


Sparsity-Parameterised Dynamic Edge Colouring

Aleksander B. G. Christiansen ✉ 

Technical University of Denmark, Lyngby, Denmark

Eva Rotenberg ✉ 

Technical University of Denmark, Lyngby, Denmark

Juliette Vlieghe ✉ 

Technical University of Denmark, Lyngby, Denmark

Abstract

We study the edge-colouring problem, and give efficient algorithms where the number of colours is parameterised by the graph’s arboricity, α . In a dynamic graph, subject to insertions and deletions, we give a deterministic algorithm that updates a proper $\Delta + O(\alpha)$ edge colouring in $\text{poly}(\log n)$ amortized time. Our algorithm is fully adaptive to the current value of the maximum degree and arboricity.

In this fully-dynamic setting, the state-of-the-art edge-colouring algorithms are either a randomised algorithm using $(1 + \varepsilon)\Delta$ colours in $\text{poly}(\log n, \varepsilon^{-1})$ time per update, or the naive greedy algorithm which is a deterministic $2\Delta - 1$ edge colouring with $\log(\Delta)$ update time.

Compared to the $(1 + \varepsilon)\Delta$ algorithm, our algorithm is deterministic and asymptotically faster, and when α is sufficiently small compared to Δ , it even uses fewer colours. In particular, ours is the first $\Delta + O(1)$ edge-colouring algorithm for dynamic forests, and dynamic planar graphs, with polylogarithmic update time.

Additionally, in the static setting, we show that we can find a proper edge colouring with $\Delta + 2\alpha$ colours in $O(m \log n)$ time. Moreover, the colouring returned by our algorithm has the following local property: every edge uv is coloured with a colour in $\{1, \max\{\deg(u), \deg(v)\} + 2\alpha\}$. The time bound matches that of the greedy algorithm that computes a $2\Delta - 1$ colouring of the graph’s edges, and improves the number of colours when α is sufficiently small compared to Δ .

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

Keywords and phrases edge colouring, arboricity, hierarchical partition, dynamic algorithms, amortized analysis

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.20

Related Version This paper is based on the master’s thesis of Juliette Vlieghe, supervised by Eva Rotenberg and Aleksander B. G. Christiansen and defended on 30th June 2023.

Master’s Thesis: <https://doi.org/10.13140/RG.2.2.18471.52648>

Funding This work was supported by the VILLUM Foundation grant VIL37507 “Efficient Recomputations for Changeful Problems”.

Acknowledgements We thank Jacob Holm for his interest in this work, and for comments and improvements to an earlier version of this manuscript. We also thank an anonymous reviewer for their highly detailed feedback and suggestions for improvement.

1 Introduction and related work

When working on rapidly evolving, large scale graphs, algorithms need to adapt to the change in data quickly. The dynamic model is interested in maintaining some property in a graph undergoing edge insertions and/or deletions, and has led to many fast algorithms, with polylogarithmic update and query time, in particular through the use of amortized analysis.



© Aleksander B. G. Christiansen, Eva Rotenberg, and Juliette Vlieghe; licensed under Creative Commons License CC-BY 4.0

19th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2024).

Editor: Hans L. Bodlaender; Article No. 20; pp. 20:1–20:18

Leibniz International Proceedings in Informatics



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

Graph colouring is a family of fundamental problems with many applications in computer science. We study the edge-colouring problem: the goal is to assign edges colours such that edges sharing an endpoint are coloured differently. This problem has implications in resource allocation and scheduling, for example to allocate bandwidth in an optical network [17].

A C -edge colouring of a graph $G = (V, E)$ can be represented by a function $f : E \rightarrow \{1, \dots, C\}$, and the smallest palette size C for which there exists a proper C edge colouring is called the edge chromatic number of G , denoted χ' . If Δ is the maximum degree of G , then the edge chromatic number is clearly at least Δ . Vizing [30] proved that G can always be coloured with $\Delta + 1$ colours. On the other hand, Holyer showed that it is NP-complete to determine the edge chromatic index of an arbitrary graph [21], and the problem remains NP-complete even for cubic graphs. A $(\Delta(uv) + C)$ -edge colouring is a proper colouring of the graph where each edge uv receives a colour from $\{1, \dots, \Delta(uv) + C\}$. Here $\Delta(uv) = \max\{d(u), d(v)\}$.

Vizing's proof is constructive and suggests a way to extend a proper partial colouring to a larger subgraph by recolouring $O(\Delta + n)$ edges. Furthermore, the colour changes can be performed in polynomial time. However, so far the fastest algorithms for statically $(\Delta + 1)$ edge colouring a graph spend $O(m\sqrt{n})$ [28] or $\tilde{O}(m\Delta)$ [18] time. For certain graphs, faster algorithms are known [4, 8].

It is interesting to see whether one can reduce the running time by slightly increasing the palette size. This line of research has been pursued before. In particular, the problem of $2\Delta - 1$ edge colouring can be solved greedily, yielding static algorithms running in near-linear time [14] and dynamic algorithms with $O(\log \Delta)$ update time [6]. In this dynamic setting, there are known algorithms [13, 16] that achieve a randomized $(1 + \epsilon)\Delta$ colouring in $\text{poly}(\log n, \epsilon^{-1})$ time, with $\epsilon > 0$ by Duan, He, and Zhang [16], and later Christiansen [13].

In the distributed setting, Chang, He, Li, Pettie and Uitto [11] designed a randomized $\Delta + O(\sqrt{\Delta})$ edge colouring in $\text{poly}(\log n)$ rounds based on the Lovasz Local Lemma and Barenboim, Elkin and Maimon [2] describe a simple deterministic distributed algorithm for $\Delta + O(\alpha)$ colouring in polylogarithmic time. There are many more papers achieving different trade-offs between time and palette size. See for instance [3, 15, 19, 29] for different trade-offs in the distributed setting, or the papers [11, 19] for a more extensive discussion.

For some algorithmic problems, especially ones where recourse is an important part of the running time of an algorithm, or the recourse is of interest on its own, the best known analysis follows a specific proof strategy; “*solution oblivious analysis*”. By solution oblivious, we mean that we do not only give guarantees against worst case input graphs at each step of the algorithm, we furthermore always, when analysing the next step, are robust against an adversary changing the solution to the worst-possible solution before every update. Examples of such analysis include the analysis of the SAP protocol for maintaining maximum matchings in bipartite graphs of Bernstein, Holm and Rotenberg [5], the analysis of recourse in the edge-colouring algorithms of Bernshteyn [3], Christiansen [13], and Duan, He and Zhang [16], and the analysis of the fully-dynamic out-orientation scheme due to Brodal & Fagerberg [10], in which a potential function bounds the reorientations by comparing to an *existing* (but not necessarily efficient) algorithm that augments paths whose lengths are bounded in an oblivious manner.

For the edge-colouring problem, an interesting lower bound has been proved by Chang, He, Li, Pettie and Uitto: they show that there exists a graph, and a partial colouring of this graph with a single uncoloured edge, such that to colour this edge, one needs to recolour a subgraph of diameter $\Omega(\frac{\Delta}{c} \log(\frac{cn}{\Delta}))$. This is then also a lower bound on the number of edges that need to be recoloured. This means that if we restrict ourselves to solution oblivious analysis, a dynamic algorithm with polylogarithmic update time will need a palette of size

at least $\Delta + O\left(\frac{\Delta}{\text{poly}(\log n)}\right)$. The analysis of Christiansen [13], and Duan, He and Zhang [16] for their $(1 + \epsilon)\Delta$ dynamic edge-colouring algorithms are solution oblivious, which results in algorithms that have a polynomial dependency on ϵ^{-1} .

Whether one can design an algorithm with poly log update time that only uses $\Delta + O(\Delta^{1-\epsilon})$ colours for some constant $\epsilon > 0$ remains a fundamental open problem. Improved results for special classes of graphs, like forests, planar graphs or sparse graphs, also receive attention in the community. In the static setting, it was shown by Bhattacharya, Costa, Panski and Solomon [8] that one can compute a $(\Delta + 1)$ edge colouring in $\tilde{O}(\min\{m\sqrt{n}, m\Delta\} \cdot \frac{\alpha}{\Delta})$ -time. Here α is the *arboricity* of the graph, and it is equal to the smallest number of forests needed to cover the edges of a graph. It is within a factor of 2 of other sparsity measures like the maximum subgraph density and the degeneracy. Many other problems like, for instance, maintaining dynamic matchings [22, 26], maintaining a dynamic data structure that can answer adjacency queries efficiently [10] and maintaining a maximal independent set [25] also have solutions that run faster in graphs with low arboricity.

Our contribution. A natural question is therefore to ask if one can further reduce the palette size in dynamic graphs that are at all times sparse. In this paper, we show that this is the case. More specifically, we show that there exists a dynamic algorithm that can maintain an edge colouring with only $\Delta + O(\alpha)$ colours in poly-logarithmic update time. Since the arboricity can be as large as $\frac{\Delta}{2}$, this is not always an improvement, however for many graph classes like forests, planar graphs, and graphs with constant arboricity, the number of colours used is significantly reduced compared to other efficient dynamic edge-colouring algorithms.

Tables 1, 2 and 3 summarise the results mentioned above and are not a comprehensive overview of the state of the art.

■ **Table 1** A comparison of static edge-colouring algorithms.

Palette size	Time	Notes	Reference
Δ	$O(m \log \Delta)$	bipartite multigraph	[14]
$2\Delta(uv) - 1$	$O(m \log \Delta)$		[6]
$\Delta + 1$	$O(m\sqrt{n})$	randomised	[28]
$\Delta + 1$	$\tilde{O}(m\Delta)$		[18]
$\Delta + 1$	$\tilde{O}(\min\{m\sqrt{n}, m\Delta\} \cdot \frac{\alpha}{\Delta})$		[8]
$\Delta(uv) + 1$	$O(n^2 \Delta)$		[13]
$\Delta(uv) + 2\alpha - 2$	$O(m \log \Delta)$		new

Independent work. In independent and concurrent work, Bhattacharya, Costa, Panski, and Solomon also maintain a $\Delta + O(\alpha)$ edge colouring in amortised polylogarithmic time per insertion or deletion [7].

20:4 Sparsity-Parameterised Dynamic Edge Colouring

■ **Table 2** A comparison of dynamic edge-colouring algorithms. If G goes through a sequence of insertions and deletions $G_1 \dots G_T$, $\Delta_{max} = \max_{1 \leq t \leq T} \Delta(G_t)$ is the maximum Δ on all graphs in the sequence. α_{max} is defined similarly.

Palette size	Update time	Notes	Reference
$2\Delta(uv) - 1$	$O(\log \Delta)$	worst case	[6]
$(1 + \epsilon)\Delta$	$O(\log^9 n \log^6 \Delta / \epsilon^6)$	worst-case, randomised	[13]
$(1 + \epsilon)\Delta$	$O(\log^8 n / \epsilon^4)$	amortized, randomised $\Delta \in \Omega(\log^2 n / \epsilon^2)$	[16]
$\Delta_{max} + O(\alpha_{max})$	$O(\log n \log \Delta_{max})$	amortized	new
$\Delta(uv) + O(\alpha)$	$O(\log^2 n \log \alpha_{max} \log \alpha \log \Delta_{max})$	amortized	new

■ **Table 3** State-of-the-art for edge-colouring algorithms in the LOCAL model.

Palette size	Rounds	Notes	Reference
$\Delta + O(\alpha)$	$O(\sqrt{\alpha} \log n)$	LOCAL model	[2]
$\Delta + 1$	$poly(\Delta, \log n)$	LOCAL model	[3]

1.1 Notations

In this paper, we focus on simple graphs. According to many definitions of edge colouring, an edge from a vertex to itself can not be coloured.

Let $G = (E, V)$ be the undirected input graph and H a subgraph of G . Define $\Gamma_H(v)$ to be the neighbourhood of v with respect to a graph H , and given a subset of vertices $U \subseteq V$, define $\Gamma_U(v)$ to be the neighbourhood of v with respect to the subgraph induced by U in G . Later, we use $N_H(v)$ to refer to a data structure containing $\Gamma_H(v)$. Define $\deg_H(v)$ to be the degree of v with respect to a graph H . Formally, given $H \subseteq G$ and $U \subseteq V$:

$$\begin{aligned} \Gamma_H(v) &= \{u \in V \mid (u, v) \in E(H)\} \\ \Gamma_U(v) &= \{u \in U \mid (u, v) \in E(G)\} \\ \deg_H(v) &= |\Gamma_H(v)| \\ \deg_U(v) &= |\Gamma_U(v)| \end{aligned}$$

Let G go through a sequence of insertions and deletions $G_1 \dots G_T$. At any iteration t , we denote by $\Delta_t(uv)$ the maximum degree of the endpoints of the edge uv and Δ_t the maximum degree of the graph considered. Δ_{max} is the maximum Δ on all graphs in the sequence. Formally, at iteration t :

$$\begin{aligned} \Delta_t(uv) &= \max\{\deg_{G_t}(u), \deg_{G_t}(v)\} \\ \Delta_t &= \Delta(G_t) \\ \Delta_{max} &= \max_{1 \leq t \leq T} \Delta(G_t) \end{aligned}$$

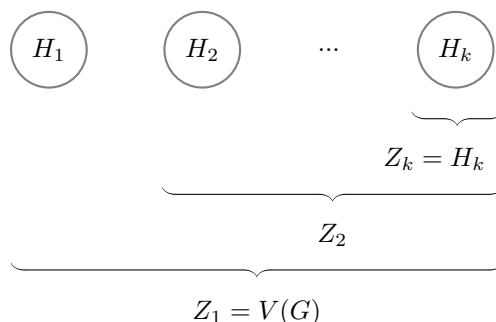
If the context is not ambiguous, we drop the subscript and write $\Delta, \Delta(uv)$ to refer to the maximum degree and the maximum degree between u and v in the current iteration.

Arboricity. The arboricity α of a graph $G = (V, E)$ is defined as:

$$\alpha = \max_{U \subseteq V, |U| > 1} \left\lceil \frac{|E(U)|}{|U| - 1} \right\rceil$$

On a more intuitive level, the arboricity can also be defined as the smallest number α such that the edges of the graph can be partitioned in α forests. The two definitions are equivalent by Nash-Williams theorem [24]. A relevant consequence is that there exists an orientation of G where each vertex has at most α out-neighbours.

H-partition. Let $H = \{H_1, \dots, H_k\}$ be a partition of the vertex set $V(G)$. If a vertex v is in H_i , we say that the level of v is $l(v) = i$. We denote by $Z_i = \bigcup_{j \geq i} H_j$ the vertices in levels i and above (Figure 1). We may abuse these notations and use H_i, Z_i to refer to the subgraph induced in G by those sets.



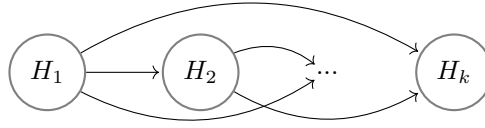
■ **Figure 1** Hierarchical partition of the vertex set of a graph G .

Orientation. Consider the following orientation of the graph: if $l(u) < l(v)$, we orient the edge from u to v (Figure 2). If u and v are on the same level, we have an edge in both directions. This orientation enables us, given a vertex v , to refer to the neighbours of v in $Z_{l(v)}$ as the out-neighbours of v . We denote by $\deg^+(v)$ the out-degree of v .

$$\deg^+(v) = \deg_{Z_{l(v)}}(v)$$

1.2 Palettes

In the following, a palette is a data structure that keeps track of the colours that are used or not at a vertex. They cover colours $[1, 2^{\lceil \log(2\Delta-1) \rceil}]$, where the value of Δ can change. In the dynamic algorithms, we will use the following result:



■ **Figure 2** Hierarchical partition. In the static setting, the out degree of a vertex is bounded by d .

► **Theorem 1** (Palettes). *Consider two palettes P, Q such that there are a colours used in P and b colours used in Q . We can find a colour that is available in $[1, a + b + 1]$ in $\log \Delta$ time.*

Proof. The proof is in the full version of the paper [12], along with the description of the data structure, which is generalised from the palettes in [6]. ◀

1.3 Roadmap

Barembaum, Elkin and Maimon describe a simple distributed algorithm for $\Delta + O(\alpha)$ colouring [2]. They form a hierarchical partition (or H-partition) of the graph, colour each set greedily, then colour the edges going out of the sets in an appropriate order. In Section 2, we show that this technique can easily be adapted into an efficient algorithm in the static setting. We also show a simpler algorithm that yields a $\Delta + 2\alpha - 2$ edge colouring within the same time by building a degeneracy order of the graph instead of an H-partition.

In Sections 3 and 4, we maintain a valid edge colouring in poly logarithmic time. We first present a simplified version of our algorithm that maintain a fully dynamic $\Delta_{max} + O(\alpha_{max})$ edge colouring in Section 3. This relies on two ideas: first, we maintain a dynamic H-partition, which only requires simple changes from the decomposition of Bhattacharya, Henzinger, Nanongkai, and Tsourakakis [9], namely, we need to maintain two palettes of available colours at each vertex, one for its neighbours and one for its out-neighbours. Then it is easy to colour an edge in a valid partition.

Then we present an algorithm that is adaptative to the maximum degree and arboricity and maintains a $\Delta(uv) + O(\alpha)$ edge colouring. Our data structure is derived from the Level Data Structure of Henzinger, Neumann and Wiese [20], which has the following property: instead of having an out-degree that depends on α , the levels of the partition have an increasingly large out-degree, and the sequence of maximum out-degree is such that it will be bounded by the current value of the arboricity, within a constant factor. Adapting to the current maximum can be done by updating the few problematic neighbours of a vertex when its degree decreases.

In the full version of the paper [12], we study the constants, and show that with small modifications of the data structure, we can get $\Delta_{max} + (4 + \epsilon)\alpha_{max}$ and $\Delta(uv) + (8 + \epsilon)\alpha$ colours in the dynamic setting.

2 Static $\Delta(uv) + O(\alpha)$ colouring

In this section, we describe a static edge-colouring algorithm. We arrange the vertices in a hierarchical partition that results in a $O(\alpha)$ out-orientation, then we colour vertices from right to left, so that for one of the endpoints, only the out-edges may already have colours. As the other endpoint has at most Δ coloured edges, the algorithm results in a $\Delta + O(\alpha)$ edge colouring.

The partition as it will be a crucial part of the dynamic algorithm. However, in the static setting, we could perform this algorithm with any α out-orientation. We show how this leads to a $\Delta + 2\alpha - 2$ edge colouring within the same asymptotic running time. We describe this algorithm first.

Arboricity and degeneracy. A graph G of arboricity α can be partitioned into α forests has at most $\alpha(n-1)$ edges, therefore it has a vertex of degree at most $2m/n \leq 2\alpha - 1$: the degeneracy of G is less than twice the arboricity. Therefore, the degeneracy is a 2-approximation of the arboricity. We can compute a degeneracy order of the graph, in linear time [23], and colour the edges as follows: we colour the out-edges of the vertices from right to left. We describe this in more details in the following proof.

► **Theorem 2.** *Given a graph of arboricity α , we can compute a $\Delta(uv) + 2\alpha - 2$ colouring of the graph in $O(m \log \Delta)$ time.*

Proof. For this proof only, let $d \leq 2\alpha - 1$ denote the degeneracy of the graph. We compute a degeneracy ordering, which can be done in linear time [23]. We greedily colour the edges from v_i to $v_{j>i}$ for $i = n-1, \dots, 1$. Let us prove by induction that we can always find an available colour in a palette of size $\Delta(uv) + d - 1$. At the first iteration, we can greedily colour a potential edge from v_{n-1} to v_n with one colour.

Assume that for $i < n-1$, we have coloured any edge $v_j v_{j'}$ s.t. $j, j' > i$ with at most $\Delta(v_j v_{j'}) + d - 1$ colours. Consider an uncoloured edge $e = v_i v_j$ with $j > i$. v_j is incident to at most $\deg(v_j) - 1$ coloured edges and v_i is adjacent to at most $d - 1$ coloured edges, therefore e sees at most $\deg(v_j) + d - 2$ colours and can find an available colour in a palette of size $\deg(v_j) + d - 1$.

If we maintain a binary tree over the palette at each vertex, we can find an available colour for an edge in $O(\log \Delta)$ time according to Theorem 1. Therefore, colouring all the edges takes $O(m \log \Delta)$ time. ◀

This could conclude the static version. However, this algorithm does not translate into an efficient dynamic algorithm, as far as we can say; therefore we introduce the notion of hierarchical partition, which will be crucial in our dynamic algorithms. The rest of the section proves a slightly weaker theorem through the use of such a partition:

► **Theorem 3.** *We can compute a static $\Delta(uv) + O(\alpha)$ edge colouring in $O(m \log \Delta)$ time.*

H-partition. In the following, we want to compute a partition such that in the corresponding orientation, each vertex has out-degree at most $O(\alpha)$. Barenboim, Elkin and Maimon H-partition from [1] describe a distributed algorithm which translates well to the static setting: we call a vertex *active* if it does not have a level assigned yet. Initially, all the vertices are active. A vertex that is active at iteration i will be part of the set Z_i . We call active degree the number of active neighbours of a vertex. Initially, the active degree of a vertex is its degree. Then at iterations $i = 1 \dots k$, we group all the vertices of active degree at most $d = 4\alpha$ into a set H_i . For each vertex in H_i , we decrement the active degree of its neighbours and continue.

► **Lemma 4.** *We can compute a H-partition $H = \{H_1, \dots, H_k\}$ of a graph G of arboricity α such that the size of the partition is at most $k = \lceil \log n \rceil + 1$ and for all i , for all $v \in H_i$, the degree of v in $G(Z_i)$ is at most 4α .*

Proof. Consider a set H_i . The average degree is:

$$2 \frac{|E(Z_i)|}{|V(Z_i)|} \leq 2\alpha \leq \frac{4\alpha}{2}$$

20:8 Sparsity-Parameterised Dynamic Edge Colouring

At most half of the vertices have a degree more than 4α , otherwise we would have an average degree greater than 2α , leading to a contradiction. Therefore:

$$\begin{aligned} |Z_{i+1}| &\leq |Z_i|/2 \\ |Z_i| &\leq n/2^{i-1} \\ |Z_{\lceil \log n \rceil + 1}| &< 1 \end{aligned}$$

Therefore, we can safely set $k = \lceil \log n \rceil$ ◀

► **Lemma 5.** *We can compute the H-partition described in lemma 4 in $O(m)$ time.*

Proof. The sum of the degrees is $2m$, so we cannot decrement the active degrees more than $O(m)$ time in total, and decrementing a degree takes constant time. Then, at iteration i , we can search the vertices of low active degree in $O(n/2^i)$ time. Therefore, the running time to compute the H-partition is $O(m + n)$. ◀

► **Lemma 6.** *Given a H-partition, we can compute a $\Delta(uv) + O(\alpha)$ colouring of the graph in $O(m \log \Delta)$ time.*

Proof. Let $d = 4\alpha$.

We greedily colour the edges from H_i to $H_{j \geq i}$ for $i = k, \dots, 1$. Let us prove by induction that we can always find an available colour in a palette of size $\Delta(uv) + d - 1$. At the first iteration, we can greedily colour H_k with $2d - 1$ colours.

Assume that for $i < k$, we have coloured $G(H_{i+1} \cup \dots \cup H_k)$ with at most $\Delta + d - 1$ colours. Consider an uncoloured edge $e = uv$ such that $u \in H_i, v \in H_i \cup \dots \cup H_k$. v is incident to at most $\deg(v) - 1$ coloured edges and u is adjacent to at most $d - 1$ coloured edges, therefore e sees $\deg(v) + d - 2$ colours and can find an available colour in a palette of size $\deg(v) + d - 1$.

If we maintain a binary tree over the palette at each vertex, we can find an available colour for an edge in $O(\log \Delta)$ time according to theorem 1. Therefore, colouring all the edges given the H-partition takes $O(m \log \Delta)$ time using theorem 1. ◀

3 Dynamic $\Delta_{max} + O(\alpha_{max})$ colouring

In this section, we show how to update a dynamic edge colouring. We first discuss how we can recolour an edge within a valid H-partition, then we show how we can maintain such a partition. The algorithm requires α_{max} and Δ_{max} to be known in advance.

3.1 Data structure

We consider a H-partition that maintains the following invariants, with $d = 4\alpha_{max}$:

1. Each vertex v such that $l(v) > 1$ has at most βd neighbours in $Z_{l(v)}$ (out-neighbours). In this section, we choose $\beta = 5$.
2. Each vertex v has at least d neighbours in $Z_{l(v)-1}$

For each vertex $v \in H_i$, we store the following:

- For each $j < i$, we store the neighbours of v at level j , $\Gamma_{H_j}(v)$, in a linked list $N_{H_j}(v)$.
- We store the out-neighbours of v , $\Gamma_{Z_i}(v)$, in a linked list $N_{Z_i}(v)$.
- We store the length of each linked list.
- For each edge uv , we store a pointer to the position of u in the appropriate neighbour list $N.(v)$ and conversely.
- We store two palettes: one for the neighbours of v and one for its out-neighbours. We refer to those palettes as $P_G(v)$ and $P_{Z_i}(i)$.

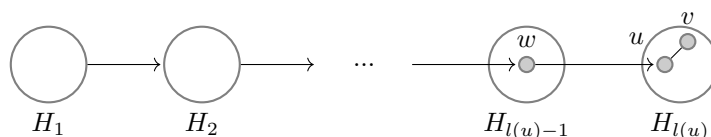
3.2 Recolouring an edge

Let us start with a key sub problem: given an uncoloured edge in an otherwise valid data structure, what is the cost of colouring the edge? We colour the edge uv as we would have in section 2.

Let u be the leftmost vertex, i.e. $i = l(u) \leq l(v)$. We ignore the neighbours of u that have a lower level, which can be done in practice by searching for an available colour in $P_{Z_i}(u) \cap P_G(v)$. The colour picked may conflict with a single edge from a neighbour of u on a lower level. If that is the case, we recolour that edge in the same way.

► **Lemma 7.** *An uncoloured edge uv in an otherwise valid data structure can be coloured in $O(\min(l(u), l(v)) \cdot \log \Delta)$ time.*

Proof. Assume wlog. $l(u) \leq l(v)$. If uv is not coloured, at most $\beta d - 1$ colours are represented at u in the palette $P(Z_{l(u)})$, and at most $\deg(v) - 1$ colours are represented at v . Therefore, there exists an available colour in the palette $[\deg(v) + \beta d - 1]$. If there exists w such that uw conflict with uv , then $l(w) < l(u)$. Therefore, the level of the left endpoint of the conflicting edge decrease at each iteration, which can happen at most $l(u) - 1$ times (Figure 1). Therefore, recolouring an edge and recursively resolving conflicts take $O(l(u) \cdot \log \Delta)$ time. ◀



■ **Figure 3** We may need to recolour at most $l(u)$ edges.

■ Algorithm 1 Recolour.

```

procedure RECOLOUR( $uv$ )
  if  $l(u) > l(v)$  then
     $u, v = v, u$ 
   $i = l(u)$ 
  // Pick a colour available in  $G(Z_i)$  in the palette  $[\deg(v) + \beta d - 1]$ .
  COLOUR( $uv, \deg(v) + \beta d - 1, P_{Z_i}(u), P_G(v)$ )
  if  $c(uv)$  is represented at  $u$  then
    Use the pointer  $C(u)[c]$  to find the conflicting edge  $wu$ 
    RECOLOUR( $wu$ )

```

3.3 Updating the hierarchical partition and full algorithm

Updates. The algorithm for the updates is the following: let us say that a vertex that violates Invariant 1 or 2 is *dirty*. As long as we have a dirty vertex v : if v violate the first invariant, we increment its level. If it violates Invariant 2, we decrement it. when doing so, we need to update our linked lists of neighbours and our tree palettes. Updating the trees is the limiting factor. The details of the updates are described in Algorithm 5. The algorithm terminates, which will be justified later, as we will define a positive potential that strictly decreases at each step.

20:10 Sparsity-Parameterised Dynamic Edge Colouring

► **Lemma 8.** *Increasing the level of a vertex takes $O(\deg^+(v) \log \Delta)$ time. Decreasing the level of a vertex takes $O(d \log \Delta)$ time.*

Proof. When we increment the level of a vertex, we first update the lists of neighbours of v : we traverse $N_{Z_i}(v)$ to split it in $N_{H_i}(v)$ and $N_{Z_{i+1}}(v)$ in $O(\deg^+(v))$ time. Then we discard $P_{Z_i}(v)$ and create $P_{Z_{i+1}}(v)$: we traverse the linked list $N_{Z_{i+1}}(v)$ and insert the $\deg_{Z_{i+1}}(v) = O(\deg^+(v))$ elements in the tree, which takes $O(\deg^+(v) \log \Delta)$ time. We also need to update the data structures of at most $\deg^+(v)$ neighbours, which takes constant time per neighbour for the linked lists and $\log \Delta$ time per neighbour for trees.

Finally, we need to check which vertices became dirty as a result of the operation. The set Z_{i+1} has one more element, and for $j \neq i+1$, Z_j is unchanged. Therefore, we check if v itself, or any vertex in $N_{Z_{i+1}}(u)$, breaks the first invariant, which takes $O(\deg_{Z_{i+1}}(v))$ time.

The procedure for decrementing a level is similar. If we decrement the level of a vertex, we know that the second invariant was not respected, i.e. $\deg_{Z_{i-1}}(v) < d$. After the operation, Z_i does not include v any more and for $j \neq i$, Z_j is unchanged. To update the lists of neighbours of v , we merge $N_{H_{i-1}}(v)$ and $N_{Z_i}(v)$, which takes $O(\deg_{Z_{i-1}}(v)) = O(d)$ time. We create the palette of v for the set Z_{i-1} in $O(\deg_{Z_{i-1}}(v)) \log \Delta = O(d \log \Delta)$ time. Then we need to update the neighbour lists and the palettes of the neighbours of u in Z_i , which also takes constant time per neighbour for the lists and $O(\log \Delta)$ time per neighbour for the palettes. Finally, we check if v or any of its neighbours in Z_i is dirty, which takes constant time per vertex. ◀

■ **Algorithm 2** Insert the edge uv .

```

procedure ADD( $uv$ )
  if  $l(u) > l(v)$  then
    ADD( $vu$ )
  End procedure.

  Add  $v$  to the out-neighbours of  $u$  and store the corresponding pointer.
  if  $l(u) < l(v)$  then
    Add  $u$  to  $N_{H_{l(u)}}(v)$ , the neighbours of  $v$  at level  $l(u)$ 
    Store the corresponding pointer.
  else
    Add  $u$  to the out-neighbours of  $v$  and store the corresponding pointer.
  // The palettes will be updated when  $uv$  gets a colour.
  Check if  $u$  or  $v$  became dirty.
  RECOVER
  RECOLOUR( $uv$ )

```

► **Theorem 9.** *We can maintain a dynamic $\Delta_{max} + O(\alpha_{max})$ edge colouring of a graph in $O(\log n \log \Delta_{max})$ amortized update time.*

Proof. We define the following potential.

$$B = \log \Delta_{max} \sum_{v \in V} \phi(v) + \log \Delta_{max} \sum_{e \in E} \psi(e)$$

$$\phi(v) = \sum_{j=1}^{l(v)-1} \max(0, \beta d - \deg_{Z_j}(v))$$

$$\psi(u, v) = 2(k - \min(l(u), l(v))) + \mathbf{1}_{l(u)=l(v)}$$

■ **Algorithm 3** Delete the edge uv .

```

procedure DELETE( $uv$ )
  if  $l(u) > l(v)$  then
    DELETE( $vu$ )
    End procedure.

  Remove the colour of  $uv$  from the palettes of neighbours of  $u$  and  $v$ .
  Remove the colour of  $uv$  from the palette of out-neighbours of  $u$ .
  if  $l(u) == l(v)$  then
    Remove the colour of  $uv$  from the palette of out-neighbours of  $v$ .
    Remove  $v$  from the neighbours of  $u$  using the corresponding pointer.
    Remove  $u$  from the neighbours of  $v$  using the corresponding pointer.
    Check if  $u$  or  $v$  became dirty.
  RECOVER

```

■ **Algorithm 4** Recursively fix the invariants.

```

procedure RECOVER
  while there exists a dirty vertex  $v$  do
     $i = l(v)$ 
    if  $\deg^+(v) > \beta d$  then INCREMENT( $v$ )
    else if  $\deg_{Z_{i-1}}(v) < d$  then DECREMENT( $v$ )

```

When we insert an edge, we create a potential $\psi(u, v) \leq 2k$. The potential of the other edges do not change and the potential of a vertex can only decrease, therefore the potential B increase by at most $2k \log \Delta_{max}$. When we delete an edge, $\phi(u)$ and $\phi(v)$ increase by at most k each, $\psi(u, v)$ is deleted, and the other potentials are not affected, therefore the potential increase by at most $2k \log \Delta_{max}$.

When a dirty vertex increment its level, the cost of the operation will be paid by the drop in potential from the edges. When a dirty vertex decrements its level, the cost of the operation is paid by the drop in potential from the vertex, despite the increase in potential from the edges. For completeness, we repeat the details of the analysis that follows closely that of [9]. Let i be the level of v before the operation.

Incrementing the level of a dirty vertex

- When $l(v)$ increases, we get $l(v) = i + 1$, so we add $\max(0, \beta d - \deg_{Z_i}(v))$ to $\phi(v)$. As invariant 1 was violated, we must have had $\deg_{Z_i}(v) > \beta d$ and therefore $\max(0, \beta d - \deg_{Z_i}(v)) = 0$: the potential of v is unchanged. The potential of the other vertices can not increase (though it might decrease for a neighbour of v).
- The potential of an edge may only change if one of the endpoints is v and the other endpoint u verifies $l(u) \geq i$. Therefore, there are exactly $\deg^+(v)$ edges whose potential drop by one or two.

The total drop in potential is at least:

$$\deg^+(v) \log \Delta_{max} = \Omega(\deg^+(v) \log \Delta)$$

■ **Algorithm 5** Incrementing / decrementing the level of a vertex.

procedure INCREMENT(v)

$i = l(v)$.

Split $N_{Z_i}(v)$ in $N_{H_i}(v)$ and $N_{Z_{i+1}}(v)$.

Create the palette of v for Z_{i+1} .

Discard the palette of v for Z_i .

for $u \in N_{Z_{i+1}}(u)$ **do**

Using the pointer in uv , remove v from $N_{H_i}(u)$, add v to $N_{Z_{i+1}}(u)$ or $N_{H_{i+1}}(u)$.

Update the pointers accordingly.

if $l(u) = i + 1$ **then**

Update the palette of u for the set Z_{i+1} : add colour $c(uv)$.

Increment $l(v)$

procedure DECREMENT(v)

$i = l(v)$.

Merge $N_{Z_i}(v)$ and $N_{H_{i-1}}(v)$ into $N_{Z_{i-1}}(v)$.

Create the palette of v for Z_{i-1} .

Discard the palette of v for Z_i .

for $u \in N_{Z_i}(v)$ **do**

Move v from $N_{H_i}(u)$ or $N_{Z_i}(u)$ to $N_{H_{i-1}}(u)$.

Update the pointers accordingly.

if $l(u) = i$ **then**

Update the palette of u for the sets Z_i : remove colour $c(uv)$.

Decrement $l(v)$

Decrementing the level of a dirty vertex

- We must have $\deg_{Z_{i-1}}(v) < d \Rightarrow \max(0, \beta d - \deg_{Z_{i-1}}(v)) \geq (\beta - 1)d = 4d$. Therefore, the $\phi(v)$ drops by at least $4d$.
- If u is a neighbour of v , $\deg_{Z_j}(v)$ is decremented if $j = i$ and is unchanged otherwise. This affects the potential of u if $l(u) > i$. Therefore, $\sum_{u \in \Gamma(v)} \phi(u)$ increase by at most $\deg_{Z_{i+1}}(v) < \deg_{Z_{i-1}}(v) < d$.
- The potential of an edge may only change if one of the endpoints is v and the other endpoint u verifies $l(u) \geq i$. Therefore, there are exactly $\deg^+(v) \leq \deg_{Z_{i-1}}(v) \leq d$ edges whose potential increase by one or two.

The total drop of potential is at least:

$$\log \Delta_{max} (4d - d - 2d) = d \log \Delta_{max} = \Omega(d \log \Delta) \quad \blacktriangleleft$$

4 Dynamic $\Delta(uv) + O(\alpha)$ colouring

The data structure from the previous section could maintain a dynamic $\Delta_{max} + O(\alpha_{max})$ colouring. We modify it further to create a data structure that adapts to the maximum degree and arboricity. We adapt the arboricity by making a structure that does not depend explicitly on α : instead, we give the level increasingly large out-degrees and show how the out-degree ends up being bounded by the current value of the arboricity within a constant factor. To adapt to the maximum degree, we locally adapt to the degree of the vertices: when the degree of a vertex decrease, we can recolour its problematic edges in polylogarithmic time.

4.1 Data Structure

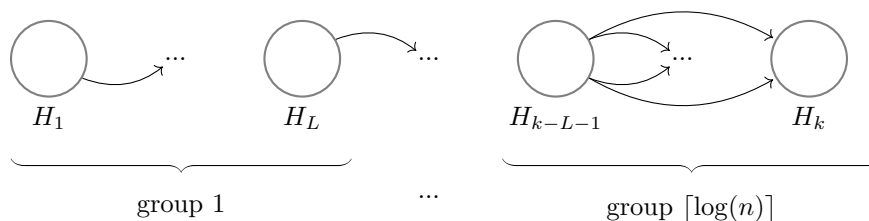
In the following, $L = 1 + \lceil \log n \rceil$. We now consider a H-partitions with $k = L \cdot \lceil \log n \rceil$ levels. Let $k' = \max_{v \in V} l(v)$ denote the maximum level of any vertex, that is, the highest non-empty level.

We partition the levels into $\lceil \log n \rceil$ groups of size L (Figure 4).

Let $g(v)$ denote the group of a vertex and $d(v) = 2^{g(v)}$. We will maintain the following:

1. Each vertex v has at most $2\beta d(v)$ neighbours in $Z_{l(v)}$. In this section, we choose $\beta = 5$, so any vertex has at most $10d(v)$ out-neighbours.
2. Each vertex v has at least $d(v)$ neighbours in $Z_{l(v)-1}$
3. The colour of an edge uv is chosen from the first $\Delta(uv) + 2\beta d(v)$ colours of the palette.

This condition enables the data structure to adapt to changes in the arboricity. In the following, we prove that for any vertex, $g(v) \leq \lceil \log(4\alpha) \rceil$, which will result in a $\Delta(uv) + O(\alpha)$ colouring.



■ **Figure 4** Hierarchical partition with two levels. The in degree of a vertex v is still only bounded by Δ , when the bound on the out degree depends on its group $g(v)$.

We store the neighbours and palettes of the vertices as described in the previous section.

► **Lemma 10** (Maximum level). *The index of the highest non empty level, k' , is at most $O(\log \alpha \log n)$.*

Proof. Consider the group $\lceil \log(4\alpha) \rceil$ and a level i in this group. We can repeat the arguments from the proof of lemma 4 and show that the last set of the group $Z_{L \lceil \log(4\alpha) \rceil}$, with $L = 1 + \lceil \log n \rceil = O(\log n)$, has at most one element, therefore, all the higher groups are empty. ◀

► **Lemma 11.** *An uncoloured edge uv in an otherwise valid data structure can be coloured in time $O(\log \alpha \log n \log \Delta)$.*

Proof. Following the same reasoning as in the previous section, we have to recolour $O(k')$ edges, which we can do in $O(k' \log \Delta)$ time. ◀

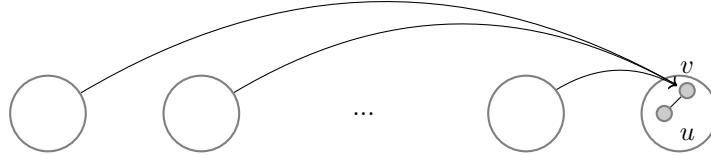
4.2 Updating the hierarchical partition and full algorithm

When edges are deleted, the arboricity α or the maximum degree Δ may decrease. To adapt to the degree, we recolour the edges from in-neighbours of v when uv is deleted. To maintain the arboricity, we will recolour the edges to the out-neighbours of v when its level decrease. As a result, we maintain that an edge uv , $l(u) \leq l(v)$, has a colour from the palette $[\deg(v) + \beta d(u) - 1]$

► **Lemma 12** (Adapting to the maximum degree). *We can recolour the edges from the in-neighbours of v in $O(\log n \log^2 \alpha \log \Delta)$ time.*

20:14 Sparsity-Parameterised Dynamic Edge Colouring

Proof. In each group $i \leq g(v)$, v might have at most one neighbour u such that uv has colour $\deg(v) + 2\beta d(u) - 1$. For each such group, it takes constant time to find this edge using the pointer $C(v)[\deg(v) + 10 \cdot 2^i - 1]$. There are therefore at most $g(v)$ edges that we may have to recolour, each in $O(l(v) \log \Delta)$ time, when the degree of v decreases. When we delete an edge, we do this for each of its two endpoints, which takes $O(g(v)l(v) \log \Delta)$ time. ◀



■ **Figure 5** When the degree of a vertex v decrease, we may need to recolour at most $l(v)$ edges.

► **Lemma 13** (Adapting to the arboricity). *Increasing the level of a vertex v takes $O(\deg^+(v) \log \Delta)$ time. Decreasing the level of a vertex takes $O(d \cdot k' \log \Delta)$ time.*

Proof. The difference with the previous section is the following: when we decrement the level of a vertex v , we may need to recolour any edge uv such that $\min(l(u), l(v))$ decrease, i.e. u was on the same level as v or higher. There are at most $\deg^+(v) = O(d)$ such edges, which can be recoloured in $O(k' \log \Delta)$ time. ◀

■ **Algorithm 6** Decrementing the level of a vertex with adaptative arboricity.

```

procedure ADAPTATIVE DECREMENT( $v$ )
     $i = l(v)$ .
    Merge  $N_{Z_i}(v)$  and  $N_{H_{i-1}}(v)$  into  $N_{Z_{i-1}}(v)$ .
    Create the palette of  $v$  for  $Z_{i-1}$ .
    Discard the palette of  $v$  for  $Z_i$ .
    for  $u \in N_{Z_i}(v)$  do
        Move  $v$  from  $N_{H_i}(u)$  or  $N_{Z_i}(u)$  to  $N_{H_{i-1}}(u)$ .
        Update the pointers accordingly.
        if  $l(u) = i$  then
            Update the palette of  $u$  for the sets  $Z_i$ : remove colour  $c(uv)$ .
        RECOLOUR( $uv$ )
    Decrement  $l(v)$ 

```

► **Theorem 14.** *We can maintain a dynamic $\Delta(uv) + O(\alpha)$ edge colouring of a graph in amortized $O(\log n \log \alpha_{max} \log \Delta_{max})$ time for insertions, $O(\log^2 n \log \alpha_{max} \log \alpha \log \Delta_{max})$ for deletions.*

Proof. We define the following potential:

$$\begin{aligned}
B &= k'_{max} \log \Delta_{max} \sum_{v \in V} \phi(v) + \log \Delta_{max} \sum_{e \in E} \psi(e) \\
\phi(v) &= \sum_{j=1}^{l(v)-1} \max(0, \beta d(v) - \deg_{Z_j}(v)) \\
\psi(u, v) &= 2(k'_{max} - \min(l(u), l(v))) + \mathbf{1}_{l(u)=l(v)} \\
k'_{max} &= L \lceil \log(4\alpha_{max}) \rceil \in O(\log n \log \alpha_{max})
\end{aligned}$$

When we insert an edge, we create a potential $\psi(u, v) \leq 2k'_{max}$. The potential of the other edges do not change and the potential of a vertex can only decrease, therefore the potential B increase by at most $\log \Delta_{max} \cdot 2k'_{max}$. When we delete an edge, we need $O(\log n \log^2 \alpha \log \Delta)$ time to update the overflowing colours of the in-neighbours of v (lemma 12). Then for the potentials: $\phi(u)$ and $\phi(v)$ increase by at most k' each, $\psi(u, v)$ is deleted, and the other potentials are not affected, therefore the potential increase by at most $k'_{max} \log \Delta_{max} \cdot 2k'$, therefore the costs.

Incrementing the level of a dirty vertex. Let d denote the value of $d(v)$ before the change of level and d' the value after the modification.

- If v changes group, we have $d' = 2d$, otherwise $d' = d$. Either way, we have $\deg^+(v) > 10d \Rightarrow \max(0, \beta d' - \deg^+(v)) = 0$. It follows that $\phi(v)$ is unchanged. The potential of the other vertices can not increase (though it might decrease for a neighbour of v).
- The potential of an edge may only change if one of the endpoints is v and the other endpoint u verifies $l(u) \geq i$. Therefore, there are exactly $\deg^+(v)$ edges whose potential drop by one or two.

The total drop in potential is at least:

$$\deg^+(v) \log \Delta_{max} = \Theta(\deg^+(v) \log \Delta)$$

Decrementing the level of a dirty vertex.

- We must have $\deg_{Z_{i-1}}(v) < d \Rightarrow \max(\beta d - \deg_{Z_{i-1}}(v)) \geq 4d$. Therefore, the $\phi(v)$ drops by at least $4d$. If the level of v decreases, the potential only decreases further.
- If u is a neighbour of v , $\deg_{Z_j}(v)$ is decremented if $j = i$ and is unchanged otherwise. This affects the potential of u if $l(u) > i$. Therefore, $\sum_{u \in \Gamma(v)} \phi(u)$ increase by at most $6 \deg_{Z_{i+1}}(v) < \deg_{Z_{i-1}}(v) < d$.
- The potential of an edge may only change if one of the endpoints is v and the other endpoint u verifies $l(u) \leq i$. Therefore, there are exactly $\deg^+(v) < \deg_{Z_{i-1}}(v) < d$ edges whose potential increase by one or two.

The total drop of potential is at least:

$$k'_{max} \log \Delta_{max} (4d - d) - \log \Delta_{max} \cdot 2d \geq k'_{max} \log \Delta_{max} \cdot d \quad \blacktriangleleft$$

5 Conclusion

In this paper, we show how to maintain a $\Delta(uv) + O(\alpha)$ edge colouring in polylogarithmic time through the use of dynamic hierarchical partition. We also propose a simpler data structure to maintain a $\Delta_{max} + O(\alpha_{max})$ edge colouring, which can be done faster than the aforementioned algorithm.

We give an amortized analysis of the running time of our dynamic algorithms. This raises the question of what can be done in worst case time. In our case, we are only limited by the updates of our hierarchical partitions, so it motivates the search for hierarchical partitions with efficient worst-case update times.

The question that motivated our research is still open for graphs that have a large arboricity compared to their maximum degrees: is it possible to maintain a $\Delta + O(\Delta^{1-\epsilon})$ edge colouring, with ϵ a positive constant, in polylogarithmic time?

In the static setting, we showed that we can make a $\Delta(uv) + 2\alpha - 2$ edge colouring in $O(m \log \Delta)$ time, which is as fast as the greedy $2\Delta(uv) - 1$ algorithm. Thus, we get a $\Delta(uv) + O(1)$ edge colouring for graphs of constant arboricity, such as planar graphs, in near-linear time: more precisely, a planar graph has arboricity at most 3 [27], so by our result, it can in near-linear time be edge-coloured with $\Delta(uv) + 2\alpha - 2 = \Delta(uv) + 4$ colours.

Recently, it was shown by Bhattacharya, Costa, Panski and Solomon [8] that one can compute a $(\Delta + 1)$ edge colouring in $\tilde{O}(\min\{m\sqrt{n}, m\Delta\} \cdot \frac{\alpha}{\Delta})$ -time, which gives a near linear time algorithm for graphs of polylogarithmic arboricity. It emphasises the question whether a near-linear time $\Delta + O(1)$ edge-colouring algorithm could be obtained for a wider class of graphs.

References

- 1 Leonid Barenboim and Michael Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. In *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, pages 25–34, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1400751.1400757.
- 2 Leonid Barenboim, Michael Elkin, and Tzalik Maimon. Deterministic distributed $(\Delta + o(\Delta))$ -edge-coloring, and vertex-coloring of graphs with bounded diversity. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 175–184. ACM, 2017. doi:10.1145/3087801.3087812.
- 3 Anton Bernshteyn. A Fast Distributed Algorithm for $((\Delta + 1))$ -Edge-Coloring. *Journal of Combinatorial Theory, Series B*, 152:319–352, 2022. doi:10.1016/j.jctb.2021.10.004.
- 4 Anton Bernshteyn and Abhishek Dhawan. Fast algorithms for vizing’s theorem on bounded degree graphs. *CoRR*, abs/2303.05408, 2023. doi:10.48550/arXiv.2303.05408.
- 5 Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $O(\log^2 n)$ replacements. *J. ACM*, 66(5):37:1–37:23, 2019. doi:10.1145/3344999.
- 6 Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic algorithms for graph coloring. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20. SIAM, 2018. doi:10.1137/1.9781611975031.1.
- 7 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Arboricity-dependent algorithms for edge coloring. *CoRR*, abs/2311.08367, 2023. doi:10.48550/arXiv.2311.08367.
- 8 Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Density-sensitive algorithms for $(\Delta+1)$ -edge coloring. *CoRR*, abs/2307.02415, 2023. doi:10.48550/arXiv.2307.02415.
- 9 Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 173–182. ACM, 2015. doi:10.1145/2746539.2746592.

- 10 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In Frank Dehne, Jörg-Rüdiger Sack, Arvind Gupta, and Roberto Tamassia, editors, *Algorithms and Data Structures*, Lecture Notes in Computer Science, pages 773–782, Netherlands, 1999. Springer. 6th International Workshop on Algorithms and Data Structures. WADS 1999 ; Conference date: 11-08-1999 Through 14-08-1999. doi:10.1007/3-540-48447-7_34.
- 11 Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. The complexity of distributed edge coloring with small palettes. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2633–2652. SIAM, 2018. doi:10.1137/1.9781611975031.168.
- 12 Aleksander B. G. Christiansen, Eva Rotenberg, and Juliette Vlieghe. Sparsity-parameterised dynamic edge colouring. *CoRR*, abs/2311.10616, 2023. doi:10.48550/arXiv.2311.10616.
- 13 Aleksander Bjørn Grodt Christiansen. The power of multi-step vizing chains. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1013–1026. ACM, 2023. doi:10.1145/3564246.3585105.
- 14 Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21(1):5–12, January 2001. doi:10.1007/s004930170002.
- 15 Peter Davies. Improved distributed algorithms for the lovász local lemma and edge coloring. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 4273–4295. SIAM, 2023. doi:10.1137/1.9781611977554.ch163.
- 16 Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '19*, pages 1937–1945, USA, 2019. Society for Industrial and Applied Mathematics.
- 17 Thomas Erlebach and Klaus Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255(1):33–50, 2001. doi:10.1016/S0304-3975(99)00152-8.
- 18 Harold N. Gabow, Takao Nishizeki, Oded Kariv, Daniel Leven, and Osamu Terada. Algorithms for edge-colouring graphs. *Technical Report*, 1985.
- 19 Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. Deterministic Distributed Edge-Coloring with Fewer Colors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 418–430, 2018.
- 20 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Explicit and implicit dynamic coloring of graphs with bounded arboricity, 2020. doi:10.48550/arXiv.2002.10142.
- 21 Ian Holyer. The np-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981. doi:10.1137/0210055.
- 22 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2014. doi:10.1007/978-3-662-43951-7_45.
- 23 David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, July 1983. doi:10.1145/2402.322385.
- 24 C. St.J. A. Nash-Williams. Decomposition of Finite Graphs Into Forests. *Journal of the London Mathematical Society*, s1-39(1):12–12, January 1964. doi:10.1112/jlms/s1-39.1.12.
- 25 Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic MIS in uniformly sparse graphs. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPICs*, pages 92:1–92:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.ICALP.2018.92.

- 26 David Peleg and Shay Solomon. Dynamic $(1 + \epsilon)$ -approximate matchings: A density-sensitive approach. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 712–729. SIAM, 2016. doi:10.1137/1.9781611974331.ch51.
- 27 K. S. Poh. On the linear vertex-arboricity of a planar graph. *J. Graph Theory*, 14(1):73–75, 1990. doi:10.1002/jgt.3190140108.
- 28 Corwin Sinnamon. A randomized algorithm for edge-colouring graphs in $O(m\sqrt{n})$ time. *CoRR*, abs/1907.03201, 2019. arXiv:1907.03201.
- 29 Hsin-Hao Su and Hoa T Vu. Towards the Locality of Vizing’s Theorem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 355–364, 2019.
- 30 V. G. Vizing. The chromatic class of a multigraph. *Cybernetics*, 1(3):32–41, May 1965. doi:10.1007/BF01885700.