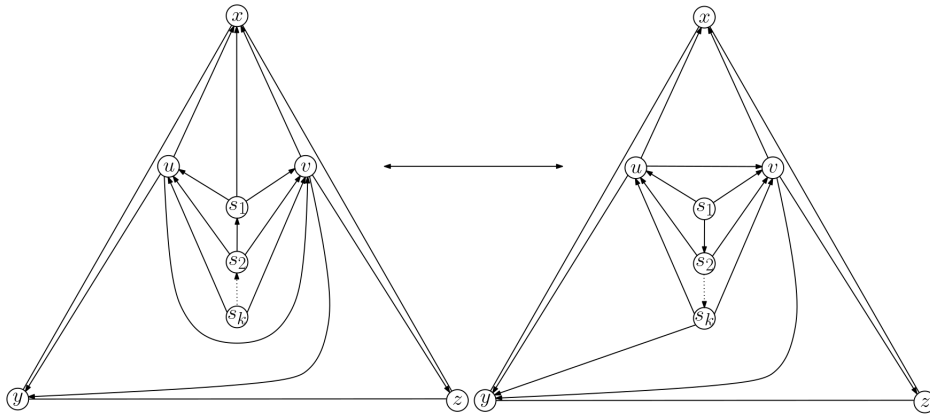
► **Lemma 19.** *Let \mathcal{A} be an algorithm explicitly maintaining a 3-orientation of an n -vertex plane triangulation under diagonal flips. Then the flip operation can be made to spend $\Omega(n)$ update time, even when considering amortised complexity.*

Proof. Consider the following plane triangulation containing a path s_1, s_2, \dots, s_k of length $k = n - 5 = \Omega(n)$ (see Figure 3). The plane triangulation is 3-degenerate, and hence by Lemma 18, it has a unique 3-orientation: By diagonally flipping the edge uv and subsequently



■ **Figure 3** The plane triangulation along with its unique 3-orientation.

the edge s_1x , one gets a new plane triangulation. It is again 3-degenerate, and hence by Lemma 18 it has a unique 3-orientation. (see Figure 4). By diagonally flipping the same edges in the opposite order, one reclaims the original graph. The new 3-orientation has $\Omega(n)$ edges oriented differently compared to the original 3-orientation, but it only requires a constant number of diagonal flips to go between the two graphs. Hence, a constant number of flips forces \mathcal{A} to change the orientation of $\Omega(n)$ edges, and thus, the update time for the diagonal flip operation must be $\Omega(n)$, even amortised, as one can force this update sequence as many times as desired. ◀



■ **Figure 4** Going between two unique 3-orientations.

3-Bounded Out-Orientations. The goal now is to extend this lower bound to 3-bounded out-degree orientations of planar graphs under insertion/deletion of edges. There are only three things to consider before doing such an extension. Firstly, now we support the operations insertion/deletion of edges and not the diagonal flip. This is however a non-issue since a diagonal flip can be simulated by first deleting the edge and then inserting the other diagonal. Secondly, we now consider 3-bounded out-degree orientations and so outer vertices also have out-edges, and not all inner vertices are required to have out-degree 3. Lastly, the lower bound should apply not only to plane graphs where an embedding is chosen, but also to planar graphs. We deal with the last two points by using at least 13 copies of the graph from above. Then, in at least one of the copies, all inner vertices must have out-degree 3 - both before and after a sequence of updates. Hence, we can do the aforementioned updates in all 13 copies, and this will then ensure that at least one copy has to behave as in Lemma 19. Now, we show Theorem 5:

► **Theorem 20** (Identical to Theorem 5). *Let \mathcal{A} be an algorithm explicitly maintaining a 3-bounded out-degree orientation of an n -vertex planar graph under insertion and deletion of edges. Then there exists a sequence of updates taking $\Omega(n)$ amortised time per update.*

Proof. Create a planar graph G by placing 13 copies of the graph P from the proof of Lemma 19 in the plane, and triangulating the outside arbitrarily. Let $n = |V(G)|$. For each copy P_i of P , we let the set I_i resp. O_i be the set of vertices that are inner resp. outer vertices of P_i , if P_i is embedded as in Lemma 19. In particular, for a specific copy of P , say P_i , the corresponding set I_i has size $|I_i| = \frac{n}{13} - 3 = \Omega(n)$. Since G is a plane triangulation, it follows from Euler's Theorem that $|E(G)| = 3n - 6$. This implies that at most 6 vertices of G can have out-degree strictly less than 3. Hence, in at least 7 of the 13 copies of P , every vertex in I has out-degree 3. Since the O vertices of each of these specific copies of P form a triangle, it follows from Lemma 18 that the edges incident to I must have the same orientation as in the plane embedding in Lemma 19 and that this orientation is unique.

Now, we simulate the flip sequence used in Lemma 19 in each copy. Doing this in all 13 copies only requires 26 insertions and 26 deletions in total. After these alterations, it is still the case that in at least 7 of the 13 copies of P every vertex in I has out-degree 3. Furthermore, since the O vertices of each of these specific copies of P form a triangle, it follows from Lemma 18 that the edges incident to I must have the same orientation as in the altered plane embedding in Lemma 19 and that this orientation is unique. At least one copy P_i of P has out-degree 3 at every vertex in I_i in both the orientation before and in

the orientation after the update sequence. Consequentially, \mathcal{A} must have reoriented at least $|I_i| - 3 = \Omega(n)$ edges during the update sequence. The update sequence consists of only a constant number of updates, and it can be reversed by first deleting uv and re-inserting it across the opposite diagonal in each copy, and then doing the same for $s_k y$ in every copy of P . Each reversal of the update sequence, requires only a constant number of updates, but forces \mathcal{A} to change at least $\Omega(n)$ edge-orientations. It follows that there exists a sequence of updates taking $\Omega(n)$ amortised time per update. ◀

6 Conclusion

We have shown how to dynamically maintain 4-bounded out-orientations and 4-arboricity decompositions with sublinear update time. We extended these algorithms to compute $\alpha + 1$ -bounded out-orientations and arboricity decompositions in α -arboricity bounded dynamic graphs. Finally, we also showed that maintaining a 3-bounded out-orientation of a planar graph explicitly, must take $\Omega(n)$ update time, even amortised. This extends the explicit lower bound of Brodal & Fagerberg to the planar case.

It would be interesting to see, if one can maintain $\alpha + 1$ out-edges or forests in sub-polynomial time. Results of Brodal & Fagerberg [9] and Harris et al. [21] show that the problems of maintaining $\alpha + 1$ out-edges resp. forests have $O(\alpha \log n)$ recourse – how close to this worst-case recourse can one get the update times?

References

- 1 Oswin Aichholzer, Franz Aurenhammer, and Günter Rote. *Optimal graph orientation with storage applications*. SFB-Report , SFB 'Optimierung und Kontrolle'. TU Graz, 1995. Reportnr.: F003-51.
- 2 Stephen Alstrup, Jacob Holm, Kristian De Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, October 2005. doi:10.1145/1103963.1103966.
- 3 Srinivasa Rao Arikati, Anil Maheshwari, and Christos D. Zaroliagis. Efficient computation of implicit representations of sparse graphs. *Discret. Appl. Math.*, 78(1-3):1–16, 1997. doi:10.1016/S0166-218X(97)00007-3.
- 4 Niranka Banerjee, Venkatesh Raman, and Saket Saurabh. Fully dynamic arboricity maintenance. In *Computing and Combinatorics – 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*, volume 11653 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2019. doi:10.1007/978-3-030-26176-4_1.
- 5 Edvin Berglin and Gerth Stolting Brodal. A Simple Greedy Algorithm for Dynamic Graph Orientation. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ISAAC.2017.12.
- 6 Markus Blumenstock. Fast algorithms for pseudoarboricity. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2016, Arlington, Virginia, USA, January 10, 2016*, pages 113–126. SIAM, 2016. doi:10.1137/1.9781611974317.10.
- 7 Markus Blumenstock and Frank Fischer. A constructive arboricity approximation scheme. In *SOFSEM 2020: Theory and Practice of Computer Science – 46th International Conference on Current Trends in Theory and Practice of Informatics, SOFSEM 2020, Limassol, Cyprus, January 20-24, 2020, Proceedings*, volume 12011 of *Lecture Notes in Computer Science*, pages 51–63. Springer, 2020. doi:10.1007/978-3-030-38919-2_5.
- 8 Enno Brehm. 3-orientations and schnyder 3-tree-decompositions, 2000.

- 9 Gerth Stolting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In *In Proc. 6th International Workshop on Algorithms and Data Structures (WADS)*, pages 342–351. Springer-Verlag, 1999.
- 10 Aleksander B. G. Christiansen. Dynamic algorithms for implicit vertex-colouring of graphs with bounded arboricity. Master’s thesis, Technical University of Denmark, Kgs. Lyngby, Denmark, October 2021.
- 11 Aleksander B. G. Christiansen and Eva Rotenberg. Fully-dynamic $\alpha + 2$ arboricity decompositions and implicit colouring. In *ICALP*, volume 229 of *LIPICs*, pages 42:1–42:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 12 Marek Chrobak and David Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theor. Comput. Sci.*, 86(2):243–266, 1991.
- 13 Giuseppe Di Battista and Roberto Tamassia. Incremental planarity testing (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October – 1 November 1989*, pages 436–441, 1989. doi:10.1109/SFCS.1989.63515.
- 14 Jack Edmonds. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, page 67, 1965.
- 15 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Inf. Process. Lett.*, 51(4):207–211, August 1994. doi:10.1016/0020-0190(94)90121-X.
- 16 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 599–608, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644208>.
- 17 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification: I. planarity testing and minimum spanning trees. *Journal of Computer and Systems Sciences*, 52(1):3–27, February 1996. doi:10.1006/jcss.1996.0002.
- 18 Harold Gabow and Herbert Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC ’88*, pages 407–421, New York, NY, USA, 1988. Association for Computing Machinery.
- 19 Harold N. Gabow. Algorithms for graphic polymatroids and parametric s-sets. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’95*, pages 88–97, USA, 1995. Society for Industrial and Applied Mathematics.
- 20 Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. Fully dynamic planarity testing with applications. *Journal of the ACM*, 46(1):28–91, 1999. doi:10.1145/300515.300517.
- 21 David G. Harris, Hsin-Hao Su, and Hoa T. Vu. On the locality of nash-williams forest decomposition and star-forest decomposition. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC ’21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 295–305. ACM, 2021. doi:10.1145/3465084.3467908.
- 22 Meng He, Ganggui Tang, and N. Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *ISAAC*, 2014.
- 23 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity. *CoRR*, abs/2002.10142, 2020. arXiv:2002.10142.
- 24 Jacob Holm and Eva Rotenberg. Fully-dynamic planarity testing in polylogarithmic time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 167–180. ACM, 2020. doi:10.1145/3357713.3384249.
- 25 Jacob Holm and Eva Rotenberg. Worst-case polylog incremental spqr-trees: Embeddings, planarity, and triconnectivity. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2378–2397. SIAM, 2020. Full version: arXiv:1910.09005. doi:10.1137/1.9781611975994.146.

- 26 Giuseppe F. Italiano, Johannes A. La Poutré, and Monika Rauch. Fully dynamic planarity testing in planar embedded graphs (extended abstract). In *Algorithms – ESA ’93, First Annual European Symposium, Bad Honnef, Germany, September 30 – October 2, 1993, Proceedings*, pages 212–223, 1993. doi:10.1007/3-540-57273-2_57.
- 27 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2014. doi:10.1007/978-3-662-43951-7_45.
- 28 Łukasz Kowalik. Approximation scheme for lowest outdegree orientation and graph density measures. In *Proceedings of the 17th International Conference on Algorithms and Computation, ISAAC’06*, pages 557–566, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11940128_56.
- 29 Łukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Inf. Process. Lett.*, 102(5):191–195, May 2007. doi:10.1016/j.ipl.2006.12.006.
- 30 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(vrank)$ iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433, 2014. doi:10.1109/FOCS.2014.52.
- 31 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262, 2013. doi:10.1109/FOCS.2013.35.
- 32 C. St.J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 1961. doi:10.1112/jlms/s1-36.1.445.
- 33 C. St.J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, s1-39(1):12–12, 1964. doi:10.1112/jlms/s1-39.1.12.
- 34 Jean-Claude Picard and Maurice Queyranne. A network flow solution to some nonlinear 0-1 programming problems, with applications to graph theory. *Networks*, 12(2):141–159, 1982. doi:10.1002/net.3230120206.
- 35 Johannes A. La Poutré. Alpha-algorithms for incremental planarity testing (preliminary version). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 706–715, 1994. doi:10.1145/195058.195439.
- 36 Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, STOC ’81*, pages 114–122, New York, NY, USA, 1981. Association for Computing Machinery. doi:10.1145/800076.802464.
- 37 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3), June 2020. doi:10.1145/3392724.
- 38 Jeffery Westbrook. Fast incremental planarity testing. In *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, pages 342–353, 1992. doi:10.1007/3-540-55719-9_86.