Orienting Dynamic Graphs, with Applications to Maximal Matchings and Adjacency Queries[⋆]

Meng He, Ganggui Tang, and Norbert Zeh

Faculty of Computer Science, Dalhousie University, Canada.
{mhe, gtang, nzeh}@cs.dal.ca

Abstract. We consider the problem of edge orientation, whose goal is to orient the edges of an undirected dynamic graph with n vertices such that vertex out-degrees are bounded, typically by a function of the graph's arboricity. Our main result is to show that an $O(\beta\alpha)$-orientation can be maintained in $O(\frac{\lg(n/(\beta\alpha))}{\beta})$ amortized edge insertion time and $O(\beta\alpha)$ worst-case edge deletion time, for any $\beta \geq 1$, where α is the maximum arboricity of the graph during update. This is achieved by performing a new analysis of the algorithm of Brodal and Fagerberg [2]. Not only can it be shown that these bounds are comparable to the analysis in Brodal and Fagerberg [2] and that in Kowalik [7] by setting appropriate values of β, it also presents tradeoffs that can not be proved in previous work. Its main application is an approach that maintains a maximal matching of a graph in $O(\alpha + \sqrt{\alpha \lg n})$ amortized update time, which is currently the best result for graphs with low arboricity regarding this fundamental problem in graph algorithms. When α is a constant which is the case with planar graphs, for instance, our work shows that a maximal matching can be maintained in $O(\sqrt{\lg n})$ amortized time, while previously the best approach required $O(\lg n/\lg\lg n)$ amortized time [13]. We further design an alternative solution with worst-case time bounds for edge orientation, and applied it to achieve new results on maximal matchings and adjacency queries.

1 Introduction

The problem of orienting the edges of a dynamic undirected graph to guarantee a low upper bound on the maximum out-degree of its vertices has attracted much attention in recent years [2, 7, 13, 6]. In this problem, an orientation of a graph $G = (V, E)$ is a directed graph $\overrightarrow{G} = (V, \overrightarrow{E})$ defined by assigning each edge of G a direction, and $\overrightarrow{G}$ is further called a Δ-orientation if the out-degree of each vertex in $\overrightarrow{G}$ is upper bounded by Δ. The goal is to maintain a Δ-orientation of G with efficient support of edge insertion and deletion, such that the value of Δ is as small as possible. For dense graphs, Δ has to be large, and thus this problem is more interesting when the graph is sparse.

As the arboricity of a graph is often used as a measurement of the sparsity of the graph, it is typically used as a parameter when bounding Δ. The

[⋆] This work is supported by NSERC and the Canada Research Chairs Program.

arboricity, α, of a graph G can be formally defined by $\alpha = \max\limits_{J} \frac{|E(J)|}{|V(J)-1|}$, where $J = (V(J), E(J))$ is any subgraph of G induced by at least two vertices. Many classes of graphs in practice have constant arboricity, including planar graphs, graphs with bounded genus and graphs with bounded tree width. Nash-Williams [11, 12] proved that G has arboricity α if and only if α is the smallest number of subsets that E can be partitioned into, such that each subset of edges with their endpoints is a forest. Such a decomposition can be computed in polynomial time [3]. In this partition, if we orient each edge in a forest towards the root of the tree containing this edge, then each vertex has out-degree at most one in each forest, which immediately gives an α-orientation of the given *static* graph.

The most fundamental application of edge orientation is perhaps the representation of dynamic graphs supporting adjacency queries. This is based on the following observation [5]: With a Δ-orientation of G, if we store the at most Δ out-neighbors of each vertex in a list, then an adjacency query can be answered in $O(\Delta)$ time by scanning the list of each of the two vertices given in the query, to see if one is an out-neighbour of the other. Thus if we can maintain a Δ-orientation of a sparse graph efficiently, then we immediately have a linear-space dynamic graph representation that answers adjacency queries in $O(\Delta)$ time [2].

Recently, Neiman and Solomon [13] found that edge orientation also has applications in maintaining a maximal matching of a dynamic graph. A *matching*, M, of a graph G is a set of non-adjacent edges of G. If a matching M has the maximum number of edges, then it is called a *maximum cardinality matching*. A *maximal matching* is defined to be a matching, M, that satisfies the following condition: there does not exist an edge, g, of G, such that $M \cup \{g\}$ is still a matching of G. It is well-known that any maximal matching is a 2-approximation for maximum cardinality matching. Graph matching is a fundamental problem in graph theory, and it has many applications in combinatorial optimization [10]. In the dynamic setting, the problem is to maintain a maximal matching or an approximate maximum cardinality matching under edge insertion and deletion. Recent progress on this [13, 4, 6] generated more interests in edge orientation.

Edge orientation has also been applied to other problems such as shortest path in dynamic planar graphs [9, 6] and graph colouring [8]. Motivated by all these important applications, we study the problem of orienting dynamic graphs.

1.1 Previous Work

Brodal and Fagerberg [2] first studied the problem of maintaining an edge orientation of a dynamic graph with n vertices under an arboricity α preserving sequence of edge insertions and deletions. Here an update operation is considered *arboricity α preserving* if, when applied to an graph of arboricity at most α, the arboricity of the graph after the update remains to be bounded by α. They proposed an approach that can maintain an $O(\alpha)$-orientation using $O(m + n)$ space, where m is the current number of edges, in $O(1)$ amortized insertion time and $O(\alpha + \lg n)$ amortized deletion time[1]. In their algorithm for update

[1] In this paper, $\lg n$ denotes $\log_2 n$.

operations, some edges may change their orientation after each update, i.e., be *reoriented*. They proved that in terms of the amortized number of edge reorientations per update, their algorithm is $O(1)$-competitive compared against any algorithm. Kowalik [7] further showed that Brodal and Fagerberg's approach can maintain an $O(\alpha \lg n)$-orientation with constant amortized insertion time and constant worst-case deletion time. More recently, Kopelowitz et al. [6] considered the problem of designing solutions to maintain edge orientation with worst-case time bounds. They showed how to maintain an $O(\Delta)$-orientation in $O(m+n)$ space with $O(\beta\alpha\Delta)$ worst-case insertion time and $O(\Delta)$ worst-case deletion time, where $\Delta \leq \inf_{\beta>1}\{\beta\alpha + \lceil \log_\beta n \rceil\}$.

For maintaining matchings in arbitrary graphs, we refer to the recent work of Neiman and Solomon [13] which can maintain a maximal matching (which is also a 3/2-approximate maximum cardinality matching) in $O(\sqrt{m})$ worst-case update time, and the work of Gupta and Peng [4] which maintains a $(1+\epsilon)$-approximation maximum cardinality matching with $O(\sqrt{m}\epsilon^{-2})$ update time for any $\epsilon > 0$. To support more efficient updates for graphs with low arboricity, Neiman and Solomon [13] showed how to use edge orientation to maintain a maximal matching. Their approach can maintain a maximal matching in $O(m+n)$ space, such that each update can be performed in $O(\Delta + \log_{\Delta/\alpha} n)$ amortized time for any $\Delta > 2\alpha$. When $\alpha = o(\lg n)$, the update time becomes $O(\frac{\lg n}{\lg((\lg n)/\alpha)} + \alpha)$. Following the same idea, Kopelowitz et al. [6] made use of their solution for edge orientation to maintain a maximal matching, and the worst-case update time is asymptotically the same as their update time for maintaining edge orientation summarized in the previous paragraph.

As discussed previously, solutions to maintaining edge orientation can be directly used to represent dynamic graphs to support adjacency queries. Kowalik [7] showed that by maintaining the list of the out-neighbours of each vertex using the dynamic dictionary of Andersson and Thorup [1], a graph can be represented in $O(m+n)$ space to support adjacency query and edge deletion in $O(\lg\lg\lg n)$ worst-case time, and edge insertion in $O(\lg\lg\lg n)$ amortized time, provided that $\alpha = O(\text{polylog}(n))$. Using the same strategy, Kopelowitz et al. [6] presented a linear-space representation of graphs with $\alpha = \text{polylog}(n)$ arboricity that can support adjacency queries in $O(\lg\lg\Delta)$ worst-case time, edge insertion in $O(\beta\alpha\Delta\lg\lg\Delta)$ worst-case time, and edge deletion $O(\Delta\lg\lg\Delta)$ worst-case time, where $\Delta \leq \inf_{\beta>1}\{\beta\alpha + \lceil \log_\beta n \rceil\}$.

1.2 Our Results

We first analyzed the algorithm of Brodal and Fagerberg [2], by constructing a new offline algorithm for their main reduction (summarized in Lemma 1). Our new analysis shows that an $O(\beta\alpha)$-orientations can be maintained in linear space with $O(\frac{\lg(n/(\beta\alpha))}{\beta})$ amortized insertion time and $O(\beta\alpha)$ worst-case deletion time, for any $\beta \geq 1$. Furthermore, no edge orientation is required when performing edge deletion. This presents a tradeoff between the maximum out-degree of vertices and insertion time in the analysis of the algorithm by Brodal and Fagerberg, which was never proved before. If we set $\beta = 1$, then our analysis shows that this algorithm maintains an $O(\alpha)$-orientation while supporting insertion in $O(\lg n)$ amortized time and deletion in $O(\alpha)$ worst-case time. This is comparable to Brodal and Fagerberg's own analysis. By setting $\beta = \lg n$, the algorithm maintains an $O(\alpha \lg n)$-orientation with a constant number of edge reorientations per edge insertion in the amortized sense and zero reorientation for each deletion, which matches Kowalik [7]'s analysis.[2] When $\beta = \sqrt{\lg n}$, this algorithm maintains an $O(\alpha\sqrt{\lg n})$-orientation with $O(\sqrt{\lg n})$ amortized insertion time and $O(\alpha\sqrt{\lg n})$ worst-case deletion time. This tradeoff can not be shown using previous analysis.

We then apply our result on edge orientation to improve previous results on maintaining maximal matchings under arboricity α preserving update sequences. More specifically, we can maintain a maximal matching using $O(m+n)$ space in $O(\alpha + \sqrt{\alpha \lg n})$ amortized update time, which is currently the best result on maintaining a maximal matching for low arboricity graphs. Our result matches the result of Neiman and Solomon [13] when $\alpha = \Omega(\lg n)$, while strictly improves their results when $\alpha = o(\lg n)$. To see the improvement when $\alpha = o(\lg n)$, suppose $\alpha = \frac{\lg n}{f(n)}$, where $f(n)$ is an arbitrary function in $\omega(1)$. Then Neiman and Solomon's result supports updates in $O(\frac{\lg n}{\lg f(n)})$ amortized time, while ours requires $O(\frac{\lg n}{\sqrt{f(n)}})$. The improvement is even obvious for graphs with constant arboricity such as planar graphs: a maximal matching can be maintained in $O(\sqrt{\lg n})$ amortized time with our work, while previously it required $O(\lg n / \lg\lg n)$ amortized time, and this improvement is surprising.

We further design solutions to these problems that guarantee worst-case time bounds. We show how to maintain a Δ-orientation in $O(\Delta)$ worst-case insertion and deletion time, where $\Delta \leq 2\alpha \lg(n/\alpha) + 2\alpha$. This is a new tradeoff when compared with the result of Kopelowitz et al. [6]: When $\alpha = \omega(\lg n)$, our insertion time is $O(\alpha \lg n)$, which is better than their $O(\alpha^2)$ insertion time, though our maximum out-degree and deletion time are worse. It is noteworthy that our approach is simpler and does not require edge reorientation during insertion. The same bounds can be proved when applying our result to maintain a maximal matching, which again compares similarly to the result of Kopelowitz et al. We can also use this to represent a graph with $O(\text{polylog}(n))$ arboricity to support adjacency queries in $O(\lg\lg\Delta)$ worst-case time, edge insertion in $O(\Delta)$ worst-case time, and edge deletion in $O(\Delta \lg\lg\Delta)$ worst-case time. For graphs with constant arboricity such as planar graphs, our representation supports adjacency query, insertion and deletion in $O(\lg\lg\lg n)$, $O(\lg n)$ and $O(\lg n \lg\lg\lg n)$ time, respectively, improving Kopelowitz et al.'s result which provides the same

[2] Kowalik [7]'s analysis in deletion time does not include the time required to find the location of the given edge within the list of out-neighbours of one of its endpoints and thus his model implicitly requires such a location to be given when performing deletion. In our work, unless otherwise specified, we follow the original model of Brodal and Fagerberg [2], which maintains out-neighbours in linked lists, and the time required to search each list for the edge to be deleted is part of deletion time. Thus when comparing with Kowalik's analysis, we consider the number of reorientations.

support for query and deletion, but requires $O(\lg n \lg\lg\lg n)$ time for insertion. The fact that our insertion algorithm for edge orientation does not require reorientation makes such an improvement possible. For non-constant α, our result is a new tradeoff: our insertion is faster than [6] but query and deletion may be slower. All our solutions use $O(n+m)$ space.

2 Preliminaries

2.1 Reduction from Online Orientations to Offline Orientations

Brodal and Fagerberg [2] analyzed their algorithm by reducing the problem of maintaining an edge orientation under online updates to the problem of finding a sequence of orientations for an update sequence given offline. A variant of their reduction to be used in our solution can be summarized as:

Lemma 1 ([2]). *Given an arbitrary arboricity α preserving sequence of edge insertions and deletions over an initially empty graph, let G_0 denote the initial empty graph, G_i denote the graph after the ith operation, and k denote the number of edge insertions.*

If there exists a sequence $\overrightarrow{G}_0, \overrightarrow{G}_1, \ldots, \overrightarrow{G}_{p+q}$ of δ-orientations that incurs at most kr edge reorientations in total for a certain r, then starting with the empty graph on n vertices under arbitrary arboricity α preserving updates, a Δ-orientation can be maintained using $O(m+n)$ space, where m is the current number of edges, such that each edge insertion can be performed in $O(\frac{r(\Delta+1)}{\Delta+1-2\delta})$ amortized time, and an edge deletion in $O(\Delta)$ worst-case time, provided $\Delta \geq 2\delta > 2\alpha$. Furthermore, the amortized number of edge reorientations incurred during each insertion is $O(\frac{r(\Delta+1)}{\Delta+1-2\delta})$, and deletion requires no reorientation.

2.2 Data Structures for Dynamic Sets with Center Elements

Kopelowitz et al. [6] defined the following data structure problem to help them maintain the invariants in their work, and we will also make use of this data structure in our solution with worst-case time bounds: Let X be a dynamic set, in which each element $x_i \in X$ is associated with a nonnegative integer key k_i. The element x_0 is designated as the center element of X which can not be inserted or deleted, but the value of its key can be updated. The goal is to support the following operations:

- `ReportMax`(X): return a pointer to an element in X with the maximum key;
- `Increment`(X, x): Given a pointer to $x \in X \setminus \{x_0\}$, increment x's key;
- `Decrement`(X, x): Given a pointer to $x \in X \setminus \{x_0\}$, decrement x's key;
- `Insert`(X, x, k): Insert a new element x with key k into X, provided $k \leq k_0 + 1$;
- `Delete`(X, x): Given a pointer to $x \in X \setminus \{x_0\}$, remove x from X;
- `IncrementCenter`(X): Increment k_0;
- `DecrementCenter`(X): Decrement k_0.

The following lemma summarizes a solution to this problem:

Lemma 2 ([6]). *Let X be a dynamic set in which each element x_i is associated with a key k_i and a fixed element x_0 is designated to be X's center. Then X can be maintained in $O(|X| + k_0)$ space to support* `ReportMax`*,* `Increment`*,* `Decrement`*,* `Insert` *and* `Delete` *in $O(1)$ time, and* `IncrementCenter` *and* `DecrementCenter` *in $O(k_0)$ time.*

3 Solutions with Amortized Time Bounds

In this section we first present a new offline algorithm to orient fully dynamic graphs. Then we make use of Lemma 1 to prove our result on maintaining edge orientation under online update operations.

In our offline strategy, let U be an arbitrary arboricity α preserving update sequence on an initially empty graph G with n vertices. Denote by G_i the graph after the ith update as in Lemma 1 (G_0 denotes the initial empty graph). We now show how to determine a sequence of δ-orientations $\overrightarrow{G}_0, \overrightarrow{G}_1, \ldots, \overrightarrow{G}_U$ with a provable upper bound on the total number of edge reorientations, for a parameter δ to be determined later. Note that it is trivial to orient the empty graph G_0.

We first divide U into *phases* each containing $\beta\alpha n$ consecutive update operations, except the last phase which may contain fewer operations, where $\beta \geq 1$. For simplicity, we assume that $\beta\alpha n$ is an integer. For the graph at the end of each phase that contains $\beta\alpha n$ operations, we compute an α-orientation using the approach described in the second paragraph of Section 1, which makes use of the algorithm in [3]. This determines the orientation of the graph at the end of each phase with the possible exception of the last phase, i.e., $\overrightarrow{G}_{\beta\alpha n}$, $\overrightarrow{G}_{2\beta\alpha n}$, $\overrightarrow{G}_{3\beta\alpha n}$, $\ldots$, $\overrightarrow{G}_{\lfloor |U|/(\beta\alpha n)\rfloor(\beta\alpha n)}$. To further orient G_i where i is not divisible by $\beta\alpha n$, we have the following definition:

Definition 1. *Consider a phase, P, of $\beta\alpha n$ consecutive updates on a graph G with n vertices, in which an update operation that inserts or deletes an edge between vertices x and y is said to* update *x and y. A vertex of G is* hot *in P if it is updated by at least $4\beta\alpha$ operations of P, and* cold *otherwise. The* hot region, *$H(G)$, of G in P is the subgraph of G induced by all the hot vertices of G in P, while the* cold region, *$C(G)$, of G in P is defined to be $G \setminus H(G)$.*

The δ-orientation sequence is determined recursively. We use the following strategy for each phase, P, of U. Without loss of generality, we assume that $|P| = \beta\alpha n$. Let G_{i+j} denote the graph after the jth operation in P. Thus G_i denotes the graph immediately before any operation in P is performed, and by our previous discussion, $\overrightarrow{G}_i$ is a α-orientation of G. We determine the orientations of some of the edges in G_{i+j} for $j \in [1..\beta\alpha n - 1]$ in increasing order of j: For an edge that is present in both G_{i+j} and G_{i+j-1}, if its orientation in G_{i+j-1} has already been determined, then in G_{i+j}, we maintain the same orientation. There are no new edges to be oriented in G_{i+j} if the jth operation in P deletes an edge. If this operation inserts an edge instead, then there are three cases. In the first case, this edge is between a hot vertex and a cold vertex, and we orient it from the cold vertex to the hot vertex. In the second case, the edge is between two cold vertices, and we orient it arbitrarily. In the remaining case, the edge is between two hot vertices, and we do not orient this edge in this level of recursion.

So far we have finished describing our top-level partition, which determines $\overrightarrow{G}_i$ for i divisible by $\beta\alpha n$, and for all other G_i's, it determines the orientations of the edges that are not inserted as an edge between two hot vertices during the phase containing this insertion. Then, for each phase, P, of U, let n' denote the number of vertices of G that are hot vertices in this phase. As each hot vertex is updated by at least $4\beta\alpha$ operations in P and each operation may update up to two hot vertices, the number of operations in P that update hot vertices is at least $2\beta\alpha n'$. As this can not be larger than the total number of operations in P, we have $2\beta\alpha n' \leq \beta\alpha n$, which implies $n' \leq n/2$. If $n' < 4\beta\alpha$, we arbitrarily orient the edges inserted between these hot vertices by operations in phase P excluding the last operation (recall that after the last operation, the graph is oriented by computing an α-orientation, so we exclude the last operation here). Otherwise, we set n to be n', set U to be the sequence of operations in P that update hot vertices only, and apply the same recursive strategy to $H(G)$. Upon returning from the recursion on $H(G)$, the direction of each edge inserted between hot vertices have been decided as it is part of the graph $H(G)$. Thus we have oriented all the G_i's. We now bound vertex out-degrees:

Lemma 3. *The offline algorithm in this section computes a sequence of* $(4\beta\alpha + \alpha)$*-orientations* $\overrightarrow{G}_0, \overrightarrow{G}_1, \ldots, \overrightarrow{G}_{p+q}$.

Proof. We prove by induction that at each level of recursion, we construct $(4\beta\alpha + \alpha)$-orientations throughout each phase. In the base case where we stop the recursion, we consider a graph with at most $4\beta\alpha$ vertices. In this case, even though we orient edges arbitrarily upon insertion, the maximum out-degree of any vertex is at most $4\beta\alpha - 1$ as the total number of vertices is at most $4\beta\alpha$.

In the inductive case, for an arbitrary phase P, let G_{i+j} denote the graph after the jth operation in P. Assume inductively that the out-degree of any vertex in $H(G)$ is at most $4\beta\alpha + \alpha$ during the execution of the operations in P, and we now prove the same claim for G. We first consider an arbitrary cold vertex x in this phase. Before any operation in P is performed, in G_i, the out-degree of x is at most α as $\overrightarrow{G_i}$ is computed as an α-orientation. By Definition 1, less than $4\beta\alpha$ edges inserted in P have x as an endpoint. Thus the maximum out-degree of x in phase P is less than $\alpha + 4\beta\alpha$. We then argue about an arbitrary hot vertex y. As any edge between y and a cold vertex is oriented towards y, the out-degree of y is always equal to its out-degree in $H(G)$, which is bounded by $4\beta\alpha + \alpha$ by inductive hypothesis. □

To bound the total number of edge reorientations, we have:

Lemma 4. *The total number of edge reorientations among* $\overrightarrow{G}_0, \overrightarrow{G}_1, \ldots, \overrightarrow{G}_U$ *is* $O(\frac{|U| \lg(n/(\beta\alpha))}{\beta})$.

Proof. We number each level of recursion by its recursion depth starting from 0. Thus at level 0, we consider the original graph G with n vertices. At level 1, each of the subgraphs being considered corresponds to a phase at level 0 and contains the hot region of G in this phase which has at most $n/2$ vertices, and so on. The number of vertices in each subgraph considered at level i is thus at most $n/2^i$, and the number of vertices of each graph considered at the last level is at most $4\beta\alpha$. Therefore, the number of levels is $O(\lg(n/(\beta\alpha)))$ and the number of edges in each subgraph considered at level i is at most $\alpha(n/2^i - 1)$.

Note that at any given level, reorientation only happens at the end of each phase defined for a subgraph at that level, when we recompute an a-orientation and use it to orient the subgraph. We also observe that each operation in U may be considered at most once at each level of partition. As the number of levels is $O(\lg(n/(\beta\alpha)))$, it suffices to prove that, when amortizing the number of reorientations at the end of each phase at any level over the operations in that phase, the number of reorientations charged to each operation in this phase is at most $1/\beta$. To see this, let t denote the number of vertices in a subgraph considered at an arbitrary level. By our algorithm, the update sequence considered for this subgraph is divided into phases each containing $\beta\alpha t$ operations, except the last phase which may contain fewer. Edge reorientations take place at the end of each phase that contains exactly $\beta\alpha t$ operations. As the total number of edges in the subgraph is at most $\alpha(t-1)$, the number of edge reorientations at the end of each such phase is thus at most $\alpha(t-1)$. When amortizing these edge reorientations over the $\beta\alpha t$ operations in the phase, each update is charged at most $\alpha(t-1)/(\beta\alpha t) < 1/\beta$ edge reorientations. □

Combining Lemmas 3 and 4, we have:

Lemma 5. *Given an arboricity α preserving sequence of edge insertions and deletions on an initially empty graph and an arbitrary parameter $\beta \geq 1$, there is a sequence of $(4\beta\alpha + \alpha)$-orientations such that the amortized number of edge reorientation for each edge insertion or deletion is $O(\frac{\lg(n/(\beta\alpha))}{\beta})$.*

We now present our first main result:

Theorem 1. *Starting with the empty graph on n vertices under arboricity α preserving updates, a Δ-orientation can be maintained in $O(n+m)$ space, where $\Delta \geq 2\delta$, $\delta = (4\beta+1)\alpha$, β is an arbitrary parameter greater or equal to 1 and m is the current number of edges, such that an edge insertion can be performed in $O(\frac{\lg(n/(\beta\alpha))}{\beta} \cdot \frac{\Delta+1}{\Delta+1-2\delta})$ amortized time, and an edge deletion in $O(\Delta)$ worst-case time. Furthermore, edge deletion does not incur edge reorientation.*

Proof. As the graph is initially empty, the number, k, of insertions is greater than or equal to the number, k', of deletions in U. Thus Lemma 5 shows that the total number of edge reorientations is $O(\frac{(k+k')\lg(n/(\beta\alpha))}{\beta}) \leq O(\frac{2k\lg(n/(\beta\alpha))}{\beta}) = O(k \cdot (\frac{\lg(n/(\beta\alpha))}{\beta}))$. The theorem thus follows from Lemma 1. □

The tradeoff summarized in Section 1.2 is obtained by setting $\Delta = 3\delta$. By applying this to maximal matchings, we have the following theorem:

Theorem 2. *Starting with the empty graph on n vertices under arboricity α preserving updates, a maximal matching can be maintained in $O(\alpha + \sqrt{\alpha \lg n})$ amortized time using $O(n + m)$ space, where m is the current number of edges.*

Proof. Neiman and Solomon [13] made use of the algorithm of Brodal and Fagerberg [2] to maintain maximal matchings in dynamic settings. Their reduction shows that if a Δ-orientation for a graph G on n vertices under arboricity α preserving updates can be maintained in $O(m+n)$ space with amortized update time T, where m denotes the current number of edges, then a maximal matching can also be maintained in $O(m+n)$ space with $O(\Delta+T)$ amortized update time.

We first observe that, according to Neiman and Solomon's reduction, a maximal matching can be maintained in $O(\beta\alpha + \frac{\lg(n/(\beta\alpha))}{\beta})$ amortized update time using $O(n + m)$ space, following from Theorem 1 by setting $\Delta = 3\delta$. When $\alpha \geq \lg n$, we set $\beta = 1$ and the update time is $O(\alpha)$. Otherwise, we set $\beta = \sqrt{\frac{\lg n}{\alpha}}$, and the update time becomes $O(\sqrt{\alpha \lg n})$. The theorem thus follows. $\square$

4 Solutions with Worst-Case Time Bounds

Let $d_o(v)$ denote the out-degree of a vertex v. Our solution with worst-case time bounds maintains the following invariant over the entire graph G during updates:

Invariant 1 *For each vertex u, there exists an ordering of its out-neighbours, $v_0, v_1, v_2, \ldots, v_{d_o(u)-1}$, such that $d_o(v_i) \geq i$ for $i = 0, 1, \ldots, d_o(u) - 1$.*

There are connections between this invariant and the invariants considered by Kopelowitz et al. [6], but they are different. The following two lemmas show why Invariant 1 can be used to bound the maximum vertex out-degree.

Lemma 6. *If the maximum out-degree, Δ, of a vertex in a directed graph G of arboricity α satisfying Invariant 1 is greater than 4α, then there are $2^k\alpha$ vertices whose out-degrees are at least $\Delta - 2k\alpha \geq 2\alpha$, for $k = 1, 2, \ldots, \lfloor \Delta/(2\alpha) \rfloor - 1$.*

Proof. The maximum value of k guarantees that $\Delta - 2k\alpha \geq 2\alpha$. To prove the rest of the lemma besides this inequality, let u be a vertex with out-degree Δ in G. We prove our claim by induction on k. In the base case, $k = 1$. Let $v_0, v_1, v_2, \ldots, v_{\Delta-1}$ be u's out-neighbours listed in the order specified in Invariant 1. Then $d_o(v_{d-1}) \geq d - 1, d_o(v_{d-2}) \geq d - 2, \ldots, d_o(v_{\Delta-2a}) \geq \Delta - 2\alpha$ by Invariant 1, which means u has at least 2α out-neighbours with out-degrees greater than or equal to $\Delta - 2\alpha$.

Assume the claim holds for $k - 1$, and we prove it for k. By the inductive hypothesis, there is a set, V_1, of $2^{k-1}\alpha$ vertices with out-degree at least $\Delta - 2(k - 1)\alpha$. By Invariant 1, each vertex in V_1 has 2α out-neighbours whose out-degrees are at least $\Delta - 2(k - 1)\alpha - 2\alpha = \Delta - 2k\alpha$. We add such 2α out-neighbours of each vertex in V_1 into another set V_2. Note that some vertices in V_1 may share out-neighbors. Any vertex in $V_1 \cup V_2$ has out-degree at least $\Delta - 2k\alpha$, and what remains is to give a lower bound on $|V_1 \cup V_2|$. Consider the subgraph G^* induced by $V_1 \cup V_2$. For each vertex in V_1, there are 2α distinct edges between it and the vertices in V_2, and thus the number of edges in G^* is at least $2\alpha|V_1| = 2^k\alpha^2$. By the definition of arboricity, we have $\alpha \geq \frac{|E(G^*)|}{|V(G^*)|-1} \geq \frac{2^k\alpha^2}{|V_1 \cup V_2|-1}$. Therefore, $|V_1 \cup V_2| \geq 2^k\alpha$. As the out-degree of each vertex in $V_1 \cup V_2$ is at least $\Delta - 2k\alpha$ from our previous discussion, our induction goes through. □

Lemma 7. *If a directed graph G satisfies Invariant 1, then the out-degree of any vertex in G is at most $2\alpha \lg(n/\alpha) + 2\alpha$.*

Proof. Let Δ denote the maximum out-degrees of the nodes in G. If $\Delta \leq 4\alpha$, the lemma holds because, in an undirected graph, we always have $\alpha \leq n/2$ and thus $2\alpha \lg(n/\alpha) + 2\alpha \geq 4\alpha$. Otherwise, by Lemma 6, the number of vertices whose out-degrees are at least 2α is $2^{\lfloor \Delta/(2\alpha) \rfloor - 1}\alpha$. Therefore, the total number of edges of G is at least $2^{\lfloor \Delta/(2\alpha) \rfloor - 1}\alpha \cdot 2\alpha = 2^{\lfloor \Delta/(2\alpha) \rfloor}\alpha^2$. Since the arboricity of G is α, we have $(2^{\lfloor \Delta/(2\alpha) \rfloor}\alpha^2)/(n-1) \leq \alpha$, and thus $(2^{\Delta/(2\alpha)-1}\alpha^2)/(n-1) < \alpha$. Therefore, $\Delta < 2\alpha \lg \frac{n-1}{\alpha} + 2\alpha$. This completes the proof. □

To maintain Invariant 1, we borrow ideas from [6] though our algorithms for edge insertion and deletion turn out to be simpler. As in [6], for each vertex u, we construct a data structure B_u to maintain information for its in-neighbours, which is further used to decide which edges should be reoriented. More precisely, for vertex u, we construct a dynamic set B_u whose center element is u itself, with $d_o(u)$ as its key. $X \setminus \{u\}$ then contains as elements all the in-neighbours of u, and the key for each such element is the out-degree of this in-neighbour. We then represent B_u using Lemma 2. Clearly all these auxiliary data structures use $O(m + n)$ space in total, where m is the current number of edges in G.

We also construct the adjacency lists for G with edge orientations, by maintaining the out-going edges of each vertex in a doubly linked list. This also requires $O(m + n)$ space. For each directed edge (u, v) in u's list, we maintain a bidirectional pointer between this edge and u's representation in B_v. With this, when our algorithm for edge deletion uses `ReportMax` to find an edge for reorientation, we can update adjacency lists in constant time. Such a construction is also required to make the approach in [6] work, though it was not mentioned explicitly. As it is trivial to maintain the adjacency lists with these pointers and the maintenance cost is subsumed by our final time bounds, we do not explicitly discuss how to update these lists in the rest of this section.

To insert an edge uv, assume without loss of generality that $d_o(u) \leq d_o(v)$. Then we orient the edge from u to v. It can be easily shown that with this strategy, Invariant 1 is maintained and no reorientation is required. We further update B_u using `IncrementCenter` and $B_{u'}$ for each out-neighbour, u', of u, using `Increment`. Algorithm 1 presents the pseudo code for edge insertion.

Algorithm 2 presents the pseudocode for edge deletion. It first removes the edge to be deleted in lines 2-3. After this, the out-degree of u is decreased by 1, and the only vertices for which Invariant 1 may not hold have to be in-neighbours of u. To find out whether the invariant is still maintained for all the in-neighbours of u, we locate the in-neighbour, v', with the largest out-degree in line 4. If the test in the while statement at line 5 is false, then the invariant still holds for

Algorithm 1 Insert(G, u, v)

```
1: {Assume without loss of generality that $d_o(u) \leq d_o(v)$}
2: Orient edge $(u, v)$ from $u$ to $v$
3: IncrementCenter($B_u$)
4: Insert($B_v, u, d_o(u)$)
5: for each out-neighbour, $u'$, of $u$ such that $u' \neq v$ do
6:    Increment($B_{u'}, u$)
```

Algorithm 2 Deletion: Delete(G, u, v)

```
1: {Assume without loss of generality that the edge $uv$ is oriented towards $v$}
2: Remove edge $(u, v)$
3: Delete($B_v, u$)
4: $v' \leftarrow$ ReportMax($B_u$)
5: while $d_o(u) < d_o(v') - 1$ do
6:    Flip the orientation of edge $(v', u)$ so that it is oriented from $u$ to $v'$
7:    Delete($B_u, v'$)
8:    Insert($B_{v'}, u, d_o(u)$)
9:    $u \leftarrow v'$
10:   $v' \leftarrow$ ReportMax($B_u$)
11: DecrementCenter($B_u$)
12: for each out-neighbour, $v'$, of $u$ do
13:   Decrement($B_{v'}, u$)
```

any in-neighbour of u. Otherwise, it is possible (though not necessary) that the invariant is not maintained for v' and some other in-neighbours of u. To maintain the invariants for these vertices, we reverse the direction of the edge (v', u) in line 6, and update auxiliary data structures accordingly in lines 7-8. After this the out-degree of u becomes the same as its original out-degree before this edge deletion is performed, and thus the invariant can not be violated for any of its in-neighbours whose out-degree did not change. The only in-neighbour whose out-degree has been changed is v', and it is easy to see that the invariant is also maintained for v' as a result of the above steps: v' lost one out-neighbour but its out-degree was also decreased by 1. Now the the only vertices for which Invariant 1 may not hold have to be in-neighbours of v'. For v', we then repeat the same process that we applied to u. This process terminates in at most $\Delta + 1$ iterations, because each time we iterate on a node whose out-degree is strictly greater than the node in the previous iteration and the maximum vertex out-degree is Δ. From the description of this process, we can also claim that, after the while loop in lines 5-10 terminates, the invariant is maintained, and lines 11-13 make sure that all the auxiliary structures are up-to-date.

As our algorithms for edge insertion and deletion maintain Invariant 1, by Lemma 7, they can maintain a Δ-orientation of G for $\Delta = 2\alpha \lg(n/\alpha) + 2\alpha$. To analyze the running time of these two operations, we first observe that each loop in the pseudocode of these two algorithms is iterated at most Δ times. Then, applying Lemma 2, we claim that both operations require $O(\Delta)$ time. Thus:

Theorem 3. *A Δ-orientation of a graph on n vertices can be maintained in $O(n+m)$ space, where $\Delta \leq 2\alpha \lg(n/\alpha) + 2\alpha$, α is the current arboricity and m is the current number of edges, such that an edge insertion or deletion can be performed in $O(\Delta)$ worst-case time. Furthermore, an edge insertion does not incur edge reorientation, while a deletion incurs at most $\Delta + 1$ reorientations.*

If we allow one reorientation in edge insertion, then we can bound Δ by $\min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{m})$ without affecting update times. We omit the details due to page limit. These results can then be easily applied to achieve new results on maximal matchings and adjacency queries in dynamic graphs:

Theorem 4. *A maximal matching of a graph on n vertices can be maintained in $O(\min(\alpha \lg(n/\alpha), \sqrt{m}))$ worst-case update time using $O(n+m)$ space, where α is the current arboricity and m is the current number of edges.*

Theorem 5. *A graph with n vertices and m edges can be represented in $O(m+n)$ space to support adjacency queries in $O(\lg\lg\Delta)$ worst-case time, edge insertion in $O(\Delta)$ worst-case time, and edge deletion in $O(\Delta \lg\lg\Delta)$ worst-case time, where $\Delta = O(\alpha \lg(n/\alpha))$ and α is the current arboricity of the graph, provided $\alpha = O(\mathrm{polylog}(n))$.*

Acknowledgments. We are grateful for Tsvi Kopelowitz for helpful discussions.

References

1. Andersson, A., Thorup, M.: Tight(er) worst-case bounds on dynamic searching and priority queues. In: STOC. pp. 335–342 (2000)
2. Brodal, G.S., Fagerberg, R.: Dynamic representation of sparse graphs. In: WADS. pp. 342–351 (1999)
3. Gabow, H.N., Westermann, H.H.: Forests, frames, and games: Algorithms for matroid sums and applications. Algorithmica 7(5&6), 465–497 (1992)
4. Gupta, M., Peng, R.: Fully dynamic (1+ e)-approximate matchings. In: FOCS. pp. 548–557 (2013)
5. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. SIAM J. Discrete Math. 5(4), 596–603 (1992)
6. Kopelowitz, T., Krauthgamer, R., Porat, E., Solomon, S.: Orienting fully dynamic graphs with worst-case time bounds. In: ICALP (2). pp. 532–543 (2014)
7. Kowalik, L.: Adjacency queries in dynamic sparse graphs. Inf. Process. Lett. 102(5), 191–195 (2007)
8. Kowalik, L.: Fast 3-coloring triangle-free planar graphs. Algorithmica 58(3), 770–789 (2010)
9. Kowalik, L., Kurowski, M.: Oracles for bounded-length shortest paths in planar graphs. ACM Transactions on Algorithms 2(3), 335–363 (2006)
10. Lovász, L., Plummer, M.: Matching Theory. AMS Chelsea Publishing (1986)
11. Nash-Williams, C.S.J.A.: Edge-disjoint spanning trees of finite graphs. Journal of the London Mathematical Society 36(1), 445–450 (1961)
12. Nash-Williams, C.S.J.A.: Decomposition of finite graphs into forests. Journal of the London Mathematical Society 39(1), 12 (1964)
13. Neiman, O., Solomon, S.: Simple deterministic algorithms for fully dynamic maximal matching. In: STOC. pp. 745–754 (2013)