FULLY DYNAMIC GRAPH ORIENTATION

by

Ganggui Tang

Submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
December 2014

Table of Contents

List of Tables

List of Figures

Abstract

In this thesis we consider the problem of edge orientation, where the goal is to orient the edges of an undirected dynamic graph with n vertices so that the out-degree of every vertex is bounded, typically by a function of the graph's arboricity.

Our first result is to show that an $O(\beta\alpha)$-orientation can be maintained in $O(\lg(\frac{n}{\beta\alpha})/\beta)$ 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 over the entire update sequence. This generalizes previous results by Brodal and Fagerberg [5] and Kowalik [30], which are special cases of our result with $\beta = 1$ and $\beta = \lg n$, respectively.

As an application of this result, we show how to maintain 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 arboricity $\alpha = o(\lg n)$. When α is a constant, which is the case for 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 [41].

Our second result is a new algorithm that maintains a Δ-orientation in worst-case $O(\Delta)$ insertion and deletion time, where $\Delta \leq \min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{2m})$, α is the arboricity of the graph at the time of the update, and m is the number of edges of the graph at the time of the update. Compared with the result of Kopelowitz et al. [29], our algorithm gives a new trade-off with faster insertion time but higher maximum vertex out-degree and slower deletion time when $\alpha = \omega(\lg n)$. In addition, it is simpler than [29] and does not require edge reorientation during insertion. We apply it to maintain a maximal matching in worst-case $O(\min(\alpha \lg(n/\alpha), \sqrt{m}))$ update time using linear space and represent a dynamic graph to support adjacency queries in $O(\lg \lg \Delta)$ worst-case time, edge insertions in $O(\Delta)$ worst-case time, and edge deletions in $O(\Delta \lg \lg \Delta)$ worst-case time.

More results in this thesis include new trade-offs for the coordinate query problem [29] achieved by applying our approaches to maintaining graph orientations, as well as lower and upper bounds on the maximum out-degree guaranteed by a specific algorithm that maintains an edge orientation without changing the directions of edges during updates.

Acknowledgements

First and foremost, I would like to give my sincere gratitude to my supervisors Dr. Meng He and Dr. Norbert Zeh. This work would not have been possible without their patient guidance, inspiration and encouragement.

In addition, I would like to thank my thesis readers Dr. Alex Brodsky and Dr. Michael McAllister for their insightful and detailed comments.

I also would like to give my thanks to NSERC for funding my study and research.

Finally, I would like to thank my family's understanding and support. I would not have started this journey without their love and support.

Chapter 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 [5, 29, 30, 41]. 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. $\overrightarrow{G}$ is called a Δ-*orientation* if the out-degree of each vertex in $\overrightarrow{G}$ is bounded by Δ. The goal is to maintain a Δ-orientation of G with efficient support of edge insertion and deletion, for Δ as small as possible. For dense graphs, Δ has to be large, so 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 *arboricity*, α, of a graph G is defined formally as $\alpha = \max_J \frac{|E(J)|}{|V(J)|-1}$, where $J = (V(J), E(J))$ is any subgraph of G induced by at least two vertices [39]. Many classes of graphs in practice have arboricity bounded by a constant, including planar graphs, graphs of bounded genus, and graphs of bounded tree width. Nash-Williams [39, 40] proved that G has arboricity α if and only if α is the smallest number of subsets that E can be partitioned into so that each subset of edges with their endpoints is a forest. Such a decomposition can be computed in polynomial time [19, 43]. In this partition, if we choose a root for each tree and orient the edges towards the root, then each vertex has out-degree at most one in each tree, which immediately gives an α-orientation of the given *static* graph.

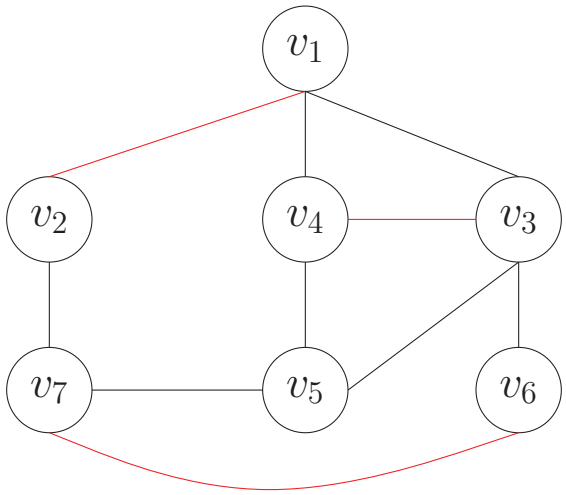


Figure 1.1: An example of graph matching

The most fundamental application of edge orientation is perhaps the representation of graphs supporting adjacency queries. This is based on the following observation [4, 27]: With a Δ-orientation of G, if we store the 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 obtain a linear-space dynamic graph representation that supports adjacency queries in $O(\Delta)$ time [5].

Recently, Neiman and Solomon [41] found that edge orientations also have applications in maintaining maximal matchings of dynamic graphs. A *matching*, M, of a graph G is a set of edges of G that do not share any endpoints. 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. Denote the number of edges of a maximum cardinality matching of G by $\nu(G)$. Then, if a matching of G has at least $\nu(G)/\gamma$ edges, we call it a *γ-approximate maximum cardinality matching*, where $\gamma \geq 1$. Clearly a maximum cardinality matching is a

maximal matching, but a maximal matching is not necessarily a maximum cardinality matching. It is well-known that any maximal matching is a 2-approximate maximum cardinality matching [1]. For example, in Figure 1.1, $M_1 = \{v_1v_2, v_3v_4, v_6v_7\}$ is a maximum cardinality matching of the graph, $M_2 = \{v_1v_3, v_5v_7\}$ is a maximal matching of the graph and is also a 3/2-approximate maximum cardinality matching, and $M_3 = \{v_1v_2, v_3v_4\}$ is a 3/2-approximate maximum cardinality matching but not a maximal matching. Graph matching is a fundamental problem in graph theory, and it has many applications in combinatorial optimization [14, 24, 33, 34, 35]. In the dynamic setting, the problem is to maintain a maximal matching or an approximate maximum cardinality matching under edge insertions and deletions. Recent progress on this problem [22, 29, 41] generated more interest in edge orientations.

Kopelowitz et al. [29] also showed that any solution to the dynamic edge orientation can be applied to the *coordinate query problem*. In this problem, we maintain an $n \times n$ symmetric matrix A and an n-dimensional vector $\overrightarrow{x}$ so that given any integer $i \in [1, \cdots, n]$, $\sum_{j=1}^{n} a_{ij}x_j$ can be computed efficiently. This is called a *coordinate query*. Kopelowitz et al. showed that coordinate queries and updates of $\overrightarrow{x}$ and A can be reduced to queries and updates of an orientation of a graph whose adjacency matrix is A. If a Δ-orientation of the graph can be maintained in t_i insertion time and t_d deletion time, then a coordinate query or an update to an element of $\overrightarrow{x}$ can be performed in $O(\Delta)$ time, an update to an entry in A from a non-zero value to zero can be performed in $O(t_d)$ time, an update to an entry in A from zero to a non-zero value can be performed in $O(t_i)$ time, and an update to an entry in A from a non-zero value to a different non-zero value can be performed in $O(1)$ time.

Edge orientations have also been applied to other problems such as bounded shortest path in planar graphs [29, 32], Steiner forest [15], subgraph listing problems in dynamic planar graphs [9], counting subgraphs [13], dynamic dominance in sparse

graphs [16], graph colouring [31], reporting maximal independent sets [16], computing the girth of a graph [32], and load balancing [6]. Motivated by these applications, we study the problem of orienting dynamic graphs and its applications to maintaining maximal matchings, adjacency queries, and coordinate queries.

1.1 Related Work

1.1.1 Arboricity and Static Graph Orientation

Nash-Williams [39, 40] showed that the exact arboricity of a graph is equal to the smallest number of forests that the graph can be decomposed into. Picard and Queyranne [43] presented the first algorithm to decompose an arbitrary graph into the smallest number of forests in $O(n^2 m \lg^2 n)$ time[1], where n is number of vertices and m is number of edges in the graph. Later, this was improved to $O(\alpha n\sqrt{m + \alpha n \lg n})$ time by Gabow and Westermann [19]. The arboricity of a planar graph is at most 3 since any subgraph with n vertices of a planar graph has at most $3n-6$ edges. Grossi and Lodi [21] proposed an algorithm to decompose a planar graph into three forests in $O(n \lg n)$ time. With the α forests that a graph is decomposed into, we can compute an α-orientation of this graph in linear time as described earlier in this chapter.

Arikati et al. [4] proposed an algorithm to compute a δ-orientation of a graph in $O(n + m)$ time without knowing its arboricity, where $\delta \leq 2\alpha - 1$. This algorithm can thus also be used to compute a 2-approximation of the arboricity of a graph in $O(n + m)$ time.

1.1.2 Dynamic Graph Orientation

Brodal and Fagerberg [5] first studied the problem of maintaining an edge orientation of a dynamic graph with n vertices under an arboricity-α-preserving sequence of

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

edge insertions and deletions. Here, an update operation is considered *arboricity-α-preserving* if, when applied to a graph of arboricity at most α, the arboricity of the graph after the update remains bounded by α. They proposed an approach that can maintain an $O(\alpha)$-orientation in $O(1)$ amortized insertion time, $O(\alpha + \lg n)$ amortized deletion time, and using $O(m + n)$ space, where m is the current number of edges. In their algorithm for update operations, some edges may change their orientations, i.e., be *reoriented*. They proved that the amortized number of edge reorientations achieved by their algorithm is within a constant factor of the amortized number of edge reorientations achieved by the optimal algorithm that knows the entire update sequence beforehand. Later, Kowalik [30] 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. [29] considered the problem of maintaining edge orientations with worst-case time bounds. They proposed two algorithms: The first algorithm maintains an $O(\Delta)$-orientation in $O(\Delta^2)$ worst-case insertion time and $O(\Delta)$ worst-case deletion time, where $\Delta \leq inf_{\beta>1}\{\beta\alpha + \log_\beta n\}$ and α is the arboricity of the graph at the time of the update. The second algorithm improves the insertion time to $O(\beta\alpha\Delta)$, but it requires knowledge of the maximum arboricity during updates beforehand. This restriction can be eliminated by using more complicated algorithms [28]. Both algorithms use $O(n + m)$ space, where n and m denote the number of vertices and edges in the graph, respectively.

1.1.3 Graph Matching

Hopcroft and Karp [25] introduced the currently best algorithm for computing a maximum cardinality matching (MCM) of a bipartite graph. Its running time is $O(m\sqrt{n})$. Later, it was extended to general graphs by Micali and Vazirani [36] with the same time complexity. To support updates to the graph, Tarjan [45] introduced

an algorithm to maintain a MCM in $O(m)$ worst-case time per update. Sankowski [44] introduced a randomized algorithm with $O(n^{1.495})$ update time which is more efficient for dense graphs. For trees, Gupta and Sharma [23] proposed an algorithm with $O(\log n)$ worst-case update time.

In a weighted graph, a *maximum weight matching* is a matching with the maximum total weight of its edges. Algorithms with time complexity $O(m\sqrt{n \log n} \log(nN))$ and $O(m\sqrt{n} \log(nN))$ to compute a maximum weight matching (MWM) were proposed by Gabow and Tarjan [20] and Duan et al. [12], respectively, where N is the maximum weight of the edges in the graph.

As the results above demonstrate, a MCM or MWM is expensive to compute. However, an approximate MCM or MWM can be computed efficiently. Hopcroft and Karp [25] and Micali and Vazirani [36] presented algorithms that can compute a $(1+\epsilon)$-approximate MCM in $O(m\epsilon^{-1})$ time, for any $\epsilon < 1$. A $(1 + \epsilon)$-approximate MWM can be computed in $O(m\epsilon^{-1} \log \epsilon^{-1})$ time using an algorithm by Duan et al. [12]. To maintain an approximate MCM in dynamic graphs, Neiman and Solomon [41] introduced a solution to maintain a maximal matching of a graph, which is also a 3/2-approximate MCM of the graph, using $O(\sqrt{m})$ worst-case update time. This is the first approach that can maintain a maximal matching in $o(m)$ worst-case update time. Later, Gupta and Peng [22] proposed an algorithm to maintain a $(1 + \epsilon)$-approximate MCM using $O(\sqrt{m}\epsilon^{-2})$ worst-case update time. The matching that they maintain is not necessarily a maximal matching since their algorithm uses the strategy of recomputing an MCM after a certain number of updates to guarantee the approximation ratio, and the updates performed before each recomputation may cause the matching to become non-maximal. They also extended their result to maintain an approximate MWM in a weighted graph where edges have weights between $[1,N]$ in $O(\sqrt{m} \log N)$ worst-case update time.

To support more efficient updates for graphs with low arboricity, Neiman and

Solomon [41] showed how to use graph orientation to maintain a maximal matching. Their approach can maintain a maximal matching in $O(m+n)$ space and supports updates 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. [29] made use of their solution for graph orientation to maintain a maximal matching in $O(\beta\alpha\Delta)$ worst-case insertion time and $O(\Delta)$ worst-case deletion time, where $\Delta \leq inf_{\beta>1}\{\beta\alpha + \log_\beta n\}$.

1.1.4 Adjacency Queries

As mentioned in Chapter 1 paragraph 3, solutions to maintaining an edge orientation on dynamic graphs can be used directly to support adjacency queries. Kowalik [30] showed that by maintaining the list of the out-neighbours of each vertex using the dynamic dictionary of Andersson and Thorup [3], a graph can be represented in $O(m+n)$ space and support adjacency queries and edge deletions in $O(\lg\lg\lg n)$ worst-case time, and edge insertions in $O(\lg\lg\lg n)$ amortized time, provided that $\alpha = O(\text{polylog}(n))$. Using the same strategy, Kopelowitz et al. [29] presented a linear-space representation of graphs with arboricity $\alpha = \text{polylog}(n)$ that supports adjacency queries in $O(\lg\lg\Delta)$ worst-case time, edge insertions in $O(\beta\alpha\Delta\lg\lg\Delta)$ worst-case time, and edge deletions in $O(\Delta\lg\lg\Delta)$ worst-case time, where $\Delta \leq \inf_{\beta>1}\{\beta\alpha + \lceil\log_\beta n\rceil\}$.

1.1.5 Coordinate Queries

Let A be an $n\times n$ symmetric matrix and $\overrightarrow{x}$ be an n-dimensional vector. Kopelowitz et al. [29] provided a solution to answer coordinate queries on them in $O(\Delta)$ worst-case time, where $\Delta = inf_{\beta>1}\{\beta\alpha + \log_\beta n\}$ and α is the arboricity of the graph G whose adjacency matrix is A, by using their solution to maintain an edge orientation [29] in their reduction for the coordinate query problem [29] as mentioned in Chapter 1

paragraph 5. In this solution, an update to an entry in A from zero to a non-zero value can be performed in $O(\alpha\beta\Delta)$ worst-case time, an update to an entry in A from a non-zero value to zero and an update to an element of $\overrightarrow{x}$ can be performed in $O(\Delta)$ worst-case time, and an update to an entry in A from a non-zero value to a different non-zero value can be performed in $O(1)$ worst-case time.

By using Brodal and Fagerberg's and Kowalik's solutions to edge orientation [5, 30] in Kopelowitz et al.'s reduction to the coordinate query problem [29], more results can be obtained. Here we consider the total time required to answer a sequence of requests, which also gives a bound on the amortized cost of each operation. In a request sequence, let n_q denote the number of coordinate queries, n_x denote the number of the updates to the elements of $\overrightarrow{x}$, n_i denote the number of the updates to the entries in A from zero to non-zero values, n_d denote the number of the updates to the entries in A from non-zero values to zero, and n_o denote the number of the updates to the entries in A from non-zero values to different non-zero values. Then, if we use Brodal and Fagerberg's solution [5] to maintain an edge orientation in the reduction, the total time to answer the request sequence is $O(n_i\cdot(\alpha+\log n)+(n_x+n_q+n_d)\cdot\alpha+n_o)$, where α is the maximum arboricity of the graph G whose adjacency matrix is A. If we use Kowalik's solution [30] to maintain an edge orientation in the reduction, then the total time to answer the request sequence is $O(n_i + (n_x + n_q + n_d)\cdot\alpha\log n + n_o)$.

1.2 Our Contributions

First, we provide a new analysis of the algorithm of Brodal and Fagerberg [5] for graphs with arbitrary arboricity, by constructing a new offline algorithm for the edge orientation problem. Our new analysis shows that an $O(\beta\alpha)$-orientation 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 reorientation is required when performing an edge deletion. This presents a trade-off 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 insertions in $O(\lg n)$ amortized time and deletions in $O(\alpha)$ worst-case time. This is comparable to Brodal and Fagerberg's original 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 reorientations for each deletion, which matches Kowalik's analysis [30].[2] When $\beta = \sqrt{\frac{\lg n}{\alpha}}$, this algorithm maintains an $O(\sqrt{\alpha \lg n})$-orientation with $O(\sqrt{\alpha \lg n})$ amortized insertion time and $O(\sqrt{\alpha \lg n})$ worst-case deletion time. This trade-off can not be shown using previous analyses.

Second, we 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 and $O(\alpha + \sqrt{\alpha \lg n})$ 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 [41] when $\alpha = \Omega(\lg n)$, while it 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 $O(\frac{\lg n}{\sqrt{f(n)}})$ time. The improvement is most significant 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 [41]. In addition, we apply our result on edge orientation to improve the previous results on answering

[2]Kowalik's analysis [30] of the 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 [5], which maintains out-neighbours in linked lists, and the time required to search each list for the edge to be deleted is part of the deletion time. Thus, when comparing with Kowalik's analysis, we consider the number of reorientations.

a sequence of requests in the coordinate query problem.

Third, we 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 $\Delta \leq \min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{2m})$, α is the arboricity of the graph at the time of the update, and m is the number of edges of the graph at the time of the update. This is a new trade-off when compared with the result of Kopelowitz et al. [29]: When $\alpha = \omega(\lg n)$, our insertion time is $O(\alpha \lg n)$, which is better than their $O(\alpha^2 + \alpha \lg n)$ insertion time. However, our maximum out-degree and deletion time are worse. 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 insertions in $O(\Delta)$ worst-case time, and edge deletions in $O(\Delta \lg \lg \Delta)$ worst-case time. For graphs with constant arboricity such as planar graphs, our representation supports adjacency queries, insertions and deletions 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 support for queries and deletions, but requires $O(\lg n \lg \lg \lg n)$ time for insertions. The fact that our insertion algorithm for edge orientation does not require reorientations makes such an improvement possible. For non-constant α, our result is a new trade-off: our insertion is faster by a factor of $\lg \lg \lg n$ than the result of Kopelowitz et al. but deletions may be slower by a factor of α. All our solutions use $O(n + m)$ space. Again, we apply our solution to maintaining an edge orientation with worst-case time bounds to the coordinate query problem to get similar tradeoffs.

Finally, we prove lower and upper bounds on the maximum out-degree, Δ, of any node in an oriented graph when using a simple online algorithm to handle updates. The algorithm supports edge insertions and edge deletions without requiring edge

	Insertion-Only Updates	At most $\alpha(n-1)$ Updates	Arbitrary Updates
Lower	$\Omega(\lg n)$	$\Omega(\min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\})$, for any $\alpha \geq 3$	$\Omega(\sqrt{n})$, for any $\alpha \geq 3$
Upper	$O(\min(\alpha \lg(n/\alpha), \sqrt{m}))$	$O(\sqrt{\alpha(n-1)})$	$O((\alpha n)^{2/3})$

Table 1.1: Summary of lower and upper bounds on maximum out-degree without reorientation

reorientations. It handles an edge insertion by orienting the newly inserted edge from the endpoint with lower out-degree to the endpoint with higher out-degree or orienting the edge arbitrarily if both endpoints have equal out-degree. It handles an edge deletion by simply deleting the edge. We show that $\Delta = O(\min(\alpha \lg(n/\alpha), \sqrt{m}))$ after performing an update sequence that contains edge insertions only, where α is the maximum arboricity of the graph throughout the update sequence and m is the length of the update sequence. However, if deletions are allowed, some update sequences of length at most $\alpha(n-1)$ may force a vertex to have out-degree $\Omega(\min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\})$. We summarize our results in Table 1.1. The row "Lower" presents lower bounds on Δ. The row "Upper" presents upper bounds on Δ. The column "Insertion-Only Updates" presents lower and upper bounds on Δ after performing an update sequence that contains insertions only. The column "At most $\alpha(n-1)$ Updates" presents lower and upper bounds on Δ after performing an update sequence that contains at most $\alpha(n-1)$ updates including insertions and deletions. The column "Arbitrary Updates" presents lower and upper bounds on Δ after performing an arbitrary update sequence.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we present our solution with amortized update bounds and its application to maintaining maximal matchings and coordinate queries. In Chapter 3, we present our solution with worst-case update bounds and its application to maintaining maximal matchings, adjacency queries,

and coordinate queries. In Chapter 4, we prove the lower and upper bounds of the maximum out-degree of a vertex in an orientation maintained using an algorithm that does not require edge reorientations during updates. In Chapter 5, we present concluding remarks and list some open problems.

Chapter 2

Dynamic Edge Orientation with Amortized Update Time

In this chapter, we present a solution to maintaining an edge orientation with amortized time bounds. Brodal and Fagerberg [5] 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 such that the total number of edge reorientations is minimized. We use the same strategy in our solution. We first recall their reduction in Section 2.1. Then, in Section 2.2, we present a new offline algorithm to orient dynamic graphs and use Brodal and Fagerberg's reduction to obtain our solution to maintaining an edge orientation undertake online updates. In Section 2.3, we present applications of our solution to maximal matching and coordinate queries, respectively.

2.1 Reduction from Online Orientations to Offline Orientations

Brodal and Fagerberg [5] proved the following lemma that reduces online graph orientation to the offline version of the same problem.

Lemma 1 ([5]) *There exists an online reorientation algorithm A with the following property:*

Let $G_0, G_1, \ldots, G_q$ be the sequence of graphs produced by an arboricity-α-preserving sequence of updates, where G_0 is the empty graph and G_i is the graph obtained after the first i updates. Further assume that the update sequence contains k insertions. If there exists a δ-orientation sequence $\overrightarrow{G}_0, \overrightarrow{G}_1, \ldots, \overrightarrow{G}_q$ that involves at most kr edge

reorientations, then A can maintain a Δ-orientation under this update sequence using $O(n+m)$ space, $O(\frac{r\Delta}{\Delta+1-2\delta})$ amortized time per insertion, and $O(\Delta)$ worst-case time per deletion, provided $\Delta \geq 2\delta > 2\alpha$.

We will use this lemma in our own solution in the same way that Brodal and Fagerberg [5] and Kowalik [30] did. Brodal and Fagerberg presented an offline algorithm to maintain a δ-orientation of a graph of arboricity α, where $\delta > \alpha$, with no edge reorientations for insertions and at most $\lceil \log_{\delta/\alpha} n \rceil$ edge reorientations per deletion. Kowalik presented an offline algorithm to maintain a $(4\alpha(\lfloor \log \alpha n \rfloor + 1))$-orientation of a graph of arboricity α with at most $2(k + 2k\alpha/\beta)$ edge reorientations, where k is number of insertions in the update sequence and β is an integer in $[1, +\infty]$. Combined with Lemma 1, their results on online graph orientation follow.

2.2 A Smooth Trade-off on Maintaining Edge Orientations

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 show how to obtain 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 divide U into *phases* containing $\beta\alpha n$ consecutive update operations each for some parameters $\beta \geq 1$, except the last phase, which may contain fewer operations. For simplicity, we assume that $\beta\alpha n$ is an integer. For the graph at the end of each *full phase* containing $\beta\alpha n$ operations, we compute an α-orientation using the approach mentioned in Section 1.1.1, which makes use of the algorithm in [19, 43]. 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 orient each graph G_i where i is not divisible by $\beta\alpha n$, we need the following definition:

Definition 1 *Consider a phase, P, of $|P| \leq \beta\alpha n$ consecutive updates on a graph G with n vertices. 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*, $H_P(G)$, of G in P is the subgraph of G induced by all the hot vertices of G in P, while the* cold region*, $C_P(G)$, of G is defined to be $G \setminus H_P(G)$.*

The δ-orientation sequence is determined recursively. We use the following strategy for each phase, P, of U. 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 an α-orientation of G. At the current level of recursion, we only determine the orientations of edges in graphs G_{i+j} with $j \in [1..|P|-1]$ if $|P| = \beta\alpha n$ or $j \in [1..|P|]$ if $|P| < \beta\alpha n$ that have at least one cold endpoint. Edges with only hot endpoints are oriented recursively. We consider the graphs G_{i+j} in increasing order of j: For an edge that is present in both G_{i+j-1} and G_{i+j}, 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, 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.

To recursively orient the edges between hot vertices in phase P, 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

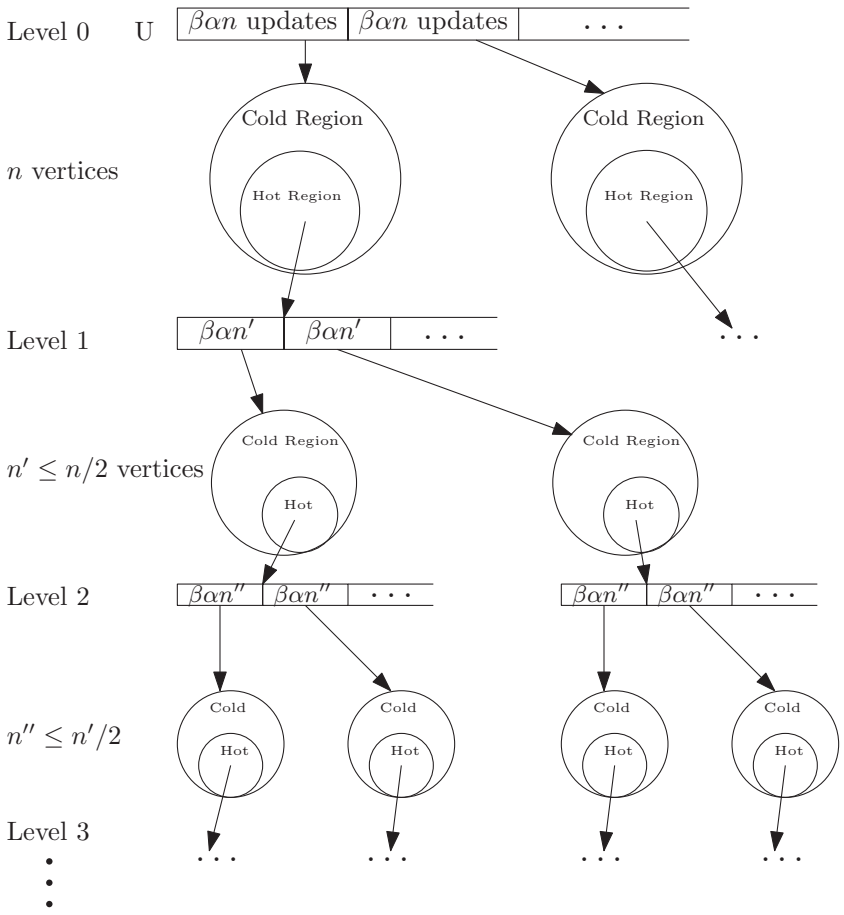


Figure 2.1: Offline recursive strategy

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 $2\beta\alpha n' \leq |P| \leq \beta\alpha n$, 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 if the phase is full (recall that after the last operation of a full phase, the graph is oriented by computing an α-orientation, so we exclude the last operation in full phase 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 $H_P(G)$. Upon returning from the recursion on $H_P(G)$, the direction of each edge inserted between hot vertices has been decided, as it is part of the graph $H_P(G)$. Thus, we have oriented all edges of all the G_is. Figure 2.1 illustrates our recursive strategy.

We now prove the following upper bound on vertex out-degrees:

Lemma 2 *The offline algorithm in this section computes a sequence of* $(4\beta\alpha + \alpha)$*-orientations* $\overrightarrow{G}_0, \overrightarrow{G}_1, \ldots, \overrightarrow{G}_q$.

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. We first consider an arbitrary cold vertex x in this phase. Before any operation in P is performed, in $\overrightarrow{G}_i$, the out-degree of x is at most α as $\overrightarrow{G}_i$ is computed as an α-orientation. By Definition 1, fewer 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 $\alpha + 4\beta\alpha$.

Now consider 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 $H_P(G)$, which is bounded by $4\beta\alpha + \alpha$ by the inductive hypothesis. □

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

Lemma 3 *The total number of edge reorientations in the orientation sequence* $\overrightarrow{G}_0, \overrightarrow{G}_1, \ldots, \overrightarrow{G}_{|U|}$ *is* $O(\frac{|U|}{\beta} \cdot \lg(n/(\beta\alpha)))$.

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 $\alpha(n/2^i - 1)$ by the definition of arboricity.

Note that at any given level, reorientation only happens at the end of each full phase defined for a subgraph at that level, when we recompute an α-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 full 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 that this is true, 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 containing $\beta\alpha t$ operations each, except the last phase, which may contain fewer. Edge reorientations take place at the end of each full 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 for at most $\alpha(t-1)/(\beta\alpha t) < 1/\beta$ edge reorientations. Since each operation in U is considered at most once at each level as mentioned above, each level is charged for at most $\frac{|U|}{\beta}$ edge reorientations in total. Thus, with $O(\lg(n/(\beta\alpha)))$ levels, the total number of edge reorientations is $O(\frac{|U|}{\beta} \cdot \lg(n/(\beta\alpha)))$. □

Combining Lemmas 2 and 3, we have:

Lemma 4 *Given an arboricity-α-preserving sequence of edge insertions and deletions on an initially empty graph with n vertices and an arbitrary parameter $\beta \geq 1$, there exists a sequence of $(4\beta\alpha + \alpha)$-orientations such that the amortized number of edge*

reorientation for each edge insertion or deletion is $O(\frac{\lg(n/(\beta\alpha))}{\beta})$.

Combining Lemma 4 with Lemma 1, we have our first main result:

Theorem 1 *There exists an online algorithm for maintaining a* Δ*-orientation of an initially empty graph on* n *vertices under arboricity-*α*-preserving updates that uses* $O(n+m)$ *space and supports edge insertions in* $O(\frac{\lg(n/(\beta\alpha))}{\beta} \cdot \frac{\Delta}{\Delta+1-2\delta})$ *amortized time and edge deletions in* $O(\Delta)$ *worst-case time, provided* $\delta = (4\beta+1)\alpha$*, for an arbitrary parameter* $\beta \geq 1$*, and* $\Delta \geq 2\delta$*. Edge deletions do not incur any edge reorientations.*

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 4 shows that the total number of edge reorientations is $O(\frac{(k+k')\lg(n/(\beta\alpha))}{\beta}) = O(k \cdot (\frac{\lg(n/(\beta\alpha))}{\beta}))$. The theorem now follows from Lemma 1 by defining $r = O(\frac{\lg(n/(\beta\alpha))}{\beta})$. □

The tradeoffs summarized in Section 1.2 are obtained by setting $\Delta = 3\delta$ in Theorem 1. By setting $\Delta = 3\delta$, we get $\frac{\Delta}{\Delta+1-2\delta} = \frac{3\delta}{\delta+1} \leq 3$ and $O(\frac{\lg(n/(\beta\alpha))}{\beta})$ amortized edge insertion time.

2.3 Applications

2.3.1 Maintaining Maximal Matchings

Neiman and Solomon [41] proposed an amortized algorithm to maintain maximal matchings by making use of Brodal and Fagerberg's solution [5] for edge orientation.

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. Thus, the following result follows from Theorem 1:

Theorem 2 *There exists an online algorithm for maintaining a maximal matching of a graph on n vertices under arboricity-α-preserving updates that uses $O(n+m)$ space and supports updates in $T = O(\alpha + \sqrt{\alpha \lg n})$ amortized time.*

Proof. We first observe that combining Theorem 1 with parameter $\Delta = 3\delta$ with Neiman and Solomon's reduction mentioned in Section 1.1.3, a maximal matching can be maintained in $O(\beta\alpha + \frac{\lg(n/(\beta\alpha))}{\beta})$ amortized update time and using $O(n+m)$ space. When $\alpha \geq \lg n$, we set $\beta = 1$ and the update time becomes $O(\alpha)$. Otherwise, we set set $\beta = \sqrt{\frac{\lg n}{\alpha}}$, and the update time becomes $O(\sqrt{\alpha \lg n})$. This completes this proof. □

2.3.2 Coordinate Queries

Using Theorem 1 in Kopelowitz et al.'s reduction for the coordinate query problem [29] mentioned in Chapter 1, we have the following theorem:

Theorem 3 *Let A be an $n \times n$ symmetric matrix and $\overrightarrow{x}$ be an n-dimensional vector. Consider a sequence of requests, consisting of n_q coordinate queries, n_x updates to the elements of $\overrightarrow{x}$, n_d updates to the entries in A from non-zero values to zero, n_i updates to the entries in A from zero to non-zero values, and n_o updates to the entries in A from non-zero values to different non-zero values. All these requests can be processed in $O(\sqrt{n_i(n_x+n_q+n_d)\alpha \lg n} + n_o)$ time in total when $n_x + n_q + n_d < n_i \frac{\lg n}{\alpha}$ or in $O(n_i \lg n + (n_x+n_q+n_d)\alpha + n_o)$ time in total when $n_x + n_q + n_d \geq n_i \frac{\lg n}{\alpha}$, where α is the maximum arboricity of the graph G whose adjacency matrix is A, provided that n_i, n_x, n_q, n_d and n_o are known in advance.*

Proof. By Theorem 1 and Kopelowitz et al.'s reduction, the total time to process the request sequence is $T = O(n_i \cdot \frac{\lg(n/(\beta\alpha))}{\beta} + (n_x+n_q+n_d)\cdot(\beta\alpha) + n_o) = O(n_i \cdot \frac{\lg n}{\beta} + (n_x+n_q+n_d)\cdot(\beta\alpha) + n_o)$. If $n_x + n_q + n_d < n_i \frac{\lg n}{\alpha}$, then $\sqrt{\frac{n_i \lg n}{(n_x+n_q+n_d)\alpha}} > 1$. By choosing $\beta = \sqrt{\frac{n_i \lg n}{(n_x+n_q+n_d)\alpha}} \geq 1$, we get $T = O(\sqrt{n_i(n_x+n_q+n_d)\alpha \lg n} + n_o)$. If

$n_x + n_q + n_d \geq n_i \frac{\lg n}{\alpha}$, we get $T = O(n_i \lg n + (n_x + n_q + n_d)\alpha + n_o)$ by choosing $\beta = 1$. This completes the proof. □

It is obvious that Theorem 3 provides a more efficient solution when $n_x + n_q + n_d < n_i \frac{\lg n}{\alpha}$ compared to the previous results described in section 1.1.5, provided that n_i, n_x, n_q, n_d and n_o are known in advance.

Chapter 3

Dynamic Edge Orientation with Worst-Case Update Time

In this chapter, we present a new solution to maintaining edge orientations under edge insertions and deletions with worst-case time bounds. In Section 3.1, we first discuss a data structure introduced by Kopelowitz et al. [29], which is also used in our solution. Then we present our new graph orientation algorithm in Section 3.2 and some of its applications in Section 3.3. In the rest of this thesis, $d_o(u)$ denotes the out-degree of vertex u unless stated otherwise.

3.1 Preliminaries

Kopelowitz et al. [29] defined the following data structure problem to help them maintain a number of invariants in their work, and we will also make use of this data structure in our solution: Let X be a dynamic set, where 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 cannot 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): Increment the key of an element $x \in X \setminus \{x_0\}$ by 1, given a pointer to x;
- `Decrement`(X, x): Decrement the key of an element $x \in X \setminus \{x_0\}$ by 1, given a pointer to x;

- `Insert`(X, x, k): Insert a new element x with key k into X, provided $k \leq k_0 + 1$;
- `Delete`(X, x): Delete an element $x \in X \setminus \{x_0\}$ from X, given a pointer to x;
- `IncrementCenter`(X): Increment k_0 by 1;
- `DecrementCenter`(X): Decrement k_0 by 1.

The following lemma summarizes the solution to this problem presented by Kopelowitz et al. [29].

Lemma 5 ([29]) *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.*

3.2 Maintaining Edge Orientations

3.2.1 A New Invariant

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 all $i = 0, 1, \ldots, d_o(u) - 1$.*

There are connections between this invariant and the two invariants considered by Kopelowitz et al. [29], but they are different invariants. Their invariants require each vertex u has at least $\min\{d_o(u), \beta\alpha\}$ out-neighbours have out-degree at least $d_o(u) - 1$, for any chosen $\beta > 1$. Our invariant provides a weaker bound on the maximum out-degree of the vertices in the graph. 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. 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 $d_o(v_{\Delta-1}) \geq \Delta - 1, d_o(v_{\Delta-2}) \geq \Delta - 2, \ldots, d_o(v_{\Delta-2\alpha}) \geq \Delta - 2\alpha$ by Invariant 1, which means u has at least 2α out-neighbours with out-degrees greater than or equal to $\Delta - 2\alpha$.

Now assume the claim holds for $k - 1$. We prove it for k. By the inductive hypothesis, there is a set, V_1, of $2^{k-1}\alpha$ vertices with out-degrees 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 these out-neighbours of each vertex in V_1 to another set V_2. Note that some vertices in V_1 may share out-neighbors. Since any vertex in $V_1 \cup V_2$ has out-degree at least $\Delta - 2k\alpha$, it remains 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 its out-neighbours in V_2. Thus, the number of edges in G^* is at least $2\alpha|V_1| = 2^k\alpha^2$. As G has arboricity α and G^* is a subgraph of G, we have $\alpha \geq \frac{|E(G^*)|}{|V(G^*)|-1} \geq \frac{2^k\alpha^2}{|V_1 \cup V_2|-1}$. Therefore, $|V_1 \cup V_2| \geq 2^k\alpha$ and the claim holds also for k. □

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-degree 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 at least $2^{\lfloor \Delta/(2\alpha) \rfloor - 1}\alpha$. This implies that, 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 $(2^{(\Delta/(2\alpha))-1}\alpha^2)/(n-1) < \alpha$. Solving for Δ in this inequality yields $\Delta < 2\alpha \lg \frac{n-1}{\alpha} + 2\alpha$. This completes the proof. □

3.2.2 Auxiliary Data Structures

To maintain Invariant 1, we borrow ideas from [29] but our algorithms for edge insertion and deletion turn out to be simpler. As in [29], 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. $B_u \setminus \{u\}$ 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 represent B_u using Lemma 5. 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 outgoing edges of each vertex in a doubly linked list. This also requires $O(m+n)$ space. We store a pointer to u's representation in B_v with edge (u, v) and vice a versa. 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 [29] work, but it was not mentioned explicitly. As it is trivial to maintain the adjacency lists with these pointers and the maintainence cost is subsumed by our final time bounds, we do not explicitly discuss how to update these lists in the rest of this section.

3.2.3 Handling Edge Insertions

To insert an edge (u, v), we orient it from u to v if $d_o(u) \leq d_o(v)$. Otherwise, it is oriented from v to u. We then update the auxiliary data structures associated with the vertex that is the tail of the newly inserted edge, as well as its out-neighbors.

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, y, of u such that $y \neq v$ **do**
6: `Increment`(B_y, u)
7: **end for**

Algorithm 1 presents the pseudo code.

To see that our algorithm for edge insertion maintains Invariant 1, assume without loss of generality that the newly inserted edge is oriented from u to v by our algorithm. Let G_{i-1} and G_i denote the graphs before and after this edge insertion, respectively, and let $d_o(x, j)$ denote the out-degree of vertex x in graph G_j. Assume inductively that the invariant is maintained in G_{i-1} (the base case is a graph with no edges, so the invariant holds trivially). Orienting the edge from u to v can only violate the invariant of u and in-neighbours of u since other vertices and their out-neighbours' out-degrees are not changed. The increase of the out-degree of an out-neighbour of a vertex will not violate the invariant of the vertex, so the invariant is still maintained for the in-neighbours of u. To argue that the invariant is also maintained for u, let $u_0, u_1, \ldots, u_{d_o(u,i-1)}$ be the out-neighbours of u in G_{i-1} listed in the order specified in the invariant. After insertion, v also became an out-neighbour of u, and we consider the list $u_0, u_1, \ldots, u_{d_o(u,i-1)}, v$, which contains all the out-neighbours of u in G_i. By our algorithm, we have $d_o(u, i) = d_o(u, i-1) + 1$ and $d_o(v, i) = d_o(v, i-1)$. As $d_o(v, i-1) \geq d_o(u, i-1)$, we have $d_o(v, i) \geq d_o(u, i) - 1$. Furthermore, the out-degrees of all other out-neighbours of u remain unchanged after insertion. Thus, the invariant is maintained in G_i. It is easy to see that all the auxiliary data structures are updated correctly after we orient u to v.

To see the time complexity of Algorithm 1, we analyze it step by step. Line 2 takes $O(1)$ time with the auxiliary data structures described in Section 3.2.2. Line

3 takes $O(d_o(u))$ time, by Lemma 5, since we use $d_o(u)$ as the key of the center element of B_u. Line 4 takes $O(1)$ time, by Lemma 5. Each iteration of the for loop in line 5 takes $O(1)$ time since line 6 takes $O(1)$ time by Lemma 5. In addition, the number of iterations of the for loop is bounded by $d_o(u)$. Therefore, lines 3 and 5–7 take $O(d_o(u))$ time. $d_o(u)$ is bounded by Δ, by Lemma 7 and because the algorithm maintains Invariant 1. Thus, in total, Algorithm 1 takes $O(\Delta)$ time.

By the above analysis, we have the following lemma:

Lemma 8 *If a directed n-vertex graph G of arboricity α satisfies Invariant 1, then Algorithm 1 can handle an edge insertion into G in $O(2\alpha \lg(n/\alpha) + 2\alpha)$ worst-case time. The graph G satisfies Invariant 1 after the insertion.*

3.2.4 Handling Edge Deletions

Algorithm 2 presents the pseudocode of edge deletion. We first remove the edge to be deleted in lines 2–3. After the edge deletion, the out-degree of u is decreased by 1 and the out-degrees of all out-neighbours of u remain unchanged, so the invariant is hold for u. 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, x, with the largest out-degree in line 4. If the test in the while statement at line 5 is false, then any in-neighbour's out-degree is at most one greater than $d_o(u)$ since x is the in-neighbour with the maximum out-degree. In this case, the invariant still holds for any in-neighbour of u. If the result of the test condition in the while statement is true, then it is possible (though not necessary) that the invariant is not maintained for x and some other in-neighbours of u. To maintain the invariants for these vertices, we reverse the direction of the edge (x, 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

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

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

is performed, and thus the invariant cannot be violated for any of its in-neighbours whose out-degree did not change. The invariant also holds for vertex u since the only new out-neighbour, x, of u has out-degree at least equal to $d_o(u)$ after reorientation and the other out-neighbours of u are not changed. The only in-neighbour whose out-degree has been changed is x, and it is easy to see that the invariant is also maintained for x as a result of the above steps: x lost one out-neighbour but its out-degree was also decreased by 1. Now the only vertices for which Invariant 1 may not hold have to be in-neighbours of x. For x, we then repeat the same process that we applied to u. It is clear that if the while loops eventually terminates, then after the while loop in lines 5–11, the invariant is maintained, and lines 12–15 make sure that all the auxiliary structures are up-to-date.

We have the following lemma showing that this while loop always terminates, which further implies the correctness of our edge deletion algorithm:

Lemma 9 *When performing an edge deletion on a Δ-orientation of G, the while loop in Algorithms 2 iterates at most $\Delta + 1$ times.*

Proof. By line 5 of Algorithm 2, at the end of each iteration of the while loop, u is set to be a vertex whose out-degree is strictly larger than the original out-degree of u before this edge deletion operation is performed. As the out-degree of a vertex is in the interval $[0, \Delta]$, this loop is iterated at most $\Delta + 1$ times. □

The time complexity of Algorithm 2 is dominated by the cost of the while loop in lines 5–11 and the for loop in lines 13–15, as well as the time required by the `DecrementCenter` operation in line 12. Each iteration of each of the while loops takes constant time, so the two while loops take $O(\Delta)$ time by Lemma 9. In addition, line 12 takes at most Δ time by Lemma 5. Thus, the algorithm takes $O(\Delta)$ time.

By Algorithm 2, Lemma 9 and the above analysis, we have Lemma 10.

Lemma 10 *If a directed n-vertex graph G of arboricity α satisfies Invariant 1, then Algorithm 2 can handle an edge deletion in G in $O(\Delta)$ worst-case time, where $\Delta < 2\alpha \lg(n/\alpha) + 2\alpha$. The graph G satisfies Invariant 1 after the deletion.*

3.2.5 Bounding Δ by $\sqrt{2m}$

We now further bound the maximum vertex out-degree, Δ, in an arbitrary directed graph that satisfies Invariant 1 by a function of the number of edges in the graph.

Lemma 11 *If a directed graph G satisfies Invariant 1, then the out-degree of any vertex in G is at most $\sqrt{2m}$, where m is the number of edges in G.*

Proof. We give a proof by contradiction. Assume to the contrary that there is a vertex u with out-degree $d_o(u) > \sqrt{2m}$ in a graph G that satisfies Invariant 1. Then vertex u must have out-neighbours $v_0, v_1, \ldots, v_{\sqrt{2m}}$, such that $d_o(v_i) \geq i$, for $i = 0, 1, \ldots, \sqrt{2m}$. The outgoing edges of all v_i's and u are distinct. Therefore, the total number of edges in the graph is at least $1 + 2 + \ldots + \sqrt{2m} + (\sqrt{2m} + 1) = \frac{1}{2}(\sqrt{2m} + 1)(\sqrt{2m} + 2) > m$, which contradicts the fact that G has m edges. □

Combining Lemmas 7, 8, 10, and 11, we have the following theorem:

Theorem 4 *A Δ-orientation of a graph on n vertices can be maintained in $O(n+m)$ space so that an edge insertion or deletion can be performed in $O(\Delta)$ worst-case time, where $\Delta \leq \min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{2m})$, α is the arboricity of the graph at the time of the update, and m is the number of edges of the graph at the time of the update. Furthermore, an edge insertion does not incur edge reorientation, while a deletion incurs at most $\Delta + 1$ reorientations.*

3.3 Applications

3.3.1 A New Algorithm to Maintain Dynamic Maximal Matching with Worst-Case Time Bounds

As in Kopelowitz et al. [29], we design our solutions to maintaining a maximal matching with worst-case time bounds by applying our result on maintaining edge orientation, again with worst-case bounds, to the approach of Neiman and Solomon [41]. The other data structures used in [41] can also be replaced by data structures with constant worst-case time bounds[1].

Theorem 4 can then be applied to achieve new results on maximal matchings:

Theorem 5 *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 arboricity of the graph at the time of the update, and m is the number of edges of the graph at the time of the update.*

[1]These details are not included in [29], but the reason why they are omitted is that it is simple to come up with these date structures, as confirmed in personal communication with Tsvi Kopelowitz [28].

3.3.2 Adjacency Query Data Structure with Worst-case Time Bounds

Kowalik [30] showed that in a solution that maintains a Δ-orientation of a graph of arboricity α for any $\Delta = O(\alpha \operatorname{polylog}(n))$, if we maintain the set of outgoing edges of each vertex using the dynamic dictionary of Andersson and Thorup [3], adjacency queries can be supported in $O(\lg\lg\Delta)$ time, provided that $\alpha = O(\operatorname{polylog}(n))$. The time of each edge insertion or deletion is then the sum of the time required to perform this operation to maintain a Δ-orientation and $O(t\lg\lg\Delta)$, where t is the number of update performed to these dictionaries. Using Theorem 4 in this reduction, we have the following theorem:

Theorem 6 *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 $\Delta = O(\min(\alpha\lg(n/\alpha), \sqrt{m}))$ and α is the arboricity of the graph at the time of the update, provided $\alpha = O(\operatorname{polylog}(n))$.*

Proof. To maintain the dictionaries mentioned in above reduction during updates, it suffices to perform at most two update operations to them each time we insert, delete or reorient an edge. Thus, if we apply our solution from Theorem 4, we update these dictionaries $O(1)$ and $O(\Delta)$ times when we perform an edge insertion or deletion, respectively. Therefore, the time for an edge insertion is $O(\Delta + \lg\lg\Delta) = O(\Delta)$, while an edge deletion takes $O(\Delta\lg\lg\Delta)$ time. □

3.3.3 Coordinate Queries

Using Theorem 4 in Kopelowitz et al.'s reduction for the coordinate query problem [29] mentioned in Chapter 1, we have the following corollary:

Corollary 1 *Let A denote an $n \times n$ symmetric matrix and $\overrightarrow{x}$ denote an n-dimensional vector. A and $\overrightarrow{x}$ can be maintained in $O(m+n)$ space to answer coordinate queries in $O(\Delta)$ worst-case time, where $\Delta \leq \min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{2m})$, α is the arboricity of the graph G whose adjacency matrix is A, and m is the number of the edges in G. Furthermore, an update to an entry in A from a non-zero value to zero, an update to an entry in A from zero to a non-zero value, and an update to an element of $\overrightarrow{x}$ can be performed in $O(\Delta)$ worst-case time, and an update to an entry in A from a non-zero value to a different non-zero value can be performed in $O(1)$ worst-case time.*

Comparing our solution to Kopelowitz et al.'s solution [29] to the coordinate query problem, our solution provides more efficient support for the updates to the entries in A from zero to non-zero values when $\alpha = \omega(\lg n)$.

Chapter 4

Maintaining Graph Orientations without Edge Reorientation

We have designed algorithms to orient dynamic graphs with amortized and worst-case time bounds in previous chapters. Edge reorientation is commonly used in these algorithms to bound the maximum out-degree, Δ, in a dynamic setting. A tight bound on the number of reorientations to maintain a Δ-orientation is still not known. Brodal and Fagerberg showed that their algorithm achieves a number of edge reorientations that is within a constant factor of the optimal bound, but they did not prove a tight bound on the optimal number of edge reorientations.

In this chapter we consider a related problem. We are interested in finding lower and upper bounds on Δ for an algorithm that does not perform any edge reorientations. In Section 4.1, we introduce a simple online algorithm to handle updates without reorientation. In Section 4.2, we prove lower bounds on Δ for different update sequences when this algorithm is used to maintain an orientation. In Section 4.3, we prove upper bounds on Δ for the same algorithm. We require all update sequences to be arboricity-α-preserving.

4.1 Orienting Dynamic Graphs Online without Edge Reorientations

The algorithm we analyze in this chapter is the following: To insert an edge (u, v), we orient it from u to v if $d_o(u) \leq d_o(v)$. Otherwise, it is oriented from v to u. To delete an edge (u, v), we simply search for the edge and delete it. Let A_{wre} denote this algorithm.

4.2 Lower Bounds

Let $(u, v)^{i_G}$ denote the insertion of an edge oriented from u to v into G. Let $(u, v)^{d_G}$ denote the deletion of an edge oriented from u to v from G. We use these in the description of some update sequences to be defined later.

4.2.1 Insertion-Only Update Sequences

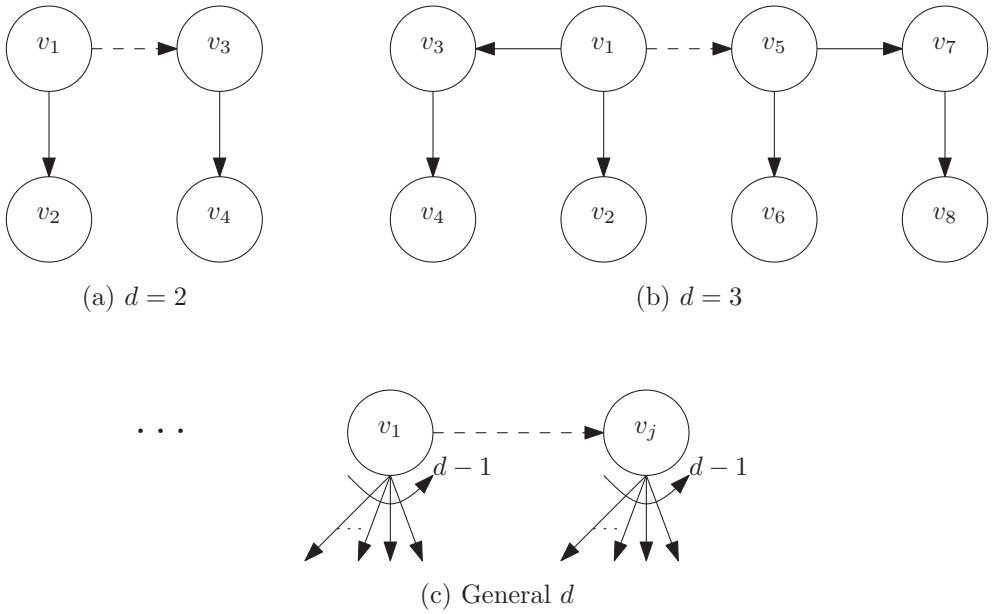
We first analyze algorithm A_{wre} by considering update sequences that contain only insertions. Note that an arboricity-α-preserving insertion sequence has at most $\alpha(n-1)$ operations because the final graph must have arboricity at most α and thus at most $\alpha(n-1)$ edges.

Lemma 12 *Consider an initially empty n-vertex graph G. There exists an arboricity-α-preserving insertion sequence that forces at least one vertex of G to have out-degree $\Omega(\lg n)$ if insertions are handled by algorithm A_{wre}.*

Proof. It suffices to prove the lemma for $\alpha = 1$, as this implies the correctness of the lemma for larger values of α. Specifically, we prove that every graph G with $n \geq 1$ vertices and no edges admits a sequence of no more than $n-1$ edge insertions that forces at least one vertex of G to have out-degree at least $\lfloor \lg n \rfloor$ and transforms G into a forest.

The proof is by induction on n. For $n = 1$, the claim is trivial because the only vertex in a 1-vertex graph has out-degree $0 = \lg 1$ without the need to perform any insertions. So assume $n > 1$ and the claim holds for all $n' < n$.

If n is not a power of 2, then $\lfloor \lg n \rfloor = \lfloor \lg(n-1) \rfloor$. By the inductive hypothesis, there exists an insertion sequence of no more than $n-2$ edge insertions on an $(n-1)$-vertex graph that forces at least one vertex to have out-degree at least $\lfloor \lg(n-1) \rfloor = \lfloor \lg n \rfloor$ and transforms G into a forest. The same process can be applied to an n-vertex graph by simply not using one of the vertices in the construction.



(a) $d = 2$ (b) $d = 3$

(c) General d

Figure 4.1: Inductive definition of $S(G, d)$

If n is a power of 2, we divide the vertex set of G into two disjoint subsets V' and V'' of size $n/2$ each. By the inductive hypothesis, there exists an insertion sequence of length at most $n/2 - 1$ over V' that creates a forest with vertex set V' and forces at least one vertex in V' to have out-degree at least $\lg(n/2) = \lg n - 1$. Let $v' \in V'$ be such a vertex. Analogously, there exists an insertion sequence of length at most $n/2 - 1$ over V'' that creates a forest with vertex set V'' and forces at least one vertex in V'' to have out-degree at least $\lg n - 1$. Let $v'' \in V''$ be such a vertex. By concatenating these two update sequences and then adding an edge between v' and v'', we obtain an update sequence of length at most $2(n/2 - 1) + 1 = n - 1$ that creates a forest with the same vertex set of G. Moreover, no matter how we orient the edge between v' and v'', either v' or v'' has degree at least $\lg n$ after this insertion. Thus, the claim holds also for n. Figure 4.1 illustrates the process of performing an update sequence $S(G, d)$ defined by above inductive proof to force one vertex of G to have out-degree d. The dashed lines correspond the edges added between v' and v'' at each inductive step. □

4.2.2 Update Sequences of Length at Most $\alpha(n-1)$

We next consider update sequences of length at most $\alpha(n-1)$, where n is the number of vertices in the graph, for two reasons. First, an insertion-only update sequence is a special update sequence of length at most $\alpha(n-1)$ without deletions. By Lemma 15, which is presented later, the maximum out-degree, Δ, is bounded by $O(\alpha \log n)$ in this special case. It is interesting to know how deletions impact the largest possible value of Δ. Second, a $(2\alpha-1)$-orientation of a graph can be computed in $O(\alpha(n-1))$ time [4], so if we recompute a $(2\alpha-1)$-orientation after $\alpha(n-1)$ updates, the overhead of rebuilding is only $O(1)$ amortized time. This can be combined with algorithm A_{wre} to handle each update using $O(1)$ edge reorientations amortized.

Lemma 13 *Consider an initially empty n-vertex graph G. For any $\alpha \geq 3$, there exists an arboricity-α-preserving update sequence of length at most $\alpha(n-1)$ that forces at least one vertex of G to have out-degree $\Omega(\min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\})$ if updates are handled by algorithm A_{wre}.*

Proof. It suffices to prove the lemma by giving an arboricity-3-preserving update sequence of length no greater than $\alpha(n-1)$ that produces a vertex of out-degree $\Omega(\min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\})$.

We inductively define an update sequence $S(G,d)$, for any positive integer d, such that after the operations in $S(G,d)$ are performed, there are $d+1$ vertices in G with out-degrees are $0, 1, \ldots, d-1, d$, provided that n is sufficiently large. Let V denote the vertices of G and $V_{S(G,d)}$ denote the vertices updated by the operations in $S(G,d)$. Let $G_{S(G,d)}$ denote the subgraph of G defined by vertices in $V_{S(G,d)}$ and the edges between them after the operations in $S(G,d)$ are performed.

In the base case, $d = 1$, and $S(G,1)$ contains a single edge insertion between two arbitrary vertices, x and y. Then $(x,y)^{i_G}$ increases x's out-degree to 1. Therefore, there are vertices, x and y, with out-degrees 1 and 0 in G, respectively, after

performing $S(G, 1)$.

Assume that we already know how to construct $S(G, d-1)$ and we want to construct $S(G, d)$. We define $S(G, d)$ by performing a sequence, U_d, of extra updates, on G after performing $S(G, d-1)$. Let $u_0, u_1, \ldots, u_{d-1}$ denote the vertices whose out-degrees become $0, 1, \ldots, d-1$, respectively, after performing $S(G, d-1)$. We define U_d to be the concatenation of d subsequences. The k'th subsequence is denoted by $sq(U_d, k)$, where $0 \leq k \leq d-1$. Let v be an arbitrary vertex in $V \setminus V_{S(G,d-1)}$. The first $d-1$ subsequences of U_d is are used to increase v's out-degree to $d-1$. The last subsequence increases u_{d-1}'s out-degree to d. Each of the first $d-2$ subsequences of U_d is further composed of three subsequences. To define $sq(U_d, k)$, for $0 \leq k \leq d-3$, let v_k be an arbitrary vertex that is not v and has not been updated by any operations before $sq(U_d, k)$. The first subsequence of $sq(U_d, k)$ contains k edge insertions between v_k and the vertices $u_0, u_1, \ldots, u_{k-1}$, i.e., insertions $\{(v_k, u_0)^{i_G}, (v_k, u_1)^{i_G}, \ldots, (v_k, u_{k-1})^{i_G}\}$. Each of these edge insertions inserts an edge between v_k and another vertex, u_p, whose out-degree is equal to $d_o(v_k)$, for $p = 0, 1, \ldots, k-1$. Therefore, by calling $Insert(G, (v_k, u_p))$, the newly inserted edge is always oriented from v_k to u_p. Hence, performing this subsequence will increase v_k's out-degree to k. The second subsequence of $sq(U_d, k)$ is a single edge insertion $(v, v_k)^{i_G}$ to increase v's out-degree by one. This edge is oriented from v to v_k because v's out-degree is $k-1$ at the time of the insertion while v_k's out-degree is k. The third subsequence of $sq(U_d, k)$ contains deletions of the edges inserted in the first subsequence of $sq(U_d, k)$, i.e., deletions $\{(v_k, u_0)^{d_G}, (v_k, u_1)^{d_G}, \ldots, (v_k, u_{k-1})^{d_G}\}$. Note that v's out-degree is $d-2$ after performing the first $d-2$ subsequences of U_d. The second last subsequence, $sq(U_d, d-2)$, of U_d contains a single edge insertion $(v, u_{d-2})^{i_G}$, which increases v's out-degree to $d-1$. The last subsequence, $sq(U_d, d-1)$, of U_d contains a single edge insertion $(u_{d-1}, v)^{i_G}$, which increases u_{d-1}'s out-degree to d.

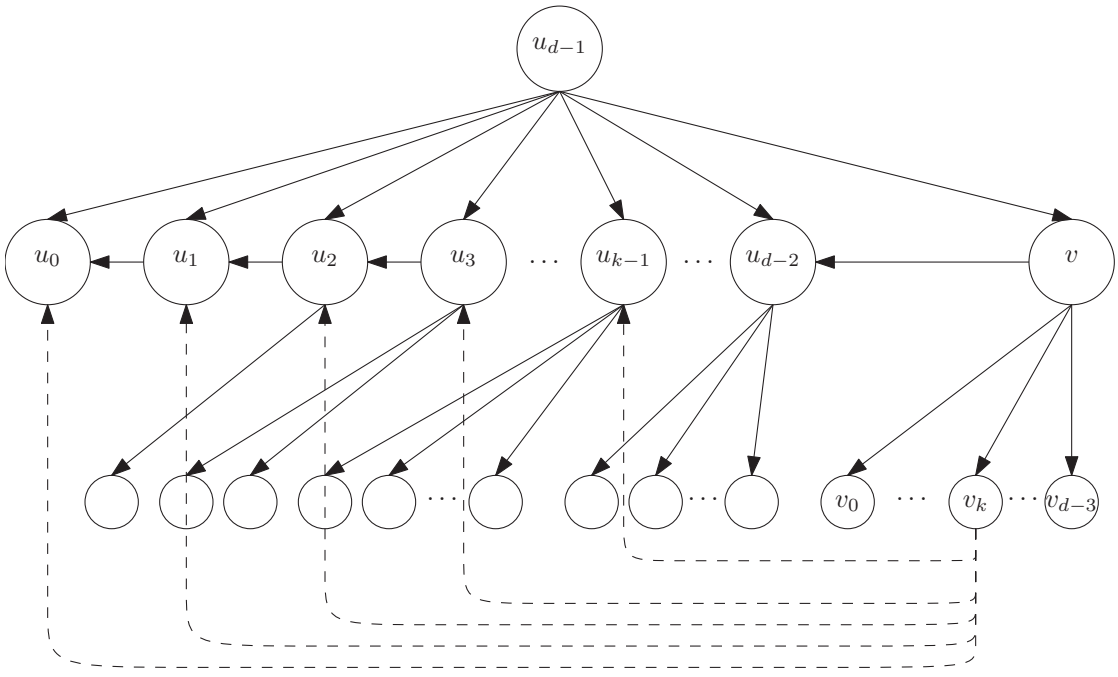


Figure 4.2: The graph $G_{S(G,d)}$

Performing U_d increases the out-degree of u_{d-1} from $d-1$ to d, and increases the out-degree of v from 0 to $d-1$. The out-degrees of $u_0, u_1, \ldots, u_{d-2}$ remain $0, 1, \ldots, d-2$, respectively. Therefore, there are $d+1$ vertices with out-degrees $0, 1, \ldots, d$ after performing $S(G, d-1) \circ U_d$, where $\circ$ denotes sequence concatenation. Thus, we have $S(G, d) = S(G, d-1) \circ U_d$. Figure 4.2 illustrates the above process. The dashed lines correspond to outgoing edges of v_k that are inserted and later deleted as part of $sq(U_d, k)$.

We claim that $S(G, d)$ is an arboricity-3-preserving update sequence by showing that G can always be decomposed into three forests. To see this, we first prove by induction that $G_{S(G,d)}$ can be decomposed into two forests for any positive integer d. In particular, we prove that $G_{S(G,d)}$ consists entirely of the following vertices and edges:

- A vertex, z, with out-degree d.
- The d out-neighbours of z, denoted by $y_0, y_1, \ldots, y_{d-1}$, form a path. More

precisely, there is exactly one edge from y_j to y_{j-1}, for $j = 1, 2, \ldots, d-1$.

- Each y_i has exactly i out-neighbours. One out-neighbour is y_{i-1}, and all the other out-neighbours have out-degree 0.

This implies that $G_{S(G,d)}$ can be decomposed into two forests since the edges between vertices $y_0, y_1, \ldots, y_{d-1}$ form a forest and the remaining edges form the other.

We now prove the above claim regarding $G_{S(G,d)}$. In the base case, $d = 1$. The graph $G_{S(G,1)}$ has two vertices connected by a single edge and the graph $G \setminus G_{S(G,1)}$ has no edges. Thus, the claim holds trivially. Next assume that this claim holds for $G_{S(G,d-1)}$. We prove it for $G_{S(G,d)}$. Let $u_0, u_1, \ldots, u_{d-1}$ denote the vertices whose out-degrees in $G_{S(G,d)}$ are $0, 1, \ldots, d-1$, respectively, and such that the out-neighbours of u_{d-1} are $u_0, u_1, \ldots, u_{d-2}$ (here we use the same vertex names as used in the construction of $S(G, d)$). Figure 4.2 illustrates this inductive hypothesis. Then, by the induction hypothesis, there is an edge from u_i to u_{i-1}, for $i = 1, 2, \ldots, d-2$. Each u_i, where $0 \leq i \leq d-2$, has exactly i out-neighbours, which are u_{i-1} and $i-1$ vertices with out-degree 0 in $G \setminus \{u_0, u_1, \ldots, u_{d-1}\}$. Then, from the way we construct U_d, in $G_{S(G,d)}$, v has an outgoing edge that connects to u_{d-2} and an incoming edge from u_{d-1}. The remaining outgoing edges of v connect to $v_0, v_1, \ldots, v_{d-3}$. Each vertex v_k has out-degree 0 after the delete operations in $sq(U_d, k)$ are performed. Hence, u_{d-1} has out-degree d and its out-neighbours $u_0, u_1, \ldots, u_{d-1}, v$ form a path in $G_{S(G,d)}$. The out-neighbours of the vertices $u_0, u_1, \ldots, u_{d-2}, v$ in $G \setminus \{u_0, u_1, \ldots, u_{d-1}, v\}$ have out-degree 0 in $G_{S(G,d)}$. The out-neighbour of v in $\{u_0, u_1, \ldots, u_{d-1}, v\}$ is u_{d-1} in $G_{S(G,d)}$. The out-neighbours of u_i in $\{u_0, u_1, \ldots, u_{d-1}, v\}$, where $0 < i \leq d-2$, is u_{i-1} in $G_{S(G,d)}$. Thus, the induction hypothesis holds also for $G_{S(G,d)}$.

Note that after an arbitrary operation in any subsequence U_d in $G_{S(G,d)}$ is performed, the edges of G that are not in $G_{S(G,d)}$ are the edges that are inserted as the outgoing edges of the same vertex, so they form a forest. These edges are the outgoing

edges of a vertex, v_k, for a certain $k \in [0, d-3]$, inserted in the first subsequence of $sq(U_d, k)$ and later deleted in the third subsequence of $sq(U_d, k)$. Therefore, graph G can always be decomposed into three forests when performing U_d for any integer d. Hence $S(G, d)$ is an arboricity-3-preserving update sequence.

Now we bound $|V_{S(G,d)}|$ and $|S(G, d)|$. From the way in which we construct $S(G, d)$, we can see that each subsequence $sq(U_d, k)$ of U_d except the last one updates a vertex that has not been updated before, so U_d updates $d-1$ new vertices and $|V_{S(G,d)}| = |V_{S(G,d-1)}| + d - 1$. Recalling that $|V_{S(G,1)}| = 2$ in the base case, we have $|V_{S(G,d)}| = 2 + 1 + 2 + 3 + \ldots + (d-1) = \frac{1}{2}d(d-1) + 2$. In addition, for each of the first $d-2$ subsequences of U_d, where $d \geq 2$, the length of the first subsequence of $sq(U_d, k)$ is k. The second subsequence has one operation, and the third subsequence has k operations. Therefore, $|sq(U_d, k)| = 2k + 1$ and the total length of the first $d-2$ subsequences of U_d is $\sum_{k=0}^{d-3}(2k+1) = (d-2)^2$. Since each of the last two subsequences of U_d contains a single edge insertion, $|U_d| = (d-2)^2 + 2$. This shows that $|S(G, d)| = |S(G, d-1)| + (d-2)^2 + 2$ and thus $|S(G, d)| = |S(G, 1)| + \sum_{r=2}^{d}((r-2)^2 + 2) = \frac{1}{3}d^3 - \frac{3}{2}d^2 + \frac{25}{6}d - 2$ since $|S(G, 1)| = 1$. Note that $|V_{S(G,d)}| < d^2$ and $|S(G, d)| < d^3$ when $n \geq 3$.

It remains to prove that the update sequence $S(G, \lfloor \min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\}\rfloor)$, which produces a vertex of out-degree $\Omega(\min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\}\rfloor))$, does not run out of vertices and has length no greater than $\alpha(n-1)$. Formally: $|V_{S(G, \lfloor \min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\}\rfloor)}| \leq n$ and $|S(G, \lfloor \min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\}\rfloor)| \leq \alpha(n-1)$. There are two cases. In the first case, $\sqrt{n} \geq \sqrt[3]{\alpha(n-1)}$ and the sequence is $S(G, \lfloor \sqrt[3]{\alpha(n-1)}\rfloor)$. From the conclusion in the previous paragraph $|V_{S(G, \lfloor \sqrt[3]{\alpha(n-1)}\rfloor)}| < (\lfloor \sqrt[3]{\alpha(n-1)}\rfloor)^2 \leq (\sqrt{n})^2 = n$ and $|S(G, \lfloor \sqrt[3]{\alpha(n-1)}\rfloor)| < (\lfloor \sqrt[3]{\alpha(n-1)}\rfloor)^3 \leq \alpha(n-1)$. In the second case, $\sqrt{n} < \sqrt[3]{\alpha(n-1)}$ and the sequence is $S(G, \lfloor \sqrt{n}\rfloor)$. Thus, $|V_{S(G, \lfloor \sqrt{n}\rfloor)}| < (\lfloor \sqrt{n}\rfloor)^2 \leq (\sqrt{n})^2 = n$ and $|S(G, \lfloor \sqrt{n}\rfloor)| < (\lfloor \sqrt{n}\rfloor)^3 \leq (\sqrt{n})^3 < (\sqrt[3]{\alpha(n-1)})^3 = \alpha(n-1)$. This completes the proof. $\square$

4.2.3 Arbitrary Update Sequences

We now consider arbitrary update sequences.

Lemma 14 *Consider an initially empty n-vertex graph G whose edges are updated using algorithm A_{wre}. For any $\alpha \geq 3$, there exists an arboricity-α-preserving update sequence that produces a vertex of out-degree $\Omega(\sqrt{n})$.*

Proof. It suffices to prove the lemma by giving an arboricity-3-preserving update sequence that produces a vertex of out-degree $\Omega(\sqrt{n})$.

We claim that the update sequence $S(G, \lfloor\sqrt{n}\rfloor)$ defined in the proof of lemma 13 is such a sequence. To see this, it suffices to show that $|V_{S(G,\lfloor\sqrt{n}\rfloor)}| \leq n$, i.e., there are enough vertices in G so that update sequence $S(G, \lfloor\sqrt{n}\rfloor)$ can be performed. From the proof of Lemma 13, $|V_{S(G,\lfloor\sqrt{n}\rfloor)}| < (\lfloor\sqrt{n}\rfloor)^2 \leq n$. This completes the proof. □

4.3 Upper Bounds

In this section, we prove upper bounds on Δ when algorithm A_{wre} is used to handle updates. Specifically, in Section 4.3.1, we prove an upper bound on Δ for insertion-only update sequences; in Section 4.3.2, we prove an upper bound on Δ for update sequences of length at most $\alpha(n-1)$; and in Section 4.3.3, we prove an upper bound on Δ for arbitrary update sequences.

4.3.1 Insertion-Only Update Sequences

Lemma 15 *Consider an initially empty n-vertex graph G whose edges are updated using algorithm A_{wre}. There does not exist an arboricity-α-preserving insertion sequence that produces a vertex of out-degree greater than $\min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{2m})$, where m is the the length of the update sequence.*

Proof. By Lemmas 7 and 11, it suffices to prove that the orientation of G always maintains Invariant 1 if algorithm A_{wre} is used to handle the operations in an arboricity-α-preserving insertion sequence.

Let $\overrightarrow{G}$ denote the orientation of G after an arbitrary update in an arboricity-α-preserving insertion sequence and u denote an arbitrary vertex of $\overrightarrow{G}$. Let d be the out-degree of u. Let $v_0, v_1, \ldots, v_{d-1}$ be the out-neighbours of u sorted in the order in which the edges $(u, v_0), (u, v_1), \ldots, (u, v_{d-1})$ are inserted. Then, $d_o(u) = i$ immediately before the edge (u, v_i) is inserted. Thus, $d_o(v_i) \geq i$ when inserting the edge (u, v_i) since the edge (u, v_i) was oriented from u to v_i by algorithm A_{wre}. Moreover, when the updates in an insertion-only update sequence are being performed, a vertex's out-degree never decreases. Therefore, u's out-neighbours $v_0, v_1, \ldots, v_{d-1}$ have out-degree $d_o(v_0) \geq 0, d_o(v_1) \geq 1, \ldots, d_o(v_{d-1}) \geq d-1$ in $\overrightarrow{G}$, respectively. Hence, $\overrightarrow{G}$ maintains Invariant 1 during the entire update sequence. □

4.3.2 Update Sequences of Length at Most $\alpha(n-1)$

Lemma 16 *Consider an initially empty n-vertex graph G whose edges are updated using algorithm A_{wre}. There does not exist an arboricity-α-preserving update sequence of length at most $\alpha(n-1)$ that produces a vertex of out-degree greater than $O(\sqrt{\alpha(n-1)})$.*

Proof. Let $\overrightarrow{G}$ denote the orientation of G after an arbitrary update in an update sequence of length at most $\alpha(n-1)$. Let u be a vertex with the maximum out-degree in $\overrightarrow{G}$ and let d be its out-degree. Let $v_0, v_1, v_2, \ldots, v_{d-1}$ be the out-neighbours of u sorted in the order in which the edges $(u, v_0), (u, v_1), \ldots, (u, v_{d-1})$ are inserted. Then, $d_o(u) = i$ immediately before the edge (u, v_i) is inserted. Thus, $d_o(v_i) \geq i$ when inserting the edge (u, v_i) since the edge (u, v_i) was oriented from u to v_i by algorithm A_{wre}. In addition, all the outgoing edges of $v_0, v_1, v_2, \ldots, v_{d-1}, u$ are

distinct and the update sequence starts from the empty graph, so there are at least $0+1+2+3+\ldots+d-1+d = \frac{1}{2}d(d+1)$ update operations in the update sequence to insert these edges. Since the length of the update sequence is at most $\alpha(n-1)$, we have $\frac{1}{2}d(d+1) \leq \alpha(n-1)$. Therefore, $d = O(\sqrt{\alpha(n-1)})$. This completes the proof. □

4.3.3 Arbitrary Update Sequences

Lemma 17 *Consider an initially empty n-vertex graph G whose edges are updated using algorithm A_{wre}. There does not exist an arboricity-α-preserving update sequence that produces a vertex of out-degree greater than $O((\alpha n)^{2/3})$.*

Proof. Let $U = \langle u_1, u_2, \ldots, u_k \rangle$ be an arbitrary arboricity-α-preserving update sequence on an initially empty graph G_0 with n vertices, and let G_i be the graph produced by applying the update sequence $\langle u_1, u_2, \ldots, u_i \rangle$ to G_0. The core of the proof is to show that, if G_k has a vertex whose out-degree is d, then there exists a graph G_i, $i \leq k$, that has at least $\frac{(2d-4)^{3/2}}{5}$ edges. Since U is arboricity-α-preserving, G_i has arboricity at most α. Since G_i has n vertices and an n-vertex graph of arboricity at most α has at most $\alpha(n-1)$ edges, we must have $\frac{(2d-4)^{3/2}}{5} \leq \alpha(n-1)$ and thus $d < \frac{(5\alpha(n-1))^{2/3}}{2} + 2 = O((\alpha n)^{2/3})$.

Let x be a vertex with out-degree d in G_k. Since we will argue about out-degrees in different graphs G_i, we use $d_i(x)$ to denote the out-degree of vertex x in G_i. In other words, $d_k(x) = d$. Since deletions do not increase vertex degrees, we can assume w.l.o.g. that the last update u_k in U is an insertion and it is the one that increases x's out-degree from $d-1$ to d, that is, $d_{k-1}(x) = d-1$. Let (x, y) be the out-edge of x added by u_k. Since $d_{k-1}(x) = d-1$ and the edge (x, y) is oriented from x to y in G_k, we must have $d_{k-1}(y) \geq d-1$. We now define $i_0 := k-1$, $V_0 := \{x, y\}$, E_0 to be the set of out-edges of x and y in G_{k-1}, and $D_0 := \{d_{k-1}(x), d_{k-1}(y)\}$. D_0 is a

multiset, that is, may contain the same value more than once. We use (i_0, V_0, E_0, D_0) as the seed to generate a sequence of tuples $S := \langle (i_j, V_j, E_j, D_j) | j \geq 0 \rangle$ such that, for all j, $D_j = \{d'_j(z) | z \in V_j\}$, where $d_{i_j}(z) \geq d'_j(z)$ for all $z \in V_j$, and E_j contains $d'_j(z)$ out-edges for each vertex $z \in V_j$. In particular, $\sum_j = \sum_{d \in D_j} d$ is a lower bound on the total number of out-edges of the vertices in V_j in G_{i_j}. For $j = 0$, these conditions are clearly satisfied, $\sum_0 \geq 2(d-1)$, and $\min D_0 = d - 1$. The last step of the proof is to show that there exists an index j such that $\sum_j \geq \frac{(2d-4)^{3/2}}{5}$.

Given a tuple (i_j, V_j, E_j, D_j), if $\min D_j = 0$, we terminate the construction and (i_j, V_j, E_j, D_j) is the last tuple in S. If $\min D_j > 0$, we construct $(i_{j+1}, V_{j+1}, E_{j+1}, D_{j+1})$ as follows. We call this the $(j+1)$*st step* in the construction of S. Since all edges in E_j are edges of G_{i_j}, there exists a maximal index $i_{j+1} < i_j$ such that $u_{i_{j+1}+1}$ inserts an edge $(x, y) \in E_j$. By the choice of i_{j+1}, all edges in $E_j \setminus \{(x, y)\}$ are edges of $G_{i_{j+1}}$. We define sets $V'_{j+1} := V_j, E'_{j+1} := E_j \setminus \{(x, y)\}$, and $D'_{j+1} := D_j \setminus \{d'_j(x)\} \cup \{d'_j(x) - 1\}$. The tuple $(i_{j+1}, V'_{j+1}, E'_{j+1}, D'_{j+1})$ clearly satisfies the conditions stated above.

If $y \in V_j$, the we define $(i_{j+1}, V_{j+1}, E_{j+1}, D_{j+1}) := (i_{j+1}, V'_{j+1}, E'_{j+1}, D'_{j+1})$ and call the $(j+1)$st step *decreasing* because $\sum_{j+1} = \sum_j -1$.

If $y \notin V_j$, we set $V_{j+1} := V'_{j+1} \cup \{y\}, E_{j+1} := E'_{j+1} \cup E_{j+1}(y)$, where $E_{j+1}(y)$ is the set of out-edges of y in $G_{i_{j+1}}$, and $D_{j+1} := D'_{j+1} \cup \{d_{i_{j+1}}(y)\}$. In this case, we call the $(j+1)$st step *increasing* because $\sum_{j+1} = \sum_j + d'_j(x) - 2$, which is greater than $\sum_j$ unless $d'_j(x) \leq 2$. To prove this bound on $\sum_{j+1}$, it suffices to observe that $d_{i_{j+1}}(y) \geq d_{i_{j+1}}(x) \geq d'_j(x) - 1$ because the insertion of edge (x, y) into $G_{i_{j+1}}$ orients the edge from x to y.

Note that, no matter whether the $(j+1)$st step is decreaseng or increasing, we have $\min D_{j+1} \geq \min D_j - 1$ because in both cases we replace $d'_j(x)$ in D_j with $d'_j(x) - 1$ in D_{j+1} and, if the $(j+1)$st step is increasing, we add $d_{i_{j+1}}(y) \geq d'_j(x) - 1$ to D_{j+1}. Thus, the sequence S has at least $\min D_0 = d - 1$ steps.

It remains to prove that there exists a tuple $(i_j, V_j, E_j, D_j) \in S$ such that $\sum_j \geq$

$\frac{(2d-4)^{3/2}}{5}$. Let $0 < j_1 < j_2 < \ldots$ be the sequence of indices of increasing steps in the construction of S, and let $\delta_h = \sum_{j_h} - \sum_{j_h-1} + 1$ for all $h \geq 1$. Then, since $\sum_{j-1} - \sum_j = 1$ for every decreasing step, we have $\sum_{j_h} = \sum_0 + \sum_{l=1}^{h} \delta_l - j_h$. Next observe that $V_{j+1} \supseteq V_j$ for all j, that $V_{j+1} = V_j$ if the $(j+1)$st step is decreasing, and that $|V_{j+1}| = |V_j| + 1$ if the $(j+1)$st step is increasing. Thus, since $|V_0| = 2$, we have $|V_{j_h}| = h + 2$. For each $0 \leq j \leq j_h$, the update u_{i_j+1} inserts an edge between vertices in $V_j \subseteq V_{j_h}$ and this edge is not inserted by the update $u_{i_{j'}+1}$ corresponding to any other step $j' \neq j$. The former follows because we just observed that $V_j \subseteq V_{j+1}$ for all j. The latter can be seen as follows: Since $V_j \subseteq V_{j+1}$ for all j, there exists exactly one index j for any vertex $x \in V_{j_h}$ such that $x \notin V_j$ and $x \in V_{j+1}$. (This holds even for the vertices in V_0 after defining $V_{-1} = \emptyset$.) By the properties of the tuples in S, $E_{j'}$ contains no out-edges of x, for any $0 \leq j' \leq j$. In the $(j+1)$st step, we add all out-edges of x in $G_{i_{j+1}}$ to E_{j+1}. For each subsequent step $j'' \geq j+1$ such that $u_{i_{j''}+1}$ inserts an out-edge (x, y) of x, this edge belongs to $E_{j''-1}$ but not to $E_{j''}$. Thus every out-edge of x is inserted by at most one update operation $u_{i_{j''}+1}$ corresponding to a step in S. The fact that each of the first j_h steps can be charged to a unique (directed) edge in a graph with $|V_h| = h + 2$ vertices implies that $j_h \leq (h+2)(h+1)$. Together with the fact that $\sum_0 \geq 2(d-1)$, we have $\sum_{j_h} \geq 2(d-1) - (h+2)(h+1) + \sum_{l=1}^{h} \delta_l$.

Since S contains at least $d-1$ steps and we have just argued that $j_h \leq (h+2)(h+1)$, S contains at least $\sqrt{d-1} - 1$ increasing steps. For the lth increasing step, we have $\delta_l \geq d'_{j_l}(x) \geq \min D_{j_l}$. Since $j_l \geq (l+2)(l+1)$, $\min D_0 = d - 1$, and $\min D_{j+1} \geq \min D_j - 1$ for all j, we thus have $\delta_l \geq \min D_{j_l} \geq d - 1 - (l+2)(l+1)$. Plugging this into the inequality for $\sum_{j_h}$, we obtain that $\sum_{j_h} \geq \frac{(3d-5)h + 6d - h^3 - 3h^2}{3}$. Now, since there are at least $\sqrt{d-1} - 1$ increasing steps, we cam set $h := \lceil \sqrt{d-1} - 1 \rceil$ in this inequality. This gives $\sum_{j_h} \geq \frac{(2d-4)\sqrt{d-1}+8}{3} > \frac{(2d-4)\sqrt{2d-2}}{5} > \frac{(2d-4)^{3/2}}{5}$, as claimed. This finishes the proof. □

Chapter 5

Conclusion and Future Work

Graph orientation has been used to solve many fundamental graph problems, including adjacency queries and maintaining maximal matchings. Motivated by these applications, we studied this problem in three major directions in this thesis: designing solutions with amortized time bounds, designing solutions with worst-case time bounds, and proving lower and upper bounds on the maximum vertex out-degree with an online algorithm that handles updates without requiring edge reorientation.

To design solutions with amortized time bounds, we proposed a new offline strategy to prove that Brodal and Fagerberg's online algorithm can maintain an $O(\beta\alpha)$-orientation in $O(\frac{\lg(n/(\beta\alpha))}{\beta})$ amortized insertion time and $O(\beta\alpha)$ worst-case deletion time under arboricity-α-preserving updates, where n is the number of vertices of the graph, for any chosen $\beta \geq 1$. This result shows a new tradeoff between update time and the maximum vertex out-degree in the orientation. We apply it to maintain maximal matchings of dynamic graphs, and our solution is currently the best for any sparse graph whose arboricity is $o(\lg n)$. This is interesting as maximal matching is a fundamental problem in graph theory.

To guarantee worst-case time bounds, we proposed a new algorithm to maintain an $O(\Delta)$-orientation with $O(\Delta)$ worst-case update time, where $\Delta \leq \min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{2m})$. This algorithm gives a different tradeoff between insertion and deletion times when $\alpha = \omega(\lg n)$, compared to Kopelowitz and Krauthgamer's algorithm which requires $O(\alpha^2 + \alpha \lg n)$ time for insertion and $O(\alpha + \lg n)$ time for deletion, and our insertion algorithm does not require edge reorientation. We apply it to maintain

maximal matchings of dynamic graphs with $O(\min(\alpha \lg(n/\alpha), \sqrt{m}))$ worst-case update time using $O(n+m)$ space. This result matches the previously best result for arbitrary graphs, and its time complexity adapts to the arboricity of graphs. We also apply it to get an adjacency query data structure with a $\lg \lg \lg n$ factor speed up for insertion when the graph's arboricity is upper bounded by $O(1)$.

To prove lower and upper bounds on maximum vertex out-degree, we proved that we can maintain an $O((\alpha n)^{2/3})$-orientation by simply orienting a newly inserted edge from the endpoint with lower out-degree to the endpoint with higher out-degree or orienting the edge arbitrarily if both endpoints have equal out-degree. We also proved that with the same strategy, the maximum vertex out-degree is upper bounded by $\min(2\alpha \lg(n/\alpha) + 2\alpha, \sqrt{2m})$ during the execution of the operations in an insertion-only arboricity-α-preserving update sequence, but the maximum vertex out-degree can go up to $\Omega(\min\{\sqrt[3]{\alpha(n-1)}, \sqrt{n}\})$ even in only $\alpha(n-1)$ updates, for any $\alpha \geq 3$, when deletions are allowed, which reflects the impact of deletions on the orientation.

There are still many open problems on dynamic graph orientation. The first is whether we can maintain an $O(\alpha)$-orientation of a graph of arboricity α in $O(\alpha)$ or $O(1)$ amortized update time. The second is how to maintain an $O(\alpha)$-orientation of a graph of arboricity α in logarithmic or polylogarithmic worst-case update time. The third is to find tight lower and upper bounds on the number of edge reorientations per update for an optimal algorithm to maintain a certain orientation of a dynamic graph. These open problems are open even for graphs of arboricity one.

These open problems are about update time and maximum vertex out-degree. Some applications may care more about space efficiency. Thus, another open problem is how to represent low-arboricity graphs succinctly to support adjacency queries. This open problem was proposed by Brodal and Fagerberg [5]. Note that there are succinct data structures for planar graphs [8, 38], but not for arbitrary graphs with constant arboricity.

Bibliography

[1] Abhash Anand, Surender Baswana, Manoj Gupta, and Sandeep Sen. Maintaining approximate maximum weighted matching in fully dynamic graphs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, pages 257–266, 2012.

[2] Arne Andersson. Faster deterministic sorting and searching in linear space. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, page 135141. IEEE Computer Society, 1996.

[3] Arne A. Andersson and Mikkel Thorup. Tight(er) worst-case bounds on dynamic searching and priority queues. In *Proceedings of the 32th Annual ACM Symposium on Theory of Computing*, STOC '00, pages 335–342, New York, NY, USA, 2000. ACM.

[4] Srinivasa Rao Arikati, Anil Maheshwari, and Christos D. Zaroliagis. Efficient computation of implicit representations of sparse graphs. *Discrete Applied Mathematics*, 78(1-3):1–16, 1997.

[5] Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representation of sparse graphs. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, WADS '99, pages 342–351, London, UK, UK, 1999. Springer-Verlag.

[6] Julie Anne Cain, Peter Sanders, and Nick Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 469–476, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[7] Boliong Chen, Makoto Matsumoto, Jianfang Wang, Zhongfu Zhang, and Jianxun Zhang. A short proof of Nash-Williams' theorem for the arboricity of a graph. *Graphs and Combinatorics*, 10(1):27–28, 1994.

[8] Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications. *SIAM Journal on Computing*, 34(4):924–945, April 2005.

[9] Marek Chrobak and David Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoretical Computer Science*, 86(2):243–266, September 1991.

[10] David Richard Clark. *Compact Pat Trees*. PhD thesis, Waterloo, Ont., Canada, Canada, 1998. UMI Order No. GAXNQ-21335.

[11] Martin Dietzfelbinger, Anna Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23(4):738–761, August 1994.

[12] Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for approximate and exact maximum weight matching. *Computing Research Repository*, abs/1112.0790, 2011.

[13] Zdeněk Dvořák and Vojtěch Tůma. A dynamic data structure for counting subgraphs in sparse graphs. In *Proceedings of the 13th International Conference on Algorithms and Data Structures*, WADS'13, pages 304–315, Berlin, Heidelberg, 2013. Springer-Verlag.

[14] Jack Edmonds and Ellis L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

[15] David Eisenstat, Philip Klein, and Claire Mathieu. An efficient polynomial-time approximation scheme for Steiner forest in planar graphs. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 626–638. SIAM, 2012.

[16] David Eppstein. All maximal independent sets and dynamic dominance for sparse graphs. *ACM Transactions on Algorithms*, 5(4):38:1–38:14, November 2009.

[17] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $\mathcal{O}(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, June 1984.

[18] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, December 1993.

[19] Harold Gabow and Herbert Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, STOC '88, pages 407–421, New York, NY, USA, 1988. ACM.

[20] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM*, 38(4):815–853, October 1991.

[21] Roberto Grossi and Elena Lodi. Simple planar graph partition into three forests. *Discrete Applied Mathematics*, 84(1-3):121–132, 1998.

[22] Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 548–557, Berkeley, CA, USA, 2013. IEEE.

[23] Manoj Gupta and Ankit Sharma. An $\mathcal{O}(\log(n))$ fully dynamic algorithm for maximum matching in a tree. *CoRR*, abs/0901.2900, 2009.

[24] F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975.

[25] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[26] Zoran Ivkovic and Errol L. Lloyd. Fully dynamic maintenance of vertex cover. In Jan van Leeuwen, editor, *WG '93 Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science*, volume 790 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 1993.

[27] Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM Journal On Discrete Mathematics*, 5(4):596–603, 1992.

[28] Tsvi Kopelowitz. Personal communication, 2014.

[29] Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *International Colloquium on Automata, Languages and Programming(ICALP)(2)*, pages 532–543, 2014.

[30] Lukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Information Processing Letters*, 102(5):191–195, 2007.

[31] Lukasz Kowalik. Fast 3-coloring triangle-free planar graphs. *Algorithmica*, 58(3):770–789, 2010.

[32] Lukasz Kowalik and Maciej Kurowski. Oracles for bounded-length shortest paths in planar graphs. *ACM Transactions on Algorithms*, 2(3):335–363, 2006.

[33] Mei-Ko Kwan. Graphic programming using odd or even points. *Chinese Math*, 1(273-277):110, 1962.

[34] Eugene L Lawler. *Combinatorial optimization: Networks and matroids*. Courier Dover Publications, 1976.

[35] L. Lovász and M.D. Plummer. *Matching Theory*. AMS Chelsea Publishing, 1986.

[36] Silvio Micali and Vijay V. Vazirani. An $\mathcal{O}(\sqrt{|V|}|e|)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, SFCS '80, pages 17–27, Washington, DC, USA, 1980. IEEE Computer Society.

[37] Peter Bro Miltersen. Error correcting codes, perfect hashing circuits, and deterministic dynamic dictionaries. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, pages 556–563, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[38] J. I. Munro and V. Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, pages 118–126, Washington, DC, USA, 1997. IEEE Computer Society.

[39] C. St. J. A Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 36(1):445–450, 1961.

[40] C. St. J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39(1):12, 1964.

[41] Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, STOC '13, pages 745–754, New York, NY, USA, 2013. ACM.

[42] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42th ACM Symposium on Theory of Computing*, STOC '10, pages 457–464, New York, NY, USA, 2010. ACM.

[43] 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.

[44] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 118–126, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[45] Robert Endre Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.