Improved Dynamic Graph Coloring

Shay Solomon
IBM Research, TJ Watson Research Center, Yorktown Heights, NY USA

Nicole Wein
EECS, Massachusetts Institute of Technology, Cambridge, MA USA

Abstract

This paper studies the fundamental problem of graph coloring in fully dynamic graphs. Since the problem of computing an optimal coloring, or even approximating it to within $n^{1-\epsilon}$ for any $\epsilon > 0$, is NP-hard in *static graphs*, there is no hope to achieve any meaningful *computational* results for *general graphs* in the dynamic setting. It is therefore only natural to consider the *combinatorial aspects* of dynamic coloring, or alternatively, study restricted families of graphs.

Towards understanding the combinatorial aspects of this problem, one may assume a black-box access to a static algorithm for C-coloring any subgraph of the dynamic graph, and investigate the trade-off between the number of colors and the number of *recolorings* per update step. Optimizing the number of recolorings, sometimes referred to as the *recourse* bound, is important for various practical applications. In WADS'17, Barba et al. devised two complementary algorithms: For any $\beta > 0$, the first (respectively, second) maintains an $O(C\beta n^{1/\beta})$ (resp., $O(C\beta)$)-coloring while recoloring $O(\beta)$ (resp., $O(\beta n^{1/\beta})$) vertices per update. Barba et al. also showed that the second trade-off appears to exhibit the right behavior, at least for $\beta = O(1)$: Any algorithm that maintains a c-coloring of an n-vertex *dynamic forest* must recolor $\Omega(n^{\frac{2}{c(c-1)}})$ vertices per update, for any constant $c \geq 2$. Our contribution is two-fold:

- We devise a new algorithm for general graphs that improves significantly upon the first trade-off in a wide range of parameters: For any $\beta > 0$, we get a $\tilde{O}(\frac{C}{\beta}\log^2 n)$-coloring with $O(\beta)$ recolorings per update, where the $\tilde{O}$ notation supresses polyloglog(n) factors. In particular, for $\beta = O(1)$ we get constant recolorings with polylog(n) colors; not only is this an exponential improvement over the previous bound, but it also unveils a rather surprising phenomenon: The trade-off between the number of colors and recolorings is highly non-symmetric.
- For uniformly sparse graphs, we use low out-degree orientations to strengthen the above result by bounding the update time of the algorithm rather than the number of recolorings. Then, we further improve this result by introducing a new data structure that refines bounded out-degree edge orientations and is of independent interest.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms, Theory of computation → Graph algorithms analysis, Theory of computation → Dynamic graph algorithms

Keywords and phrases coloring, dynamic graph algorithms, graph arboricity, edge orientations

Digital Object Identifier 10.4230/LIPIcs.ESA.2018.72

Acknowledgements The authors thank Krzysztof Onak, Baruch Schieber and Virginia Vassilevska Williams for fruitful discussions.

1 Introduction

1.1 Background

Graph coloring is one of the most fundamental and well studied problems in computer science, having found countless applications over the years, ranging from scheduling and computational vision to biology and chemistry. A *proper* C*-coloring* of a graph $G = (V, E)$, for a positive integer C, assigns a color in $\{1, \ldots, C\}$ to every vertex, so that no two adjacent vertices are assigned the same color. The *chromatic number* of the graph is the smallest integer C for which a proper C-coloring exists. (We shall write "coloring" as a shortcut for "proper coloring", unless otherwise specified.)

This paper studies the problem of graph coloring in fully dynamic graphs subject to edge updates. A *dynamic graph* is a graph sequence $\mathcal{G} = (G_0, G_1, \ldots, G_M)$ on a fixed vertex set V, where the initial graph is $G_0 = (V, \emptyset)$ and each graph $G_i = (V, E_i)$ is obtained from the previous graph G_{i-1} in the sequence by either adding or deleting a single edge.

We investigate general graphs as well as *uniformly sparse* graphs. The "uniform density" of the graph is captured by its *arboricity*: a graph $G = (V, E)$ has *arboricity* α if $\alpha = \max_{U \subseteq V} \left\lceil \frac{|E(U)|}{|U|-1} \right\rceil$, where $E(U) = \{(u, v) \in E \mid u, v \in U\}$. That is, the arboricity is close to the maximum *density* $|E(U)|/|U|$ over all induced subgraphs of G. The class of constant arboricity graphs, which contains planar graphs, bounded tree-width graphs, and in general all minor-free graphs, as well as some classes of "real-world" graphs, has been subject to extensive research in the dynamic algorithms literature [10, 11, 50, 56, 46, 47, 39, 25, 14]. A dynamic graph of *arboricity* α is a dynamic graph such that all graphs G_i have arboricity bounded by α.

It is NP-hard to approximate the chromatic number of an n-vertex graph to within a factor of $n^{1-\epsilon}$ for any constant $\epsilon > 0$, let alone to compute the corresponding coloring [60, 33]. Consequently, there is no hope to achieve any meaningful *computational* results for *general graphs* in the dynamic setting. It is perhaps for that reason that the literature on dynamic graph coloring is sparse (see Section 1.1.1). Nevertheless, as discussed next, one may view the area of dynamic graph algorithms as lying within the wider area of *local algorithms*, in which there has been tremendous success in the context of graph coloring.

When dealing with networks of large scale, it is important to devise algorithms that are intrinsically *local*. Roughly speaking, a local algorithm restricts its execution to a small part of the network, yet is still able to solve a global task over the entire network. There is a long line of work on local algorithms for graph coloring and related problems from various perspectives. For example, seminal papers on distributed graph coloring [16, 26, 40, 4, 41, 42] laid the foundation for the area of symmetry breaking problems, which remains the subject of ongoing intensive research. Refer to the book of Barenboim and Elkin [7] for a detailed account on this topic. Additionally, graph coloring is well-studied in the areas of property testing [27, 17] and local computation algorithms [52, 24].

1.1.1 Dynamic graph coloring

In light of the computational intractability of graph coloring, previous work on dynamic graph coloring is devoted mostly to heuristics and experimental results [43, 51, 59, 29, 28, 48, 53]. From the theoretical standpoint, it is natural to consider the *combinatorial aspects* of dynamic coloring or to study restricted families of graphs; to the best of our knowledge, the only work on this pioneering front is that of Barba et al. from WADS'17 [5] and Bhattacharya et al. from SODA'18 [12]. Additionally, Parter, Peleg, and Solomon [49] studied this problem in the dynamic distributed setting, and Barenboim and Maimon [8] studied the related problem of dynamic *edge coloring*. (Our work focuses on amortized time bounds; we henceforth do not distinguish between amortized and worst-case time bounds, unless explicitly specified.)

Barba et al. [5] studied the combinatorial aspects of dynamic coloring in general graphs. They assumed that at all times the graph can be C-colored and further assumed black-box access to a static algorithm for C-coloring any subgraph of the current graph. They investigated the trade-off between the number of colors and the number of *recolorings* (i.e., the number of vertices that change their color) per update step. Optimizing the number of recolorings, sometimes referred to as the *recourse* bound, is important for various practical applications They devised two complementary algorithms: for any $\beta > 0$, the first (respectively, second) maintains an $O(C\beta n^{1/\beta})$ (resp., $O(C\beta)$)-coloring while recoloring $O(\beta)$ (resp., $O(\beta n^{1/\beta})$) vertices per update step. While these trade-offs coincide at $\beta = \log n$, each providing $O(C \log n)$-coloring with $O(\log n)$ recolorings per update, any slight improvement on one of these parameters triggers a significant blowup to the other. In particular, the extreme point $\beta = O(1)$ on the first and second trade-off curves yields a polynomial number of colors and recolorings, respectively. Barba et al. [5] also showed that the second trade-off exhibits the right behavior, at least for $\beta = O(1)$: Any algorithm that maintains a c-coloring of an n-vertex *dynamic forest* must recolor $\Omega(n^{\frac{2}{c(c-1)}})$ vertices per update, for any constant $c \geq 2$. The following question was left open.

▶ **Question 1.1.** *Does the first trade-off of [5] exhibit the right behavior, and in particular, does a constant number of recolorings require a polynomial number of colors?*

Bhattacharya et al. [12] studied the problem of dynamically coloring bounded degree graphs. For graphs of maximum degree Δ they presented a randomized (respectively deterministic) algorithm for maintaining a $(\Delta+1)$ (resp., $\Delta(1+o(1))$-coloring with amortized expected $O(\log \Delta)$ (resp., polylog(Δ)) update time. These results provide meaningful bounds only when *all vertices* have bounded degree. The following question naturally arises.

▶ **Question 1.2.** *Can we get meaningful results for the more general class of bounded arboricity graphs?*

Question 1.2 is especially intriguing because, as shown in [5], dynamic forests (which have arboricity 1) appear to provide a hard instance for dynamic graph coloring.

Parter, Peleg, and Solomon [49] studied Question 1.2 in dynamic distributed networks: They showed that for graphs of arboricity α an $O(\alpha \cdot \log^* n)$-coloring can be maintained with $O(\log^* n)$ update time. The update time in this context, however, bounds the number of *communication rounds* per update, while the number of recolorings done (and number of messages sent) per update is polynomial in n, even for forests.

1.2 Our results

We use $\tilde{O}$ notation throughout to suppress polyloglog factors.

1.2.1 General graphs

The following theorem summarizes our main result for general graphs.

▶ **Theorem 1.3.** *For any n-vertex dynamic graph that can be C-colored at all times, there is a fully dynamic deterministic algorithm for maintaining an $O(\frac{C}{\beta}\log^3 n)$-coloring with $O(\beta)$ (amortized) recolorings per update step, for any $\beta > 0$. Using randomization (against an oblivious adversary), the number of colors can be reduced by a factor of $\tilde{O}(\log n)$ while achieving an expected bound of $O(\beta)$ recolorings.*

Theorem 1.3 with $\beta = O(1)$ yields $O(1)$ recolorings with polylog(n) colors, thus answering Question 1.1 in the negative. Not only is this result an exponential improvement over the previous bound of [5], but it also unveils a rather surprising phenomenon: The trade-off between the number of colors and recolorings is highly non-symmetric.

We also note that the number of recolorings can be de-amortized. The details are omitted due to space constraints.

A runtime bound. Assuming black-box access to two efficient coloring algorithms we can bound the runtime of the algorithm from Theorem 1.3.

Black-box static algorithm. Let $A_{\mathcal{G},C}$ be a static algorithm that takes as input a graph G from a graph class $\mathcal{G}$ and a subset S of vertices in G, and computes the induced graph $G[S]$ and a C-coloring of $G[S]$ in time $T(|S|)$.

Black-box dynamic algorithm. Let A' be a fully dynamic algorithm that colors graphs of maximum degree Δ using $O(\Delta)$ colors. Such algorithms exist: there is a randomized algorithm with $O(1)$ expected amortized update time and a deterministic algorithm with $O(\text{polylog}(\Delta))$ amortized update time [12]. Let $T'(\Delta, n) \leq \text{polylog}(\Delta)$ be the runtime of an optimal deterministic algorithm for this problem. We state our results in terms of $T'(\Delta, n)$ to emphasize that any improvement over the deterministic algorithm of [12] would yield an improvement to the runtime of our algorithm.

▶ **Theorem 1.4.** *The randomized algorithm from Theorem 1.3 has expected amortized update time $O\left(\frac{\beta}{n \log n} \sum_{i=0}^{\log n} 2^i T(n/2^i)\right)$ and the deterministic algorithm from Theorem 1.3 has the same amortized update time with an additional additive factor of $T'(\frac{\log^2 n}{\beta}, n)$.*

▶ Remark. The randomized black-box dynamic algorithm of [12] that we apply in Theorem 1.4 is actually a simple observation (referred to as a "warm-up result" in [12]) which gives a 2Δ-coloring with $O(1)$ expected update time. The main result of [12], however, is an algorithm to bound the number of colors by only $\Delta + 1$ (or slightly more).

1.2.2 Uniformly sparse graphs

We answer Question 1.2 in the positive by showing that by applying the algorithms from Theorem 1.4 to arboricity α graphs we can obtain a bound on the update time rather than only the number of recolorings.

▶ **Theorem 1.5.** *There is a fully dynamic deterministic algorithm for graphs of arboricity α that maintains an $O((\frac{\alpha}{\beta})^2 \log^4 n)$-coloring in amortized $T'(\frac{\alpha \log^3 n}{\beta^2}, n) + O(\beta)$ time per update for any $\beta > 0$. Using randomization (against an oblivious adversary), the number of colors can be reduced by a factor of $\tilde{O}(\log n)$ and the expected amortized update time becomes $O(\beta)$.*

Furthermore, we improve over this result when $\beta = o(\sqrt{\log n})$ by designing an algorithm that specifically exploits the structure of arboricity α graphs.

▶ **Theorem 1.6.** *There is a fully dynamic deterministic algorithm for graphs of arboricity α that maintains an $O(\alpha \log^2 n)$-coloring in amortized $\tilde{O}(1)$ time.*

The proof of Theorem 1.6 relies on a new *layered data structure* (LDS) for bounded arboricity graphs that we expect will be more widely applicable. See Section 1.3 for more details about the LDS and the related notion of bounded out-degree edge orientations.

▶ **Definition 1.7.** Given a dynamic graph G of arboricity α, a *layered data structure (LDS)* with parameters k and Δ is a partition of the vertices into k layers $L_1, \ldots, L_k$ so that all vertices v have at most Δ neighbors in layers equal to or higher than the layer containing v.

▶ **Theorem 1.8.** *Let A'' be an algorithm for arboricity α graphs that maintains an orientation of the edges with out-degree at most D that performs amortized $F(n)$ flips per update. Then there is an algorithm to maintain an LDS for a fully dynamic graph of arboricity α with $k = O(\log n)$ and $\Delta = O(D + \alpha \log n)$ in amortized deterministic time $O(F(n))$.*

1.3 Technical overview

1.3.1 Low out-degree dynamic edge orientations

All of our results are, in different ways, intimately related to the dynamic edge orientation problem for arboricity α graphs, where the goal is to dynamically maintain a low out-degree orientation of the edges in a graph (an orientation with out-degree α always exists [45]). Our algorithm for general graphs (outlined in Section 1.3.2) is inspired by an algorithm for the dynamic edge orientation problem. Our algorithm for bounded arboricity graphs from Theorem 1.5 uses a dynamic edge orientation algorithm as a black-box. Our algorithm for bounded arboricity graphs from Theorem 1.6 uses a dynamic edge orientation to define a potential function useful in the runtime analysis (outlined in Section 3.1).

Brodal and Fagerberg [14] initiated the study of the dynamic edge orientation problem and gave an algorithm that maintains an $O(\alpha)$ out-degree orientation in amortized $O(\alpha + \log n)$ time. To analyze this algorithm, they reduced the "online" setting, where we have no knowledge of the future, to the "offline" settings, where we know the entire sequence of edge updates in advance. Thus, in the the subsequent results, it sufficed to consider only the offline setting. Kowalik [36] used an elegant argument to derive a result complementary to [14]: one can maintain an $O(\alpha \log n)$ out-degree orientation in amortized $O(1)$ time. He, Tang, and Zeh [30] completed the picture with a trade-off bound: for all $\beta \geq 1$, one can maintain an $O(\beta\alpha)$ out-degree orientation in amortized $O(\frac{\log n}{\beta})$ time. The worst-case update time of this problem has also been studied by Kopelowitz et al. [34] and Berglin and Brodal [9].

Dynamic bounded out-degree orientations are a key ingredient in a number of dynamic algorithms for graphs of bounded arboricity [37, 35, 10, 11, 46, 47, 25, 22], as well as in dynamic algorithms for general graphs [55, 10, 11, 13].

1.3.2 Overview of algorithm for general graphs

We apply two black-box coloring algorithms defined in Section 1.2.1, one static and one dynamic. For each vetex v, if it is assigned color c_1 by the static algorithm and color c_2 by the dynamic algorithm, its true color is defined by the pair (c_1, c_2).

Periodically, we run the static algorithm on a carefully chosen induced subgraph of the current graph. To select these subgraphs, we keep track of the *recent degree* of each vertex v: the number of edges incident to v that were inserted since the last time v was included as input to an instance of the static algorithm. Then, we choose the vertices of highest recent degree as input to the static algorithm, thus setting the recent degree of these vertices to zero. By repeatedly setting the recent degree of the highest recent degree vertices to zero, we obtain a bound on the maximum recent degree in the graph. Then we apply the dynamic algorithm for bounded degree graphs on only the edges that contribute to recent degrees.

We can further reduce the maximum recent degree in the graph by employing randomization: In addition to the vertices already chosen to participate in the static algorithm, we randomly select some vertices incident to newly inserted edges.

To obtain an upper bound on the maximum recent degree at all times, we model the changes in recent degree by an online 2-player balls and bins game. The game was first introduced in 1988 [38, 19] and has found a number of applications in the dynamic algorithms literature for obtaining worst-case guarantees [20, 15, 1, 57, 2, 44, 57, 58, 9, 21, 32]. To the best of our knowledge, our techniques are the first to demonstrate improved amortized guarantees using the game. We anticipate that this game will find additional applications in amortized algorithms as well as in translating offline strategies to online strategies.

The main technical content that remains are the details of each instance of the static algorithm: we have not specified when to run each instance, the precise subgraph to input, and which palette of colors to draw from. Understanding these details illuminates the key insight that allows us to improve the number of colors from the polynomial bound in [5] to polylogarithmic. We hierarchically bipartition the update sequence into $\log_2 n$ levels of nested time intervals and at the end of each interval, we apply the static algorithm. We use a separate palette of colors for each level of intervals but for subsequent instances of the static algorithm on the same level we use the same palette. To avoid color conflicts caused by this reuse of colors, we ensure that when a vertex participates in an instance of the static algorithm at the end of an interval I on a level L, it also participates in the instance of the static algorithm at the end of every superinterval of I; thus it is recolored once for each superinterval. This recoloring frees the level L color palette for future instances of the static algorithm at level L. In summary, the hierarchical partition of the update sequence into levels provides the structure that allows us to reuse colors without creating color conflicts.

This partition of the update sequence is inspired by the offline algorithm of [30] for the dynamic edge orientation problem. Adapting their ideas to our setting requires overcoming two main hurdles: a) transitioning from graphs of bounded arboricity to general graphs, and b) transitioning from the offline setting to the online setting.

1.3.3 Overview of algorithm for low arboricity graphs

The proof of Theorem 1.5 is based on the following observation: the black-box static algorithm used in Theorem 1.3 can be made efficient if $\mathcal{G}$ is the class of arboricity α graphs and we have access to a low out-degree orientation of the graph.

The bulk of the proof of Theorem 1.6 concerns the LDS (defined in Section 1.2.2). The definition of the LDS is inspired by the following property of arboricity α graphs: there exists an ordering of the vertices $v_1, \ldots, v_n$ such that every vertex has at most 2α neighbors that appear after it in the ordering [3]. Given such an ordering, consider the procedure of iteratively removing the vertices from the graph in order (or adding the vertices to the graph in reverse order) so that when each vertex is removed (or added) its degree to the current graph is only 2α. This procedure has been a key ingredient in algorithms in a variety of settings including distributed algorithms [6], parallel algorithms [3], property testing [23], and social network analysis [31, 54, 18]. We are the first to devise a data structure that dynamically maintains (an approximate version of) this ordering.

The LDS is useful for maintaining a proper coloring of a graph because the graph induced by each layer of vertices has low degree. Thus, we can apply a dynamic algorithm for graphs of bounded maximum degree on the graph induced by each individual layer. Then, because there are not too many layers in total, we can use a disjoint palette of colors for each layer.

On the other hand, simply using a low out-degree orientation of the edges does not seem to suffice for solving dynamic coloring. In general, one shortfall of a low out-degree orientation is that it is an inherently *local* data structure; each vertex only keeps track of information about its immediate neighborhood. In contrast, an LDS maintains a *global*

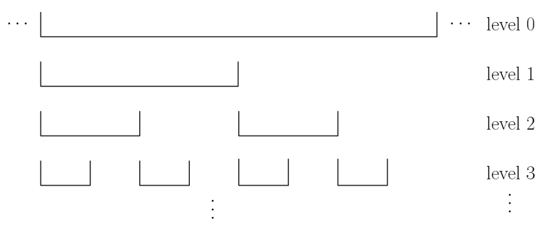

Figure 1 The set of interals.

partition of the vertices into layers. Furthermore, the LDS is designed to store strictly more information than a bounded out-degree edge orientation; by orienting all edges in an LDS from lower to higher layers, we get a bounded out-degree edge orientation. We anticipate that the LDS could be useful for solving more dynamic problems for which a bounded out-degree edge orientation does not appear to suffice.

2 Algorithm for general graphs

In this section we prove Theorem 1.3. We omit the proof of Theorem 1.4.

▶ **Theorem 2.1** (Restatement of Theorem 1.3). *There is a fully dynamic deterministic algorithm for maintaining an $O(\frac{C}{\beta}\log^3 n)$-coloring with $O(\beta)$ (amortized) recolorings per update step, for any $\beta > 0$. Using randomization (against an oblivious adversary), the number of colors can be reduced to $O(\frac{C}{\beta}\log^2 n(\log\log n + \log\beta))$ while achieving an expected bound of $O(\beta)$ recolorings.*

The algorithm is as follows. At all times, each vertex v is assigned a color c_1 by the black-box static algorithm (from the last time v was input to an instance of the static algorithm) and a color c_2 by the black-box dynamic algorithm. The true color of each v defined by the pair (c_1, c_2), so the total number of colors is the product of the number of colors used in each black-box algorithm. As mentioned in the algorithm overview (Section 1.3.2), we define a hierarchical partition of the update sequence to specify the instances of the static algorithm. First, we describe this partition, then we describe how to apply the static algorithm, and then we describe how to apply the dynamic algorithm.

2.1 Partition of update sequence

We partition the update sequence (without knowing its contents) into a set of intervals as follows. An interval is said to be of *length* ℓ if it contains ℓ update steps. We partition the entire update sequence into intervals of length $n\ell$ for some parameter ℓ (which we will later set to $\frac{\log n}{\beta}$). We say that this set of intervals is on *level* 0. Next, for each $i = 1, \ldots, \log_2 n$, the level-i intervals are obtained from the $i-1$-level intervals by splitting each $i-1$ interval in two subintervals of equal length. Note that the intervals on level $\log n$ are of length ℓ and in general the intervals on level i are of length $n\ell/2^i$.

It will be easier to work with these intervals if no two have the same ending point. So, for every pair of intervals with the same endpoint, we remove the interval on the higher numbered level. The resulting set of intervals, shown in Figure 1 is the set of intervals that we work with in the algorithm.

2.2 Applying the black-box static algorithm

At the end of each interval, we apply the black-box static algorithm. For each interval I, let $\mathcal{A}_I$ be the instance of the black-box algorithm that is executed at the end of interval I. If I is an interval on level i, we say that $\mathcal{A}_I$ is on level i. For each level, we use a separate palette of C colors, and all instances of the algorithm on the same level use the same palette of colors. In particular, if $\mathcal{A}_I$ is on level i, it uses the C colors in the range from $i \cdot C + 1$ to $(i+1)C$.

We determine the input to each $\mathcal{A}_I$ as follows. If I is on level 0, the input to $\mathcal{A}_I$ is simply the entire graph. Otherwise, we decide the input based on the update sequence. For each vertex v, we keep track of its *recent degree*, defined as the number of edges incident to v that were inserted since the last time v was included as input to an instance of the static algorithm. For each interval I, we let v_I be the vertex of highest recent degree at the end of interval I (breaking ties arbitrarily). For the deterministic algorithm, the input to each $\mathcal{A}_I$ is the set $\{v_{I'} | I' \text{ is a subinterval of } I\}$ (where an interval is considered a subinterval of itself).

For the randomized algorithm, in addition to v_I we select another vertex u_I at the end of each interval I. Specifically, we pick uniformly at random an edge insertion (y, z) from the last ℓ updates (if one exists) and then we let u_I be either y or z, chosen at random. Then the input to each $\mathcal{A}_I$ is the set $\{v_{I'} \cup u_{I'} | I' \text{ is a subinterval of } I\}$.

We note that each interval on level $\log_2 n$ contains only 1 subinterval (itself), and generally, each interval on level i contains $n/2^i$ subintervals. Thus, each $\mathcal{A}_I$ on level i takes $O(n/2^i)$ vertices as input.

2.3 Applying the black-box dynamic algorithm

We apply the black-box dynamic algorithm on the graph with the full vertex set but only the edges that count towards the recent degree of both of its endpoints. Specifically, if G denotes the input dynamic graph then the dynamic graph G' that we input to the black-box dynamic algorithm is defined as follows. G' is initially the empty graph on the same vertex set as G and whenever there is an edge update to G, the same edge is updated in G'. Additionally, when a vertex v is included as input to the static algorithm, every edge incident to v is deleted from G'.

To apply the black-box dynamic algorithm, we need to show that G' has bounded maximum degree. To do this, we apply an online 2-player balls and bins game. The game begins with N empty bins. The goal of Player 1 is to maximize the size of the largest bin and the goal of Player 2 is the opposite. At each step, the players each make a move according the following rules.

- Player 1 distributes at most k new balls to its choice of bins.
- Player 2 removes all of the balls from the largest bin (breaking ties arbitrarily).

▶ **Theorem 2.2** ([19]). *In the balls and bins game, every bin always contains $O(k \log N)$ balls.*

A randomized variant of the game will be useful in analyzing our randomized algorithm. In this variant, in addition to emptying the largest bin, Player 2 also chooses a number i from $[k]$ uniformly at random and empties the bin to which Player 1 added its i^{th} ball during its last turn. Player 1 is oblivious to the behavior of Player 2.

▶ **Theorem 2.3** ([20]). *In the randomized variant of the balls and bins game, in a game with N moves every bin always contains $O(k \log\log N + k \log k)$ balls with high probability.*[1]

Recall that ℓ is a parameter introduced in Section 2.1.

▶ **Lemma 2.4.** *In the deterministic algorithm the maximum degree of G' is always $O(\ell \log n)$. In the randomized algorithm the maximum degree of G' is always $O(\ell \log\log n + \ell \log \ell)$.*

Proof. We will argue that in the balls and bins game with $N = n$ and $k = 2\ell$, the number of balls in the largest bin is an upper bound for the maximum degree of G'. Then, applying Theorems 2.2 and 2.3 completes the proof.

We first note that by construction, the degree of each vertex v in G' is at most the recent degree of v so it suffices to bound recent degree. (In particular, the recent degree of v could be larger because it counts edges to vertices that have recently been included as input to the static algorithm.)

The only way for the recent degree of a vertex v to increase is due to the insertion of an edge incident to v. On the other hand, the recent degree of a vertex v decreases when a) an edge incident to v is deleted causing its recent degree to decrement, and b) v is included as input to the static algorithm causing its recent degree to be set to 0.

We consider the special case of the balls and bins game where for each edge insertion (u, v), Player 1 places one ball in the bin corresponding to u and one ball in the bin corresponding to v. Then, when each interval ends (which happens once every ℓ updates), Player 2 moves. Recall that at this point the recent degree of v_I is set to 0 (and in the randomized algorithm, so is that of u_I). It is clear from this description that the balls and bins game parallels all of the increases and some of the decreases in recent degree in the algorithm. From here, it is easy to verify that the number of balls in the largest bin is an upper bound for the maximum recent degree in both the deterministic and randomized settings. We omit the formal proof of this fact due to space constraints. ◀

2.4 Correctness

We will show that our algorithm produces a proper coloring after every update. Recall that the color of each vertex v is defined by the pair of colors (c_1, c_2) where c_1 is the color assigned to v by the black-box static algorithm and c_2 is the color assigned to v by the black-box dynamic algorithm.

Consider an edge (u, v) in the graph at a fixed point in time. We will show that our algorithm assigns different colors to u and v. If (u, v) is included as input to the black-box dynamic algorithm (i.e. if (u, v) is in G'), then its two endpoints are assigned different colors by this algorithm, and are thus assigned different colors by the overall algorithm.

Otherwise, by the definition of the input to the black-box dynamic algorithm, after the edge (u, v) was last inserted at least one of u or v was included as input to the static algorithm. We claim that u and v are assigned different colors by the static algorithm. If u and v were last colored by the same instance $\mathcal{A}_I$ of the static algorithm, then $\mathcal{A}_I$ was executed after the edge (u, v) was inserted (by assumption). Thus, the edge (u, v) was included as input to $\mathcal{A}_I$, causing u and v to be assigned different colors. If u and v were last colored by instances of the static algorithm on different levels, then they are assigned different colors since each level uses a separate palette of colors.

[1] "High probability" means that for all $c > 0$, there is an N such that the probability is at least $1 - N^{-c}$

The only remaining case is that u and v were last included as input to the static algorithm by two different instances of the static algorithm on the same level i. We will show that this is impossible. This case is the crux of the correctness argument and the reason that we define the intervals in precisely the way that we do. It cannot be the case that $i = 0$ since every vertex is recolored at the end of every interval on level 0. Suppose by way of contradiction that u was most recently colored by $\mathcal{A}_I$ (the instance of the static algorithm at the end of interval I) and v was most recently colored by $\mathcal{A}_{I'}$ where interval I comes before interval I' and both are on level i. We will show that between the end of interval I and the end of interval I', u is recolored by an instance of the static algorithm on a level $j < i$ (a contradiction). By the construction of the intervals (see Figure 1), between the ending points of I and I' is the end of an interval I'' on a level $j < i$ that contains interval I as a subinterval. By the definition of the algorithm, every vertex that is included as input to $\mathcal{A}_I$ is also included as input to $\mathcal{A}_{I''}$. Thus, u is recolored on level j before $\mathcal{A}_{I'}$ was executed, a contradiction.

2.5 Analysis

Static algorithm. *Number of colors.* The static algorithm uses C colors per level and there are $O(\log n)$ levels for a total of $O(C \log n)$ colors.
Number of recolorings. In the deterministic algorithm, each interval I has an associated vertex v_I and in the randomized algorithm, each interval has two associated vertices v_I and u_I. Each such vertex is included as input to the static algorithm for all superintervals of I. Since there are $O(\log n)$ levels and each level consists of a set of disjoint intervals, each interval has at most $O(\log n)$ superintervals. Thus, for each interval I, v_I and u_I are included as input to $O(\log n)$ instances of the static algorithm. Every interval ends after a multiple of ℓ updates so the number of recolorings is amortized $O(\frac{\log n}{\ell})$.

Dynamic algorithm. *Number of colors.* Given a dynamic graph of maximum degree Δ, the black-box dynamic algorithm maintains an $O(\Delta)$-coloring. By Lemma 2.4, G' (the graph input to the black-box dynamic algorithm) has maximum degree $O(\ell \log n)$ in the deterministic setting and $O(\ell \log\log n + \ell \log \ell)$ in the randomized setting. The randomized bound is with high probability and in the low probability event that the maximum degree exceeds the bound, we will immediately end all intervals, thereby recoloring the entire graph. Thus, the runtime bound is probabilistic but the bound on the number of colors is not.
Number of recolorings. Using the following simple greedy algorithm we can get constant number of recolorings per update. When an edge is added between two vertices of the same color, simply scan the neighborhood of one of them and recolor it with a non-conflicting color. If the maximum degree of the graph is Δ, this algorithm produces a $\Delta + 1$ coloring.

Combining the static and dynamic algorithms. *Number of colors.* Recall that if a vertex v is assigned color c_1 by the black-box static algorithm and color c_2 by the black-box dynamic algorithm, then our algorithm assigns v the color (c_1, c_2). So the number of colors is the product of the number of colors used in each black-box algorithm, which is $O(C\ell \log^2 n)$ for the deterministic algorithm and $O(C\ell \log n(\log\log n \log \ell))$ for the randomized algorithm.
Number of recolorings. The total number of recolorings is the sum of the number of recolorings performed in each of the black-box algorithms, which is $O(\frac{\log n}{\ell})$. Setting $\ell = \frac{\log n}{\beta}$ completes the proof.

3 Algorithm for low arboricity graphs

In this section we prove Theorem 1.6. The bulk of the argument is to prove Theorem 1.8. We omit the proof of Theorem 1.5.

▶ **Theorem 3.1** (Restatement of Theorem 1.6). *There is a fully dynamic deterministic algorithm for graphs of arboricity α that maintains an $O(\alpha \log^2 n)$-coloring in amortized $\tilde{O}(1)$ time.*

Given a partition of the vertices of a graph into layers $L_1, L_2, \ldots$, for all vertices v let $d_{up}(v)$ (the *up-degree* of v) be the number of neighbors of v in layers equal to or higher than that of v.

▶ **Definition 3.2.** Given a dynamic graph G of arboricity α, a *layered data structure (LDS)* with parameters k and Δ is a partition of the vertices into k layers $L_1, \ldots, L_k$ so that for all vertices v, $d_{up}(v) \leq \Delta$.

▶ **Theorem 3.3** (Restatement of Theorem 1.8). *Let A'' be an algorithm for arboricity α graphs that maintains an orientation of the edges with out-degree at most D that performs amortized $F(n)$ flips per update. Then there is an algorithm to maintain an LDS for a fully dynamic graph of arboricity α with $k = O(\log n)$ and $\Delta = O(D + \alpha \log n)$ in amortized deterministic time $O(F(n))$.*

3.1 Proof overview

The idea of the algorithm is essentially to move vertices to new layers when the required properties of the data structure are violated. Roughly, when there is a vertex v with $d_{up}(v) \geq \Delta$ we move v to a higher layer so that $d_{up}(v)$ decreases to $O(\alpha)$. Additionally, to control the number of layers, whenever a vertex v has up-degree less than $d = O(\alpha)$ and v can be moved to a lower layer while maintaining up-degree less than d, we move v to a lower layer. The fact that d and Δ differ by a logarithmic factor ensures that vertices don't move between layers too often which is essential for bounding the runtime.

To help with the runtime analysis, we maintain *two* dynamic orientations of the edges: one is defined by the algorithm A'' and the other is maintained by our algorithm and ensures that all edges with endpoints in different layers are oriented toward the higher layer. We compare the number of edge flips in the orientation defined by our algorithm to the number of edge flips in the orientation algorithm A'' using a potential function: $\phi(i) =$ the number of oppositely oriented edges in the two algorithms. This potential function is also used in [14].

The main idea of the analysis is to observe how ϕ changes in response to vertices moving between levels. We claim that when we move a vertex to a higher level, ϕ decreases substantially. Our algorithm is defined so that we only move a vertex to a higher layer if its up-degree decreases substantially as a result. Because our algorithm orients edges from lower to higher layers, when we move a vertex v to a higher layer many edges incident to v are flipped towards v. Then because A'' maintains an orientation of low out-degree, many of these edges flipped towards v end up oriented in the same direction in the two orientations. Thus, ϕ decreases substantially as a result of v moving to a higher layer. On the other hand, when a vertex moves to a lower layer, ϕ might increase. The idea of the argument is to use the substantial decreases in ϕ that result from moving vertices to higher layers to pay for the increases in ϕ that result from moving vertices to lower layers.

3.2 Invariants

In this section we introduce four invariants that together imply that $d_{up}(v) \leq \Delta$ and $k = O(\log n)$.

We maintain two dynamic orientations of the edges in the graph, one defined by our algorithm and the other defined by the algorithm A''. Unless otherwise stated, when we refer to an orientation, we mean the orientation defined by our algorithm.

For ease of notation, let $d = 4\alpha$ and let $d' = \Delta/2$.
We define the following for each vertex v:

- $L(v)$ is the layer containing v.
- $L_{max}(v)$ is the lowest layer for which if v were in this layer, $d_{up}(v)$ would be at most d.
- $d^+(v)$ is the out-degree of v.
- $d^-_L(v)$ is the in-degree of v from neighbors in $L(v)$.

Invariant 1 defines how edges are oriented between layers and is useful for analyzing the update time of the algorithm, as outlined in Section 3.1.

▶ **Invariant 1.** *All edges with endpoints in different layers are oriented towards the vertex in the higher layer.*

The next two invariants bound $d^+(v)$ and $d^-_L(v)$, which helps to bound $d_{up}(v)$.

▶ **Invariant 2.** *For all vertices v, $d^+(v) \leq d'$.*

▶ **Invariant 3.** *For all vertices v, $d^-_L(v) \leq d'$.*

▶ **Claim 3.4.** *Invariants 1-3 together imply that $d_{up}(v) \leq 2d' = \Delta$.*

Proof. By Invariant 1, for all vertices v, every neighbor of v in a layer equal to or higher than $L(v)$ is either an out-neighbor of v or an in-neighbor of v in $L(v)$, so $d_{up}(v) = d^+(v) + d^-_L(v)$. Then by Invariants 2 and 3, $d^+(v) + d^-_L(v) \leq 2d'$. ◀

Invariant 4 serves to bound the number of layers k.

▶ **Invariant 4.** *For all vertices v, $L(v) \leq L_{max}(v)$.*

▶ **Claim 3.5.** *Invariant 4 implies that $k = O(\log n)$.*

Proof. First we observe that under Invariant 4, all vertices of degree at most d are in L_1. Now, consider removing all vertices in L_1 from the graph. In the remaining graph, all vertices of degree at most d are in layer L_2. More generally, after removing all vertices in layers 1 through i for any i, all vertices of degree at most d must be in layer L_{i+1}.

The total number of edges in a graph of arboricity alpha is less than αn. So at least a $(1 - 2\alpha/d)$ fraction of the vertices have degree at most d. Any subgraph of an arboricity α graph also has arboricity α so after the vertices in any given layer are removed, the graph still has arboricity α. Thus, after removing the vertices in layers 1 through i for any i, at least a a $(1 - 2\alpha/d)$ fraction of the remaining vertices are in L_{i+1}. Therefore, the number k of layers total is at most $\log_{\frac{d}{2\alpha}} n = O(\log n)$. ◀

3.3 Algorithm

The idea of the algorithm is essentially to move vertices to new layers when the required properties of the data structure are violated. We define two recursive procedures RISE and DROP which move vertices to higher and lower layers respectively. In particular, when a vertex v violates Invariant 2 or 3 (i.e. either $d^+(v) > d'$ or $d^-_L(v) > d'$), we call the procedure RISE(v) which moves v up to the layer $L_{max}(v)$. The movement of v to a new higher layer may increase the up-degree of some neighbors u of v causing u to violate Invariant 2 or 3, in which case we recursively call RISE(u). On the other hand, when a vertex v violates Invariant 4 (i.e. $L_{max}(v) < L(v)$), we call the procedure DROP(v) which moves v down to the layer $L_{max}(v)$. The movement of v to a new lower layer may decrease $L_{max}(u)$ for some neighbors u of v causing u to violate Invariant 4, in which case we recursively call DROP(u).

We defer the pseudocode of the algorithm, the analysis of the algorithm, and the proof of Theorem 1.6 to the full version.

References

1 Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, January 16-19, 2017*, pages 440–452, 2017.

2 A. Andersson and Mikkel Thorup. Dynamic ordered sets with exponential search trees. *J. ACM*, 54(3):13, 2007.

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

4 Baruch Awerbuch, Michael Luby, Andrew V Goldberg, and Serge A Plotkin. Network decomposition and locality in distributed computation. In *FOCS, 1989., 30th Annual Symposium on*, pages 364–369. IEEE, 1989.

5 Luis Barba, Jean Cardinal, Matias Korman, Stefan Langerman, André van Renssen, Marcel Roeloffzen, and Sander Verdonschot. Dynamic graph coloring. In *Proceedings of the 15th International Symposium on Algorithms and Data Structures, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017*, pages 97–108, 2017.

6 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. *Distributed Computing*, 22(5-6):363–379, 2010.

7 Leonid Barenboim and Michael Elkin. Distributed graph coloring: Fundamentals and recent developments. *Synthesis Lectures on Distributed Computing Theory*, 4(1):1–171, 2013.

8 Leonid Barenboim and Tzalik Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. In *Proceedings of the International Conference on Computational Science, ICCS 2017, Zurich, Switzerland, June 12-14, 2017*, pages 89–98, 2017.

9 Edvin Berglin and Gerth Stølting Brodal. A simple greedy algorithm for dynamic graph orientation. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 92. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

10 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Part I*, pages 167–179, 2015.

11 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711, 2016.

12 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20, 2018.

13 Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. *arXiv preprint arXiv:1606.07864*, 2016.

14 G. S. Brodal and R. Fagerberg. Dynamic representation of sparse graphs. In *Proc. of 6th WADS*, pages 342–351, 1999.

15 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. *arXiv preprint arXiv:1711.06883*, 2017.

16 Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

17 Artur Czumaj and Christian Sohler. Testing hypergraph coloring. In *International Colloquium on Automata, Languages, and Programming*, pages 493–505. Springer, 2001.

18 Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. *communities*, 28:43, 2018.

19 Paul Dietz and Daniel Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 365–372. ACM, 1987.

20 Paul F Dietz and Rajeev Raman. Persistence, amortization and randomization. In *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pages 78–88. Society for Industrial and Applied Mathematics, 1991.

21 Paul F Dietz and Rajeev Raman. A constant update time finger search tree. *Information Processing Letters*, 52(3):147–154, 1994.

22 Zdeněk Dvořák and Vojtěch Tuma. A dynamic data structure for counting subgraphs in sparse graphs. In *Workshop on Algorithms and Data Structures*, pages 304–315. Springer, 2013.

23 Talya Eden, Reut Levi, and Dana Ron. Testing bounded arboricity. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2081–2092. SIAM, 2018.

24 Guy Even, Moti Medina, and Dana Ron. Deterministic stateless centralized local algorithms for bounded degree graphs. In *European Symposium on Algorithms*, pages 394–405. Springer, 2014.

25 Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Fully dynamic shortest paths in digraphs with arbitrary arc weights. *Journal of Algorithms*, 49(1):86–113, 2003.

26 Andrew V Goldberg, Serge A Plotkin, and Gregory E Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988.

27 Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

28 Bradley Hardy, Rhyd Lewis, and Jonathan Thompson. Modifying colourings between time-steps to tackle changes in dynamic random graphs. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 186–201. Springer, 2016.

29 Bradley Hardy, Rhyd Lewis, and Jonathan Thompson. Tackling the edge dynamic graph colouring problem with and without future adjacency information. *Journal of Heuristics*, pages 1–23, 2017.

30 M. He, G. Tang, and N. Zeh. Orienting dynamic graphs, with applications to maximal matchings and adjacency queries. In *Proc. 25th ISAAC*, pages 128–140, 2014.

31 Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of the 26th International Conference on World Wide Web*, pages 441–449. International World Wide Web Conferences Steering Committee, 2017.

32 Alexis Kaporis, Christos Makris, George Mavritsakis, Spyros Sioutas, Athanasios Tsakalidis, Kostas Tsichlas, and Christos Zaroliagis. Isb-tree: a new indexing scheme with efficient expected behaviour. In *International Symposium on Algorithms and Computation*, pages 318–327. Springer, 2005.

33 Subhash Khot and Ashok Kumar Ponnuswami. Better inapproximability results for max-clique, chromatic number and min-3lin-deletion. In *Proc. 33rd ICALP*, pages 226–237, 2006.

34 T. Kopelowitz, R. Krauthgamer, E. Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Proc. 41st ICALP*, pages 532–543, 2014.

35 Łukasz Kowalik. Fast 3-coloring triangle-free planar graphs. In *European Symposium on Algorithms*, pages 436–447. Springer, 2004.

36 Łukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Information Processing Letters*, 102(5):191–195, 2007.

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

38 Christos Levcopoulos and Mark H. Overmars. A balanced search tree with O (1) worst-case update time. *Acta Inf.*, 26(3):269–277, 1988.

39 Min Chih Lin, Francisco J Soulignac, and Jayme L Szwarcfiter. Arboricity, h-index, and dynamic algorithms. *Theoretical Computer Science*, 426:75–90, 2012.

40 Nathan Linial. Distributive graph algorithms-global solutions from local data. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 331–335, 1987.

41 Nathan Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

42 Michael Luby. Removing randomness in parallel computation without a processor penalty. *J. Comput. Syst. Sci.*, 47(2):250–286, 1993.

43 Cara Monical and Forrest Stonedahl. Static vs. dynamic populations in genetic algorithms for coloring a dynamic graph. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 469–476. ACM, 2014.

44 Christian Worm Mortensen. Fully dynamic orthogonal range reporting on ram. *SIAM Journal on Computing*, 35(6):1494–1525, 2006.

45 C St JA Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 1(1):12–12, 1964.

46 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the 45th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2013, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013.

47 K. Onak, B. Schieber, S. Solomon, and N. Wein. Fully dynamic mis in uniformly sparse graphs. In *Proc. 45th ICALP*, 2018.

48 Linda Ouerfelli and Hend Bouziri. Greedy algorithms for dynamic graph coloring. In *Communications, Computing and Control Applications (CCCA), 2011 International Conference on*, pages 1–5. IEEE, 2011.

49 Merav Parter, David Peleg, and Shay Solomon. Local-on-average distributed tasks. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 220–239, 2016.

50 David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, 2016.

51 Davy Preuveneers and Yolande Berbers. Acodygra: an agent algorithm for coloring dynamic graphs. *Symbolic and Numeric Algorithms for Scientific Computing (September 2004)*, 6:381–390, 2004.

52 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proc. of 1st ITCS*, pages 223–238, 2011.

53 Scott Sallinen, Keita Iwabuchi, Suraj Poudel, Maya Gokhale, Matei Ripeanu, and Roger Pearce. Graph colouring as a challenge problem for dynamic graph processing on distributed systems. In *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*, pages 347–358. IEEE, 2016.

54 Stephen B Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.

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

56 Shay Solomon. Local algorithms for bounded degree sparsifiers in sparse graphs. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

57 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2005, Baltimore, MD, USA, May 21-24, 2005*, pages 112–119, 2005.

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

59 Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Effective and efficient dynamic graph coloring. *Proceedings of the VLDB Endowment*, 11(3):338–351, 2017.

60 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.