PROGRESS ON THE ADJACENT VERTEX DISTINGUISHING EDGE COLOURING CONJECTURE

GWENAËL JORET AND WILLIAM LOCHET

Abstract. A proper edge colouring of a graph is *adjacent vertex distinguishing* if no two adjacent vertices see the same set of colours. Using a clever application of the Local Lemma, Hatami (2005) proved that every graph with maximum degree Δ and no isolated edge has an adjacent vertex distinguishing edge colouring with $\Delta + 300$ colours, provided Δ is large enough. We show that this bound can be reduced to $\Delta + 19$. This is motivated by the conjecture of Zhang, Liu, and Wang (2002) that $\Delta + 2$ colours are enough for $\Delta \geqslant 3$.

1. Introduction

We use the notation $[n] := \{1, 2, \ldots, n\}$. By 'graph' we mean a finite, undirected, and loopless graph. Graphs are assumed to have no parallel edges, unless otherwise stated. We generally follow the terminology of Diestel [4] for graphs, and refer the reader to this textbook for undefined terms and notations. Given a graph G, we let $N_G(u)$ denote the neighbourhood of vertex u in G, and let $d_G(u)$ denote the degree of u. We omit the subscript G when the graph is clear from the context.

In this paper, a *colouring* of a graph G always means an edge colouring of G, defined as a mapping $c : E(G) \to \mathbb{N}$ associating integers (colours) to the edges of G. A colouring is *proper* if no two adjacent vertices receive the same colour. Given a proper colouring c of G and a vertex $u \in V(G)$, we let $S_c(u)$ denote the set of colours appearing on edges incident to u. A proper colouring c is *adjacent vertex distinguishing* (*AVD* for short) if $S_c(u) \neq S_c(v)$ for every edge $uv \in E(G)$.

Using a clever application of the Local Lemma, Hatami [10] proved that every graph with maximum degree $\Delta > 10^{20}$ and no isolated edge has an AVD-colouring with at most $\Delta + 300$ colours. Our main result is that this bound can be reduced to $\Delta + 19$ (for large enough Δ):

Theorem 1. *There exists an integer $\Delta_0 \geqslant 0$ such that every graph G with maximum degree $\Delta \geqslant \Delta_0$ and no isolated edge has an AVD-colouring with at most $\Delta + 19$ colours.*

(G. Joret) Computer Science Department, Université Libre de Bruxelles, Brussels, Belgium

(W. Lochet) Université Côte d'Azur, CNRS, I3S, INRIA, Sophia Antipolis and ENS Lyon, LIP, Lyon, France

E-mail addresses: gjoret@ulb.ac.be, william.lochet@gmail.com.

Date: April 17, 2018.

G. Joret is supported by an ARC grant from the Wallonia-Brussels Federation of Belgium.

This is motivated by the following conjecture of Zhang, Liu, and Wang [20].

Conjecture 2 ([20])**.** *Every graph with maximum degree $\Delta \geqslant 3$ and no isolated edge has an AVD-colouring with $\Delta + 2$ colours.*

($\Delta \geqslant 3$ is required because C_5 needs five colours, it is the only connected graph known to require more than $\Delta + 2$ colours.) This conjecture captured the attention of several researchers over the years. It is known to be true e.g. if G is bipartite [1]; if $\Delta = 3$ [1]; if G is planar and $\Delta \geqslant 12$ [2, 11]; if G is 2-degenerate [19]. The conjecture is also known to hold asymptotically almost surely for random 4-regular graphs [9].

Various variants and strengthenings of the conjecture have been proposed in the literature. A natural one is a list version of the problem, where each edge of G has a list of colours it can use. Recall that for proper edge colourings, the famous list colouring conjecture states that the minimum list size such that G has a proper edge colouring where each edge receives a colour from its list is equal to its chromatic index; in particular, it is at most $\Delta + 1$. Kahn [13] was the first to establish an upper bound of the form $\Delta + o(\Delta)$ on the required list size, and the best asymptotic bound known to this day is $\Delta + O(\sqrt{\Delta}(\log \Delta)^4)$ due to Molloy and Reed [15]. Kwaśny and Przybyło [14] recently strenghened the latter result by showing that the resulting list edge colouring can moreover be guaranteed to be an AVD-colouring. Horňák and Woźniak [12] conjectured that the list edge colouring conjecture also holds in the AVD setting, and thus in particular that a $\Delta + O(1)$ bound on the list size should be enough.

Another strengthening of the AVD-colouring conjecture, due to Flandrin *et al.* [8], states that every graph with no isolated edge and maximum degree $\Delta \geqslant 3$ has a proper edge colouring c with colours in $[\Delta+2]$ such that, for every edge uv, the sum of colours of edges incident to u and that of v are distinct. This is called a *neighbour sum distinguishing* colouring. The best asympotic bound is due to Przybyło [17, 18], who obtained an upper bound of $\Delta + O(\Delta^{1/2})$ on the number of required colours in such a colouring. It is an intriguing open problem whether a $\Delta + O(1)$ bound holds; neither Hatami's proof [10] for AVD-colourings nor ours seem to be adaptable to this setting. We refer the reader to [3, 18] and the references therein for further pointers to the relevant literature.

The paper is organised as follows. In Section 2 we set up the plan for the proof of Theorem 1. In particular, vertices with big degrees (at least $\Delta/2$, roughly) and those with small degrees are treated independently, and differently. Then, Section 3 and Section 4 are devoted to handling big and small degree vertices, respectively.

2. Initial colouring

Fix some ϵ with $0 < \epsilon < 1/2$, which will be assumed small enough later in the proof. Let G be a graph with maximum degree Δ and no isolated edge. We assume Δ is large enough for various inequalities appearing in the proof to hold.

The beginning of our proof of Theorem 1 follows closely that of Hatami [10]. In particular, we reuse his approach of treating differently vertices with 'small' degrees and those with 'big' degrees, except we use $(1/2-\epsilon)\Delta$ as the threshold instead of $\Delta/3$ in [10]. This larger threshold helps a little bit in reducing the additive constant in the main theorem; however, the bulk of the reduction from 300 to 19 comes from treating big degree vertices differently.

Let $d := \lceil (1/2-\epsilon)\Delta \rceil$. Taking Δ large enough, we may assume that $d < \Delta/2$. We begin as in [10] by modifying the graph G as follows. Let G' be the *multigraph* obtained from G by contracting each edge $uv \in E(G)$ such that $d_G(u) < d$ and $d_G(v) < d$ but neither u nor v has any other neighbour w with $d_G(w) < d$. Then G' has maximum degree Δ and maximum edge multiplicity at most 2. Every proper colouring c' of G' can be extended to a proper colouring c of G with the same set of colours as follows: For each edge $e \in E(G)$ appearing in G', set $c(e) := c'(e)$. For each edge $uv \in E(G)$ that was contracted, we know that $d_G(u) + d_G(v) < \Delta$. Thus some colour α of c' is not used on any of the edges incident to u and v, set then $c(uv) := \alpha$.

In [10], the author points out that if moreover c' is an AVD-colouring of G' then c is an AVD-colouring of G. Using this observation, the proof in [10] then focuses on finding an AVD-colouring of G'. This is done by starting with a proper colouring c' with $\Delta+2$ colours, which exists by Vizing's theorem, and then recolouring some edges of G' with new colours to obtain an AVD-colouring of G'. The advantage of working in G' instead of G is that the subgraph of G' induced by the vertices with degree strictly less than d has no isolated edge, which is important in that proof.

In our proof, we follow a similar approach but we keep the focus on G: We start with a proper colouring c' of G' with $\Delta+2$ colours obtained from Vizing's theorem and extend it to a colouring c of G as in the above remark. Thus, the colouring c uses $\Delta+2$ colours and satisfies the following property:

$$S_c(u) \cap S_c(v) = \{c(uv)\} \quad \forall uv \in E(G) \text{ s.t.} \{w \in N(u) \cup N(v) : d_G(w) < d\} = \{u, v\}. \quad (1)$$

Then, we modify c to obtain an AVD-colouring of G. Thus G' is only used to produce the initial colouring c of G. One advantage of working in G is that we avoid having to deal with parallel edges, which would introduce (trivial but annoying) technicalities in our approach. On the other hand, a small price to pay compared to [10] is that we will have to watch out for these edges uv such that $\{w \in N(u) \cup N(v) : d_G(w) < d\} = \{u, v\}$ in our proof.

Say that a vertex $u \in V(G)$ is *small* if $d_G(u) < d$, and *big* otherwise. Let A and B be the sets of small and big vertices of G, respectively. Our goal is to transform the colouring c into an AVD-colouring of G. The plan for doing so is roughly as follows. First we show that we can uncolour a bounded number of edges per big vertex in such a way that edges uv with $S_c(u) = S_c(v)$ and $u, v \in B$ that remain form a matching satisfying some specific properties. Then we show how we can recolour these uncoloured edges, plus a few other edges of G, to obtain a colouring where every edge uv with $u, v \in B$ satisfies $S_c(u) \neq S_c(v)$. Finally, we recolour edges with both endpoints in A in such a way that the resulting colouring is an AVD-colouring of G.

3. Big vertices

For each vertex $u \in B$, choose an arbitrary subset $N^+(u)$ of $N(u)$ of size d. We use a randomised algorithm, Algorithm 1, to select a subset $U^+(u) \subseteq N^+(u)$ of size 2 for each vertex $u \in B$. For each vertex $v \in V(G)$, we let $U^-(v) := \{u \in B : v \in U^+(u)\}$. Algorithm 1 chooses the subsets $U^+(u)$ iteratively, one big vertex u at a time. Hence, we see the sets $U^+(u)$ as variables, and the sets $U^-(v)$ ($v \in V(G)$) as being determined by these variables. (For definiteness, we set $U^+(u) := \emptyset$ for every small vertex u.) Just after choosing the subset $U^+(u)$ of a big vertex u, the algorithm checks whether this choice triggered any 'bad event'. If so, the bad event is handled, which involves *resetting* the variable $U^+(u)$, which means setting $U^+(u) := \emptyset$, and possibly resetting other variables $U^+(v)$ for some well-chosen big vertices v close to u in G.

Thanks to these bad events, the selected subsets satisfy a number of properties. A key property is that $|U^-(v)| \leqslant q$ for every $v \in V(G)$, with $q := 13$ being the constant that is optimised in this proof.

At any time during the execution of the algorithm, we say that an edge $uv \in E(G)$ is *selected* if $v \in U^+(u)$ or $u \in U^+(v)$. In the algorithm, we will make sure that if $v \in U^+(u)$ then $u \notin U^+(v)$ (that is, an edge can be selected 'at most once').

After the algorithm terminates, selected edges will be used to fix locally the colouring c for big vertices: The plan is to recolour them using $q+3$ new colours, and then recolour a well-chosen matching of G with yet another new colour, in such a way that at the end $S_c(u) \neq S_c(v)$ holds for all edges $uv \in E(G)$ with $u, v \in B$. The resulting colouring of G will use $\Delta + q + 6$ colours in total.

Let us explain the conventions and terminology used in Algorithm 1. First, we assume that the vertices of G are ordered according to some fixed arbitrary ordering. This naturally induces an ordering of each subset of $V(G)$ as well, of each set of pairs of vertices (say using lexicographic ordering), and more generally of any set of structures built using vertices of G. This is used implicitly in what follows.

We say that an edge uv linking two big vertices is *finished* if $U^+(w) \neq \emptyset$ for each big vertex w in $N(u) \cup N(v)$ (note that this set includes u and v). For $u \in V(G)$, define $S'(u)$ as the set $S_c(u)$ minus colours of edges incident to u that are selected. That is,

$$S'(u) := \{c(uv) : v \in N(u), v \notin U^+(u) \cup U^-(u)\}.$$

In the algorithm, we check whether $S'(u) = S'(v)$ for two big vertices u, v with $d_G(u) = d_G(v)$ only when uv is finished, the idea being that if there are still big vertices adjacent to u or v waiting to be treated then this could potentially impact the sets $S'(u)$ and $S'(v)$. We say that an edge $uv \in E(G)$ is *bad* if $u, v \in B$, uv is finished, $d_G(u) = d_G(v)$, and $S'(u) = S'(v)$.

As mentioned earlier, we need to watch out for edges uv such that $d_G(u) = d_G(v) < d$ and $d_G(w) \geqslant d$ for all $w \in (N(u) \cup N(v)) - \{u, v\}$. This is because every edge incident to u or v distinct from uv is incident to a big vertex, and all these edges will have a

fixed colour when we are done dealing with big vertices. Indeed, in the next section we only recolour edges in the subgraph induced by small vertices. Thus, if u and v were to see the same set of colours at the end of this step, we would have no way to fix this later. Note that at the beginning u and v see disjoint sets of colours in colouring c except for colour $c(uv)$. Once the algorithm terminates, we will recolour at most $q+2$ edges incident to each big vertex w (with new colours). Thus, if $d_G(u) = d_G(v) \geqslant q+4$, we know that there will be at least one edge e incident to u distinct from uv that kept its original colour $c(e)$. Since v does not see the colour $c(e)$, we are then assured that u and v see different sets of colours after the recolouring step. Therefore, it is only when $d_G(u) = d_G(v) \leqslant q+3$ that we need to be careful when selecting edges incident to u or v to recolour. Let us call such edges *fragile* edges, i.e. uv is fragile if $d_G(u) = d_G(v) \leqslant q+3$ and $d_G(w) \geqslant d$ for all $w \in (N(u) \cup N(v)) - \{u, v\}$. Fragile edges will be carefully handled in the algorithm.

As mentioned, the algorithm considers remaining big vertices u with $U^+(u) = \emptyset$ one by one, selects the subset $U^+(u)$ randomly each time, and deals with any bad event that may occur. Let us explain how the random choices are made. Given a big vertex u with $U^+(u) = \emptyset$, an unordered pair $\{v, w\} \subseteq N^+(u)$ is *admissible* for u if the following three conditions are satisfied:

- $v, w \notin U^-(u)$;
- if $vw \in E(G)$ then vw is not fragile, and
- setting $U^+(u) := \{v, w\}$ does not create any bad edge incident to u.

At the beginning of the `while` loop, the algorithm chooses an admissible pair for the vertex $u \in B$ under consideration uniformly at random among the first $s := \binom{d-q}{2} - 3d$ admissible pairs. Lemma 3 below shows that there are always at least s such pairs, thus this random choice can always be made.

Five types of bad events are considered in the algorithm. They correspond to the five conditions tested by the `if` / `else if` statements; we refer to them as Bad Event 1, Bad Event 2, etc. in order. These events state the existence of certain structures in the graph. We remark that there could be more than one instance of the structure under consideration in the graph. (For instance, there could be two vertices $v \in N(u)$ with $|U^-(v)| = q+1$ in Bad Event 1.) In this case, we assume that the algorithm chooses one according to some deterministic rule. For the convenience of the reader, the five types of bad events considered are illustrated in Figure 1. Let us emphasise that if any bad event is triggered, then the current vertex u is always reset (i.e. the algorithm sets $U^+(u) := \emptyset$). This will ensures that no other bad event remains in the graph after dealing with the bad event under consideration.

The following lemma establishes some key properties of Algorithm 1. Note that by an *invariant* of the `while` loop, we mean a property that is true every time the condition of the loop is being tested. Thus, such a property holds when a new iteration of the loop starts, and also when the loop (and thus the algorithm) stops.

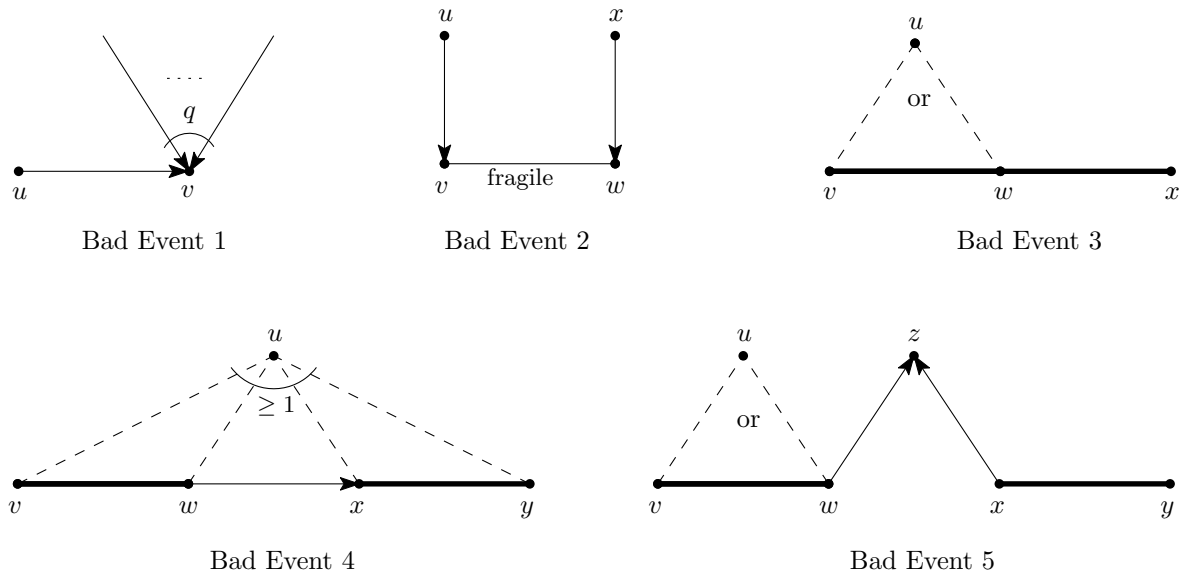

FIGURE 1. The five types of bad events in Algorithm 1. Bad edges are drawn in bold. (Note that possibly $z = u$ in Bad Event 5.)

Algorithm 1: Uncolouring some edges incident to big vertices.

$U^+(u) \leftarrow \emptyset \quad \forall u \in B$
while $\exists v \in B$ *with* $U^+(v) = \emptyset$ **do**
  $u \leftarrow$ first such vertex
  $U^+(u) \leftarrow$ admissible pair chosen uniformly at random among first s ones
  if $\exists v \in N(u)$ *with* $|U^-(v)| = q + 1$ **then**
    $U^+(w) \leftarrow \emptyset \quad \forall w \in U^-(v)$
  else if $\exists$ *fragile edge* vw *and* $x \in V(G) - \{u, v, w\}$ *s.t.* $u \in U^-(v)$ *and* $x \in U^-(w)$ **then**
    $U^+(a) \leftarrow \emptyset \quad \forall a \in \{u, x\}$
  else if $\exists$ *distinct bad edges* vw, wx *with* $u \in N(v) \cup N(w)$ **then**
    $U^+(a) \leftarrow \emptyset \quad \forall a \in \{u, v, x\}$
  else if $\exists$ *two independent bad edges* vw, xy *with* $u \in N(v) \cup N(w) \cup N(x) \cup N(y)$ *and* $x \in U^+(w)$ **then**
    $U^+(a) \leftarrow \emptyset \quad \forall a \in \{u, v, w, y\}$
  else if $\exists$ *two independent bad edges* vw, xy *with* $u \in N(v) \cup N(w)$, *and* $\exists z \in V(G) - \{v, w, x, y\}$ *with* $w, x \in U^-(z)$ **then**
    $U^+(a) \leftarrow \emptyset \quad \forall a \in \{u, v, w, x, y\}$

Lemma 3. *The following properties are invariants of the* `while` *loop in Algorithm 1:*

(1) $|U^-(v)| \leqslant q$ *for every* $v \in V(G)$.
(2) At least one of $U^-(v), U^-(w)$ *is empty for each fragile edge* vw.
(3) Bad edges form a matching.
(4) If $w, x \in B$ *belong to distinct bad edges, then* $(\{w\} \cup U^+(w)) \cap (\{x\} \cup U^+(x)) = \emptyset$.

(5) Every $u \in B$ with $U^+(u) = \emptyset$ has at least s admissible pairs.

Proof. Let us start with property (1). Clearly, $|U^-(v)| \leqslant q$ for every $v \in V(G)$ the first time the condition of the `while` loop is being tested. This remains true for every subsequent test of the condition, thanks to Bad Event 1: Selecting the subset $U^+(u)$ for a vertex $u \in B$ could create up to two vertices v with $|U^-(v)| = q + 1$ but these get fixed immediately when u is reset. Hence, (1) is an invariant of the loop.

Similarly, it is clear that property (2) is an invariant of the `while` loop, thanks to Bad Event 2.

Let us consider property (3). The property is true at the beginning of the algorithm, since there are no bad edges. Next, suppose that property (3) held true at the beginning of the loop but that there are two incident bad edges e, f just after selecting the admissible pair $U^+(u)$ for a big vertex u. Then at least one of e, f, say e, became bad just after treating u. Note that e cannot be incident to u, by definition of admissible pairs. Thus e is at distance 1 from u. It suffices to show that some bad event is triggered, since then u is reset and e is no longer bad (since e is not finished). This is clearly true, since either Bad Event 1 or Bad Event 2 is triggered, and if not then Bad Event 3 is triggered for sure (because of the existence of the pair e, f). Thus we see that property (3) is an invariant of the loop.

The proof for property (4) is similar. The property clearly holds at the beginning of the algorithm. Next, suppose that it held true at the beginning of the loop but that just after selecting the subset $U^+(u)$ for a big vertex u, there are two independent bad edges vw, xy s.t. $\{w\} \cup U^+(w)$ and $\{x\} \cup U^+(x)$ intersect. Then at least one of the two edges, say vw, is at distance exactly 1 from u. (Recall that u cannot be incident to either of the two edges, by definition of admissible pairs.) As before, it suffices to show that some bad event occurs, since then u is reset and vw is no longer bad. Let $z \in (\{w\} \cup U^+(w)) \cap (\{x\} \cup U^+(x))$ and say none of the first four bad events happens. Then $z \notin \{v, w, x, y\}$, since otherwise Bad Event 4 would have been triggered. But this shows that Bad Event 5 occurs. We deduce that property (4) is maintained.

Finally, it remains to show that property (5) is an invariant of the loop. Consider thus any vertex $u \in B$ with $U^+(u) = \emptyset$ when the condition of the loop is being tested. (Thus, a new iteration of the loop is about to start.) From invariant (1), we know that there are at least $\binom{d-q}{2}$ unordered pairs of distinct vertices in $N^+(u) - U^-(u)$. Next, a key observation is that for every $x \in N(u)$, if there exists $\{v, w\} \subseteq N^+(u) - U^-(u)$ s.t. setting $U^+(u) := \{v, w\}$ makes the edge ux bad, then the set $\{v, w\}$ is uniquely determined. Hence, potential bad edges forbid at most $|N(u)| \leqslant \Delta$ pairs of vertices in $N^+(u) - U^-(u)$. Finally, among the remaining pairs $\{v, w\}$, at most $\lfloor d/2 \rfloor$ of them are s.t. vw is a fragile edge. Therefore, we conclude that there are at least $\binom{d-q}{2} - \Delta - \lfloor d/2 \rfloor \geqslant \binom{d-q}{2} - 3d = s$ admissible pairs for u. □

The properties listed in Lemma 3 hold in particular when Algorithm 1 stops. However, it is not clear at first sight that the algorithm should ever stop. Our next result shows

that it does so with high probability. For simplicity, we sometimes call one iteration of the `while` loop a *step*.

Theorem 4. *The probability that Algorithm 1 stops in at most t steps tends to* 1 *as* $t \to \infty$.

We use an 'entropy compression' argument to prove this theorem, a proof method introduced by Moser and Tardós [16] in their celebrated algorithmic proof of the Lovász Local Lemma. In a nutshell, the main idea of the proof is to look at sequences of t random choices such that the algorithm does not stop in at most t steps. Exploiting the fact that the algorithm did not stop, we show how one can get an implicit lossless encoding of these sequences, by writing down a concise log of the execution of the algorithm. Then, looking at the structure of the algorithm, we prove that there are only $o(s^t)$ such logs. Since in total there are s^t random sequences of length t, we deduce that only a $o(1)$-fraction of these make the algorithm run for at least t steps. Theorem 4 follows.

To describe the log of an execution of the algorithm, we need the following definitions. First, recall that a *Dyck word* of semilength k is a binary word $w_1 w_2 \dots w_{2k}$ with exactly k 0s and k 1s such that the number of 0s is at least the number of 1s in every prefix of the word. A *descent* in a Dyck word is a maximal sequence of consecutive 1s, its *length* is the number of 1s.

For our purposes, it will be more convenient to drop the requirement that a Dyck word has the same number of 0s and 1s. Let us define a *partial Dyck word* of semilength k as a binary word $w_1 w_2 \dots w_p$ with exactly k 0s and *at most* k 1s such that the number of 0s is at least the number of 1s in every prefix of the word. Descents are defined in the same way as for normal Dyck words.

Let us consider a sequence $(r_1, \dots, r_t)$ of t random choices such that Algorithm 1 does not stop in at most t steps when run with these random choices. In other words, the algorithm is about to start its $(t+1)$-th iteration of the `while` loop, at which point we freeze its execution. Each random choice r_i consisted in choosing an admissible pair for some big vertex u among its first s admissible pairs, thus we see r_i as a number in $[s]$.

For each $i \in [t+1]$, let U_i^+ and U_i^- denote the functions U^+ and U^-, respectively, at the beginning of the i-th iteration, and let B_i denote the subset of vertices $u \in B$ with $U_i^+(u) = \emptyset$. We associate to the sequence $(r_1, \dots, r_t)$ a corresponding *log* $(W, \gamma, \delta, U_{t+1}^+)$, where W is a partial Dyck word of semilength t such that the length of each descent is in the set $\{2, 3, 4, 5, q+1\}$, and $\gamma = (\gamma_1, \dots, \gamma_t)$ and $\delta = (\delta_1, \dots, \delta_t)$ are two sequences of integers.

The partial Dyck word W is built as follows during the execution of the algorithm: Starting with the empty word, we add a 0 at the end of the word each time a big vertex is treated. If the corresponding random choice triggers a bad event, we moreover add ℓ 1s at the end of the word, where ℓ is the number of big vertices that are reset (so $\ell = q+1, 2, 3, 4, 5$ for bad events of types $1, 2, 3, 4, 5$, respectively). Thus descents in

W are in bijection with bad events treated during the execution, and the length of a descent tells us the type of the corresponding bad event.

The two sequences γ and δ are defined as follows. For $i \in [t]$, the integers γ_i and δ_i encode information about the bad event handled during iteration i. If there was none, we simply set $\gamma_i := \delta_i := -1$. Otherwise, γ_i is a nonnegative integer encoding the set of big vertices that are reset when the bad event is handled, and δ_i is a nonnegative integer encoding extra information which will help us recover the random choice r_i from the log. The precise definitions of γ_i and δ_i depend on the type of the bad event (see the list below); however, before giving these definitions we must explain the assumptions we make.

The definition of γ_i assumes that the set B_i is known. In turn, γ_i will encode enough information to determine completely B_{i+1} from B_i. Since $B_1 = B$, it then follows that we can read off all the sets $B_1, B_2, \ldots, B_{t+1}$ from the sequence γ: For $i = 1, \ldots, t$, either $\gamma_i \geqslant 0$, in which case B_{i+1} is determined by B_i and γ_i. Or $\gamma_i = -1$, in which case no bad event occurred during iteration i, and thus $B_{i+1} := B_i - \{u\}$ where u is the first vertex in B_i.

As already mentioned, the purpose of the log is to encode all t random choices $r_1, \ldots, r_t$ that have been made during the execution. To encode r_i ($i \in [t]$), we work *backwards*: We assume that the function U^+_{i+1} is known, and we show that one can then deduce r_i and U^+_i using the log. Since U^+_{t+1} is part of the log, this implies that the log uniquely determines $r_t, r_{t-1}, \ldots, r_1$, as desired. Let us remark that if no bad event occurred during the i-th iteration, then we can already deduce r_i and U^+_i from U^+_{i+1} using the sets B_i and B_{i+1}. Indeed, in this case $B_i = B_{i+1} \cup \{u\}$ where u is the vertex treated during the i-th iteration. Thus, for $v \in B$,

$$U^+_i(v) = \begin{cases} U^+_{i+1}(v) & \text{if } v \neq u \\ \emptyset & \text{if } v = u \end{cases}$$

Furthermore, $U^+_{i+1}(u)$ tells us what was the random choice r_i that was made for u during iteration i. Indeed, using U^+_i we can deduce what was the set of admissible pairs for u at the beginning of iteration i. Then, r_i is the position of the pair $U^+_{i+1}(u)$ in the ordering of these admissible pairs. Therefore, it is only when a bad event happens that we need extra information to determine r_i and U^+_i. This is precisely the role of δ_i.

Definitions of γ and δ. Let $i \in [t]$. If no bad event occurred during iteration i, set $\gamma_i := -1$ and $\delta_i := -1$. Otherwise, say that a bad event β of type j was handled. The definition of γ_i assumes that B_i is known, while that of δ_i assumes that B_i and U^+_{i+1} are both known. In particular, we know the vertex u treated at the beginning of the iteration, since it is the first vertex in B_i. With these remarks in mind, γ_i and δ_i are defined as follows:

$j = 1$ The bad event β was triggered because the admissible pair chosen for u contained a vertex v with $|U^-_i(v)| = q$. Vertex u and the q vertices in $U^-_i(v)$ were subsequently reset. There are at most d choices for vertex v and at most $\binom{\Delta}{q}$

choices for $U_i^-(v)$. We may thus encode v and $U_i^-(v)$ with a number $\gamma_i \in \left[d\binom{\Delta}{q}\right]$. Observe that $B_{i+1} = B_i \cup U_i^-(v)$.

Now that v and $U_i^-(v)$ are identified, we want to encode the admissible pair $\{v, x\}$ that was chosen for u at the beginning of the iteration, and the sets $U_i^+(w)$ for each vertex $w \in U_i^-(v)$. There are at most d choices for x, and similarly for each $w \in U_i^-(v)$ there are at most d choices for the vertex in $U_i^+(w)$ which is distinct from v. We let $\delta_i \in [d^{q+1}]$ encode these choices. Since U_i^+ only differs from U_{i+1}^+ on vertices $w \in U_i^-(v)$, with the encoded information we can deduce U_i^+ from U_{i+1}^+. Note also that r_i is determined by the admissible pair $\{v, x\}$ that was chosen for u.

$j = 2$ The bad event β was triggered because the admissible pair chosen for u contained a vertex v incident to a fragile edge vw with $U_i^-(w) \neq \emptyset$. Then two vertices were reset, namely u and some vertex x in $U_i^-(w)$. There are at most d choices for vertex v. Once v is identified, we know vertex w since fragile edges form a matching. Finally, there are at most $q+2$ choices for x, since $d_G(w) \leqslant q+3$ and $x \neq v$. We let $\gamma_i \in [(q+2)d]$ encode v, w, and x. Observe that $B_{i+1} = B_i \cup \{x\}$.

Next, to encode r_i we only need to specify the vertex in the admissible pair chosen for u that is distinct from v (d choices). Similarly, there are at most d possibilities for the set $U_i^+(x)$ since we know that it includes w. Hence, we can encode this information with a number $\delta_i \in [d^2]$. Note that, knowing x and $U_i^+(x)$, we can directly infer U_i^+ from U_{i+1}^+, since $U_i^+(y) = U_{i+1}^+(y)$ for all $y \in B - \{x\}$.

$j = 3$ After selecting the admissible pair for u, we had $S'(v) = S'(w) = S'(x)$ for three distinct vertices $v, w, x \in B - \{u\}$ with $vw, wx \in E(G)$ and $u \in N(v) \cup N(w)$. Then u, v, x were reset. There are at most $2\Delta^3$ choices for the triple v, w, x (the factor 2 is due to the fact that u can be adjacent to v or w). We let $\gamma_i \in [2\Delta^3]$ encode v, w, x. Observe that $B_{i+1} = B_i \cup \{v, x\}$.

Knowing v, w, x and U_{i+1}^+, our next aim is to encode U_i^+ and r_i using δ_i. First, we simply encode the admissible pair $\{u_1, u_2\}$ that was chosen for u during the i-th iteration explicitly, thus there are $\binom{d}{2}$ possibilities.[1] Next, we observe that $U_i^+(y) = U_{i+1}^+(y)$ for every $y \in B - \{u, v, x\}$, and $U_i^+(u) = \emptyset$. Thus it only remains to encode $U_i^+(v)$ and $U_i^+(x)$. Here the idea is that, since at this point we know the set $U_i^+(w)$ and the admissible pair $\{u_1, u_2\}$ chosen for u, there are only $O(1)$ possibilities for the sets $U_i^+(v)$ and $U_i^+(x)$ in order to have that $S'(v) = S'(w) = S'(x)$ just before u, v, x were reset.

[1] A reader familiar with these types of encoding arguments might wonder why we bother resetting u in the algorithm if we end up writing down $\{u_1, u_2\}$ explicitly in the encoding. It is indeed true that in this case we only 'win' something thanks to the implicit encoding of the choices made for v and x. Nevertheless, we still need to reset u as well, to keep the property that the current vertex is always reset whenever a bad event happens.

Let us focus on the set $U_i^+(v)$, the argument for $U_i^+(x)$ will be symmetric. First, let us write down the following local information: (1) Is $w \in U_i^+(v)$? (2) Is $w \in U_i^+(x)$? (3) Is $v \in U_i^+(x)$? Thus there are 8 possibilities. (1)–(2) gives enough information to reconstruct the set $S'(w)$ just before the resets, since we already know $U_i^+(w)$ and whether $w \in \{u_1, u_2\}$ or not. From (3) we also know the set $S''(v) := S'(v) - \{c(vv') : v' \in U_i^+(v)\}$ just before the resets, since we know whether $v \in \{u_1, u_2\}$ or not, and whether $v \in U_i^+(z)$ or not for every $z \in N(v) - \{u\}$. Now, it only remains to observe that $U_i^+(v)$ is determined by the two sets $S''(v)$ and $S'(w)$, namely $U_i^+(v) = \{v' \in N(v) : c(vv') \in S''(v) - S'(w)\}$.

Proceeding similary for the set $U_i^+(x)$ (8 possibilities again), this fully determines U_i^+. Now, given U_i^+ we know exactly the set of admissible pairs for u at the beginning of the i-th iteration. Since we know that the pair $\{u_1, u_2\}$ was chosen, we can deduce the value of r_i. Hence, this shows that U_i^+ and r_i can be encoded using a number $\delta_i \in [64\binom{d}{2}]$. (The constant 64 could be reduced with a more careful analysis but this would not make a difference later on.)

$j = 4$ Here we let $\gamma_i \in [4\Delta^4]$ encode the four vertices v, w, x, y as seen from u (the factor 4 comes from the fact that u is adjacent to at least one of them but we do not know which one). Since u, v, w, y are reset during this iteration, we have $B_{i+1} = B_i \cup \{v, w, y\}$.

Next, we set up δ_i to encode U_i^+ and r_i knowing U_{i+1}^+. As in the previous case, we encode the admissible pair $\{u_1, u_2\}$ that was chosen for u during the i-th iteration explicitly ($\binom{d}{2}$ choices). Once we know U_i^+, we know which are the admissible pairs for u at the beginning of the i-th iteration, and thus we can determine r_i, exactly as before. Thus, it only remains to encode $U_i^+(v), U_i^+(w)$, and $U_i^+(y)$.

Let us start with $U_i^+(w)$. We already know that $x \in U_i^+(w)$, and we encode the other vertex in $U_i^+(w)$ explicitly (d choices).

Next, consider $U_i^+(v)$. Here, the idea is the same as for Bad Event 3, namely once $U_i^+(w)$ is known there are only $O(1)$ possibilities for $U_i^+(v)$ to have that $S'(v) = S'(w)$ just before the resets. To be precise, we write down the following local information: (1) Is $w \in U_i^+(v)$? (2) Is $w \in U_i^+(x)$? (3) Is $w \in U_i^+(y)$? (4) Is $v \in U_i^+(x)$? (5) Is $v \in U_i^+(y)$? Thus there are 32 possibilities. (1)–(3) gives us enough information to reconstruct the set $S'(w)$ just before the resets, since we already know $U_i^+(w)$ and whether $w \in \{u_1, u_2\}$ or not. Similarly, (4)–(5) allow us to determine the set $S''(v) := S'(v) - \{c(vv') : v' \in U_i^+(v)\}$ just before the resets, which in turn determines $U_i^+(v)$ since $U_i^+(v) = \{v' \in N(v) : c(vv') \in S''(v) - S'(w)\}$.

For $U_i^+(y)$, we proceed exactly as for $U_i^+(v)$, exchanging v with y and w with x. The only difference here is that x is not reset, thus $U_i^+(x) = U_{i+1}^+(x)$. We similarly conclude that there are at most 32 possibilities for the set $U_i^+(y)$. In summary, we may encode all the necessary information with a number $\delta_i \in \left[2^{10} d \binom{d}{2}\right]$.

$j = 5$: We let $\gamma_i \in [2\Delta^5]$ encode the vertices v, w, x, y, z. (Recall that possibly $z = u$.) Since u, v, w, x, y are reset during this iteration, we have $B_{i+1} = B_i \cup \{v, w, x, y\}$.

Next, we encode U_i^+ and r_i based on U_{i+1}^+. Again, we encode the admissible pair $\{u_1, u_2\}$ chosen for u explicitly ($\binom{d}{2}$ choices), which will determine r_i once we know U_i^+. It only remains to encode $U_i^+(v), U_i^+(w), U_i^+(x), U_i^+(y)$.

Similarly as for Bad Event 4, there are most d possibilities for the set $U_i^+(w)$, since we already know that $z \in U_i^+(w)$. The same is true $U_i^+(x)$.

For $U_i^+(v)$ we proceed exactly as in the previous case, exploiting the fact that $U_i^+(w)$ is already encoded: Writing down which sets among $U_i^+(v), U_i^+(x), U_i^+(y)$ include vertex w, and similarly which of $U_i^+(x), U_i^+(y)$ include v, is enough to determine $U_i^+(v)$. Thus there are 32 choices. This is also true for $U_i^+(y)$ since the situation is completely symmetric (swapping v, w with y, x, respectively). Hence, we can record the desired information with a number $\delta_i \in \left[2^{10} d^2 \binom{d}{2}\right]$.

Let $\mathcal{R}_t$ denote the set of sequences $(r_1, \dots, r_t)$ with each $r_i \in [s]$ such that Algorithm 1 does not stop in at most t steps when using $r_1, \dots, r_t$ for the random choices. Also, let $\mathcal{L}_t$ denote the set of logs defined by the algorithm on these sequences. The following lemma follows from the discussion above.

Lemma 5. *For each $t \geqslant 1$, there is a bijection between the two sets $\mathcal{R}_t$ and $\mathcal{L}_t$.*

Next, we bound $|\mathcal{L}_t|$ from above when t is large. To do so we need to count some specific Dyck words where each descent is weighted with some integer: Given a set $E = \{(l_1, w_1), \dots, (l_k, w_k)\}$ of couples of positive integers with all l_j's distinct, we let $C_{t,E}$ be the number of Dyck words of semilength t where each descent has length in the set $\{l_1, \dots, l_k\}$, and each descent of length l_j is weighted with an integer in $[w_j]$.

For our purposes, we take $E := \{(l_1, w_1), \dots, (l_5, w_5)\}$, where (l_j, w_j) is determined by the characteristics of Bad Event j: l_j is the number of vertices that are reset, and w_j is an upper bound on the number of values the corresponding pair (γ_i, δ_i) can take in the log during the corresponding i-th iteration of the algorithm. Thus, following the discussion of bad events above, we take:

- $l_1 = q + 1$ and $w_1 = \binom{\Delta}{q} d^{q+2}$
- $l_2 = 2$ and $w_2 = (q+2)d^3$
- $l_3 = 3$ and $w_3 = 2^7 \Delta^3 \binom{d}{2}$
- $l_4 = 4$ and $w_4 = 2^{12} \Delta^4 d \binom{d}{2}$
- $l_5 = 5$ and $w_5 = 2^{11} \Delta^4 d^2 \binom{d}{2}$

In our logs we deal with *partial* Dyck words that are weighted as above. The difference between the number of 0s and 1s in the partial Dyck word corresponds to the number of big vertices $u \in B$ for which $U^+(u)$ is currently not set; we call this quantity its *defect*. Observe that partial weighted Dyck words of semilength t and defect k can be mapped injectively to weighted Dyck words of semilength $t+k$ by adding k occurrences of 011 at the end, where each of the k new descents of length 2 are weighted with, say, the number 1. Since $k \leqslant n = |V(G)|$, we obtain the following lemma.

Lemma 6. $|\mathcal{L}_t| \leqslant \sum_{k=0}^{n} C_{t+k,E}$.

In our setting, n and s are fixed while t varies; thus, to prove that $|\mathcal{L}_t| \in o(s^t)$, it is enough by the above lemma to show that $C_{t,E} \in o(s^t)$. In order to bound $C_{t,E}$ from above, we follow [6] and use a bijection between Dyck words and rooted plane trees.

Lemma 7. *The number $C_{t,E}$ is equal to the number of weighted rooted plane trees on $t+1$ vertices, where each vertex has a number of children in $E \cup \{0\}$, and for each $i \in [5]$ each vertex with l_i children is weighted with an integer in $[w_i]$ (leaves are not weighted).*

The proof of this lemma is essentially that of Lemma 7 in [6].

Now we use generating functions and the analytic method described e.g. in [5, Section 1.2]. Let

$$y(x) := \sum_{t \geqslant 1} C_{t,E} x^t$$

denote the generating function associated to our objects, and let

$$\phi(x) := 1 + \sum_{i=1}^{5} w_i x^{l_i}.$$

Then $y(x)$ satisfies $y(x) = x\phi(y(x))$. As noted in [5, Theorem 5] (see also [7, p.278, Proposition IV.5]), the following asymptotic bound holds for $C_{t,E}$.

Theorem 8. *Let R denote the radius of convergence of ϕ and suppose that $\lim_{x \to R^-} \frac{x\phi'(x)}{\phi(x)} > 1$. Then there exists a unique solution $\tau \in (0, R)$ of the equation $\tau\phi'(\tau) = \phi(\tau)$, and $C_{t,E} = O(\gamma^t)$, where $\gamma := \phi(\tau)/\tau$.*

The radius of convergence of our function ϕ is $R = \infty$, and $\lim_{x \to \infty} \frac{x\phi'(x)}{\phi(x)} > 1$, thus the theorem applies. For our purposes, it is not necessary to compute exactly τ, a good upper bound on $\gamma = \phi(\tau)/\tau$ will be enough. To obtain such an upper bound we use the following lemma.

Lemma 9. *For every $x \in (0, R)$, if $x\phi'(x)/\phi(x) < 1$ then $\phi(\tau)/\tau < \phi(x)/x$.*

Proof. As noted in [7, Note IV.46] the function $x\phi'(x)/\phi(x)$ is increasing on $(0, R)$. Thus, $x\phi'(x)/\phi(x) < 1$ if and only if $x < \tau$. Consider the function $x\phi'(x)/\phi(x)$ on $(0, \tau)$. Since $x\phi'(x)/\phi(x) < 1$, we have $x\phi'(x) - \phi(x) < 0$. Moreover, since $\frac{\partial}{\partial x}(\frac{\phi(x)}{x}) = \frac{x\phi'(x) - \phi(x)}{x^2}$, we see that $\frac{\phi(x)}{x}$ is decreasing on $(0, \tau)$. Hence, $\frac{\phi(x)}{x} > \frac{\phi(\tau)}{\tau}$. □

Using these tools we can bound γ from above.

Lemma 10. *$\gamma < s$ when d is large enough.*

Proof. We will use Lemma 9. Let $\epsilon_1 > 0$ be fixed (at the end of the proof ϵ_1 will be taken small enough as a function of $q = 13$). Let

$$x := \left(\frac{1}{q(1+\epsilon_1)w_1}\right)^{1/(q+1)}.$$

We claim that $x\phi'(x)/\phi(x) < 1$ when d is large enough. First, let us give some intuition: If we counted only the subset of weighted Dyck words where each descent is of length $l_1 = q+1$ and is weighted with an integer in $[w_1]$, then the corresponding function ϕ would be $\phi(x) = 1 + w_1 x^{q+1}$, and one would get $\tau = \left(\frac{1}{qw_1}\right)^{1/(q+1)}$. As it turns out, the value of τ for our function of ϕ tends to that one (from below) as $d \to \infty$, hence our choice of $\left(\frac{1}{qw_1}\right)^{1/(q+1)}$, slightly scaled down, for x.

To show $x\phi'(x)/\phi(x) < 1$, we make the following observations, each of which is self evident:

- $x\phi'(x) = \sum_{i=1}^{5} l_i w_i x^{l_i}$
- $\phi(x) \geqslant 1 + w_1 x^{q+1}$
- $x = O\left(\frac{1}{d^2}\right)$
- $l_i w_i = O(d^{2l_i - 1})$ for each $i \in [2, 5]$.

It follows that

$$\frac{\sum_{i=2}^{5} l_i w_i x^{l_i}}{\phi(x)} = O\left(\frac{1}{d}\right)$$

and

$$\frac{x\phi'(x)}{\phi(x)} \leqslant \frac{(q+1)w_1 x^{q+1}}{1 + w_1 x^{q+1}} + O\left(\frac{1}{d}\right) = \frac{\frac{1}{1+\epsilon_1} \cdot \frac{q+1}{q}}{1 + \frac{1}{(1+\epsilon_1)q}} + O\left(\frac{1}{d}\right) = \frac{\frac{1}{1+\epsilon_1} \cdot \frac{q+1}{q}}{\frac{1}{1+\epsilon_1} \cdot \frac{q+1}{q} + \frac{\epsilon_1}{1+\epsilon_1}} + O\left(\frac{1}{d}\right).$$

Thus $x\phi'(x)/\phi(x) < 1$ when d is large enough, as claimed. Hence, to prove the lemma it is enough to show that $\phi(x)/x < s$ for d large enough, by Lemma 9.

Observe that

$$\frac{\phi(x)}{x} = \frac{1}{x} + w_1 x^q + O(d).$$

Since $s = \binom{d-q}{2} - 3d = \Theta(d^2)$, to prove that $\phi(x)/x < s$ for d large enough it is enough to show that $1/x + w_1 x^q < (1-\delta)s$ for some fixed $\delta > 0$. Let

$$c_{q,\epsilon_1} := (q(1+\epsilon_1))^{1/(q+1)} + \left(\frac{1}{q(1+\epsilon_1)}\right)^{q/(q+1)}.$$

Using that $\binom{a}{b} \leqslant \frac{a^b}{b!}$ and $d \leqslant \Delta/2$, we obtain the following bound:

$$\frac{1}{x} + w_1 x^q = c_{q,\epsilon_1} \left(\binom{\Delta}{q} d^{q+2} \right)^{1/(q+1)} \leqslant c_{q,\epsilon_1} \left(\frac{\Delta^{2q+2}}{2^{q+2} q!} \right)^{1/(q+1)} = c_{q,\epsilon_1} \left(\frac{1}{2^{q+2} q!} \right)^{1/(q+1)} \Delta^2.$$

Since $s = \binom{d-q}{2} - 3d$, for any fixed $\epsilon' > 0$ we have $s \geqslant \frac{1-\epsilon'}{2} d^2 \geqslant \frac{(1-\epsilon')(1/2-\epsilon)^2}{2} \Delta^2$ if d is large enough. Hence, to conclude the proof it suffices to show that the following inequality holds if ϵ, ϵ' and ϵ_1 are chosen small enough:

$$c_{q,\epsilon_1} \left(\frac{1}{2^{q+2} q!} \right)^{1/(q+1)} < \frac{(1-\epsilon')(1/2-\epsilon)^2}{2}. \tag{2}$$

This is true, since $c_{q,0} \left(\frac{1}{2^{q+2} q!} \right)^{1/(q+1)} \simeq 0.12292 < 1/8$ for $q = 13$. □

It follows from Theorem 8 and the above lemma that $C_{t,E} \in o(s^t)$, and hence $|\mathcal{L}_t| \in o(s^t)$, when d is large enough. (To avoid any confusion, let us emphasise that here the $o(\cdot)$ notation is w.r.t. the variable t, that is, we first assume that d is large enough for Lemma 10 to hold, and then when the graph is fixed we let t vary.) Since there are s^t random sequences of length t, Theorem 4 follows from Lemma 5 and Lemma 10.

It follows from Theorem 4 that Algorithm 1 stops on some random sequence, and thus a function U^+ satisfying the properties of Lemma 3 exists. Consider such a function U^+ and the corresponding set of selected edges. Recall that each vertex of G is incident to at most $q+2$ selected edges, as follows from property (1) of Lemma 3. Using Vizing's theorem we recolour the set of selected edges properly using $q+3$ new colours, say from the set $[\Delta+3, \Delta+q+5]$. Let c' denote the resulting edge colouring of G. That is, $c'(e) := c(e)$ if e was not selected, and $c'(e)$ denotes the new colour of e if e was selected.

For each bad edge $uv \in M$, choose one of its two endpoints, say u, and mark one edge uw for some vertex $w \in U^+(u)$, with $w \neq v$ in case $v \in U^+(u)$. It follows from property (4) of Lemma 3 that marked edges form a matching, and that each bad edge is incident to exactly one marked edge. Recolouring all marked edges with a new colour, say colour $\Delta+q+6$, we obtain a proper colouring c'' of G with $\Delta+q+6$ colours such that $S_{c''}(u) \neq S_{c''}(v)$ for all edges $uv \in E(G)$ with $u, v \in B$ and $d_G(u) = d_G(v)$. Indeed, if uv is a bad edge this holds because there is exactly one marked edge incident to uv, and it is distinct from uv itself. If uv is not a bad edge, then by definition u and v see distinct sets of colours in the colouring c when considering only non-selected edges. Since marked edges form a subset of selected edges, we see that $S_{c''}(u) \neq S_{c''}(v)$ as desired.

Finally, consider edges $uv \in E(G)$ with $u, v \in A$ that are isolated in $G[A]$ (i.e. such that all neighbours of u, v outside $\{u, v\}$ are big) with $d_G(u) = d_G(v)$. Recall that $d_G(u) = d_G(v) \geqslant 2$, since uv is not isolated in G. Recall also that in the initial colouring c of G we had $S_c(u) \cap S_c(v) = \{c(uv)\}$, that is, u and v saw no common colour except for that of uv. If uv is fragile, then at least one of u, v is such that no incident edge was selected, by property (2) of Lemma 3, and hence $S_{c''}(u) \neq S_{c''}(v)$ (since marked edges

form a subset of selected edges). If uv is not fragile, then $d_G(u) = d_G(v) \geqslant q + 4$ by definition. Since at most $q+2$ edges incident to u were selected, and same for v, we see that u and v are each incident to a non-selected edge distinct from uv. It follows that $S_{c''}(u) \neq S_{c''}(v)$.

4. Small vertices

At this point, we know that $S_{c''}(u) \neq S_{c''}(v)$ for every edge $uv \in E(G)$ with $u, v \in B$, and for every edge $uv \in E(G)$ with $u, v \in A$ which is isolated in $G[A]$. However, we could have $S_{c''}(u) = S_{c''}(v)$ for some non-isolated edges uv of $G[A]$. Let A' be the subset of vertices of A that are not incident to an isolated edge of $G[A]$. In this section we modify the colouring c'' on the graph $G[A']$ only, and make sure that $S_{c''}(u) \neq S_{c''}(v)$ for every $uv \in E(G)$ with $u, v \in A'$. Since this has no effect on the sets $S_{c''}(u)$ for $u \in B \cup (A - A')$, the resulting colouring will be an AVD-colouring of G.

First, uncolour every edge of $G[A']$ and fix an arbitrary ordering of these edges. We colour these edges one by one using the following iterative algorithm; at all times, we let c''' denote the current partial colouring of G. Consider the first uncoloured edge uv in the ordering. Let $s := \lceil 2\epsilon\Delta \rceil$. Since $(d_G(u) - 1) + (d_G(v) - 1) \leqslant 2(d-1) \leqslant \Delta - 2\epsilon\Delta$, there are at least $s+q+6$ available choices for the edge uv in order to maintain a proper (partial) colouring. In case all other edges around u are already coloured, we possibly remove one colour from the set of available choices as follows: Say that a neighbour w of u in $A - \{v\}$ is *dangerous* for u if $d_G(u) = d_G(w)$, all edges incident to w are already coloured, and $S_{c'''}(w) = S_{c'''}(u) \cup \{i\}$ for some colour $i \in [\Delta + q + 6]$; the colour i is a *dangerous colour* for u. Dangerous neighbours and colours for v are defined similarly. If u has exactly one dangerous neighbour, remove the corresponding dangerous colour from the set of available choices. Do the same for v. Thus, there are at least $s+q+4 \geqslant s$ available choices remaining for the edge uv. Colour uv with a colour chosen at random among the first s colours available.

As with the algorithm from the previous section we define some bad events that could happen after colouring the edge uv. Here, we only need to consider one type of bad event:

The edge uv received a colour that was dangerous for u or v.

If such an event happens, consider a corresponding dangerous neighbour w, say it was dangerous for u. Let F denote the set of edges incident to u in $G[A']$ that are distinct from uv. Observe that $|F| \geqslant 2$, since otherwise we would have removed the dangerous colour for u from the available choices. Our ordering of the edges of $G[A']$ induces an ordering of the edges in F; it will be convenient to see this ordering as a *cyclic* ordering. With these notations, the bad event is handled as follows:

Uncolour uv and the edge just after uw in the cyclic ordering of F.

After possibly handling one such bad event, the algorithm proceeds with the next uncoloured edge in this fashion, until every edge is coloured.

Lemma 11. *If the algorithm terminates, then the resulting colouring c''' is an AVD-colouring of G.*

Proof. Consider an edge $uv \in E(G)$ with $d_G(u) = d_G(v)$. We already know that $S_{c'''}(u) \neq S_{c'''}(v)$ if $u, v \in B$, so let us assume that $u, v \in A$. Arguing by contradiction, suppose that $S_{c'''}(u) = S_{c'''}(v)$. Recall that $G[A']$ has no isolated edges, thus there is at least one edge incident to u or v which is distinct from uv in $G[A']$. Let e be the last edge coloured by the algorithm among all such edges. Suppose w.l.o.g. that e is incident to v, say $e = vw$. Then, just before the edge vw was coloured for the last time, vertex u was dangerous for v, with dangerous colour $c'''(vw)$. Hence, a bad event has been triggered after colouring vw. The bad event that was handled by the algorithm could have been the one with edge uv, or another one corresponding to another edge incident to v or w. In any case, the edge vw got uncoloured, a contradiction. □

Thanks to the above lemma, to conclude the proof it only remains to show that the algorithm terminates with nonzero probability, which we do now.

Theorem 12. *The algorithm terminates with high probability.*

Proof. The proof is very similar to the corresponding proof in the previous section (but simpler). Let us encode the first t steps (iterations) of an execution of the algorithm with a corresponding *log* $(W, \gamma, \delta, c''')$, where

- W is a partial Dyck word of semilength t, obtained by adding a 0 (a 1) each time an edge is coloured (uncoloured, respectively);
- $\gamma = (\gamma_1, \ldots, \gamma_t)$;
- $\delta = (\delta_1, \ldots, \delta_t)$;
- c''' is the current colouring at the end of the t-th iteration.

For each $i \in [t]$, we let $\gamma_i := -1$ and $\delta_i := -1$ in case no bad event was triggered during the i-th iteration. Otherwise, if a bad event occurred, say involving a vertex w that was dangerous for one of the two endpoints of the edge uv coloured during the iteration, we let $\gamma_i \in [2d]$ identify vertex w knowing uv (recall that u and v have degree at most d). Observe that this identifies also the extra edge that is uncoloured (besides the edge uv).

Then, we let $\delta_i \in [2]$ identify the colours of the two edges that got uncoloured, assuming we know these two edges and the colouring c''' at the end of iteration i. Observe that we already know the *set* of colours that was used for these two edges, these are the two colours appearing around w but not around the vertex (u or v) that triggered the bad event. Thus it only remains to specify the mapping of these two colours to the two edges (2 possibilities).

Reading W and γ from the beginning, one can deduce which subset of the edges of $G[A']$ was coloured at any time during the execution. Then using the colouring c''' at the end of the t-th iteration and working backwards, we can reconstruct the colouring c''' at any time during the execution using γ and δ, and deduce in particular which random choice

was made for the edge under consideration during the i-th iteration. Hence, the log $(W, \gamma, \delta, c''')$ uniquely determines the t random choices that were made.

As before, we see a random choice as a number in $[s]$. Let $\mathcal{R}_t$ denote the set of random vectors $(r_1, \ldots, r_t)$ of length t, where each entry is a number in $[s]$. Let $\mathcal{L}_t$ denote the set of logs after t steps resulting from executions of the algorithm that last for at least t steps. By the discussion above, there is an injective mapping from $\mathcal{L}_t$ to $\mathcal{R}_t$. Since $|\mathcal{R}_t| = s^t$, to prove Theorem 12 it only remains to show that $|\mathcal{L}_t| = o(s^t)$.

Here, a rather crude counting will do. First, we count the partial Dyck words W of semilength t that can appear in our logs. Each such word has only descents of length 2. They can be mapped to Dyck words of semilength t simply by adding the missing 1s at the end. Notice that each Dyck word of semilength t is the image of at most two such partial Dyck words. (Two of our partial Dyck words have the same image iff they are the same except one ends with 0 and the other ends with 011.) Hence, the number of our partial Dyck words of semilength t is at most twice the number of Dyck words of semilength t, and thus is at most $2 \cdot 4^t$.

Next, given a log $(W, \gamma, \delta, c''')$, the indices $i \in [t]$ such that $\gamma_i \neq -1$ and $\delta_i \neq -1$ correspond to descents of W. Thus there are at most $t/2$ such indices, and we see that the number of possible pairs (γ, δ) for a given W is at most $(2d)^{t/2} \cdot 2^{t/2} = (4d)^{t/2} \leqslant (2\Delta)^{t/2}$.

Finally, the number of partial colourings c''' of G is at most $|E(G)|^{\Delta+q+7}$, and is in particular independent of t.

Assuming that Δ is large enough so that $s = \lceil 2\epsilon\Delta \rceil > (32\Delta)^{1/2}$, we conclude that

$$|\mathcal{L}_t| \leqslant 2 \cdot 4^t \cdot (2\Delta)^{t/2} \cdot |E(G)|^{\Delta+q+7} = O\left((32\Delta)^{t/2}\right) = o(s^t),$$

as desired. □

Acknowledgements

We thank Marthe Bonamy for inspiring discussions.

References

[1] P. N. Balister, E. Győri, J. Lehel, and R. H. Schelp. Adjacent vertex distinguishing edge-colorings. *SIAM J. Discrete Math.*, 21(1):237–250, 2007.

[2] M. Bonamy, N. Bousquet, and H. Hocquard. Adjacent vertex-distinguishing edge coloring of graphs. In J. Nešetřil and M. Pellegrini, editors, *The Seventh European Conference on Combinatorics, Graph Theory and Applications*, pages 313–318, 2013.

[3] M. Bonamy and J. Przybyło. On the neighbor sum distinguishing index of planar graphs. *Journal of Graph Theory*, 85(3):669–690, 2017. arXiv:1408.3190.

[4] R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, fourth edition, 2010.

[5] M. Drmota. Combinatorics and asymptotics on trees. *Cubo Journal*, 6(2), 2004.

[6] L. Esperet and A. Parreau. Acyclic edge-coloring using entropy compression. *European Journal of Combinatorics*, 34(6):1019–1027, 2013. arXiv:1206.1535.
[7] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, New York, NY, USA, 2009.
[8] E. Flandrin, A. Marczyk, J. Przybyło, J.-F. Saclé, and M. Woźniak. Neighbor sum distinguishing index. *Graphs and Combinatorics*, 29(5):1329–1336, Sep 2013.
[9] C. Greenhill and A. Ruciński. Neighbour-distinguishing edge colourings of random regular graphs. *Electronic Journal of Combinatorics*, 13:#R77, 2006.
[10] H. Hatami. $\Delta + 300$ is a bound on the adjacent vertex distinguishing edge chromatic number. *J. Combin. Theory Ser. B*, 95(2):246–256, 2005. arXiv:math/0701012.
[11] M. Horňák, D. Huang, and W. Wang. On neighbordistinguishing index of planar graphs. *Journal of Graph Theory*, 76(4):262–278, 2014.
[12] M. Horňák and M. Woźniak. On neighbour-distinguishing colourings from lists. *Discrete Mathematics & Theoretical Computer Science*, 4(2), 2012.
[13] J. Kahn. Asymptotically good list-colorings. *Journal of Combinatorial Theory, Series A*, 73(1):1–59, 1996.
[14] J. Kwaśny and J. Przybyło. Asymptotically optimal bound on the adjacent vertex distinguishing edge choice number. 2017. arXiv:1705.01637.
[15] M. Molloy and B. Reed. Near optimal list colorings. *Random Struct. Algorithms*, 17(3-4):376–402, 2000.
[16] R. A. Moser and G. Tardos. A constructive proof of the general Lovász Local Lemma. *J. ACM*, 57(2):Art. 11, 2010. arXiv:0903.0544.
[17] J. Przybyło. Neighbor distinguishing edge colorings via the combinatorial nullstellensatz. *SIAM Journal on Discrete Mathematics*, 27(3):1313–1322, 2013.
[18] J. Przybyło. A note on asymptotically optimal neighbour sum distinguishing colourings. 2017. arXiv:1703.00406.
[19] Y. Wang, J. Cheng, R. Luo, and G. Mulley. Adjacent vertex-distinguishing edge coloring of 2-degenerate graphs. *Journal of Combinatorial Optimization*, 31(2):874–880, Feb 2016.
[20] Z. Zhang, L. Liu, and J. Wang. Adjacent strong edge coloring of graphs. *Appl. Math. Lett.*, 15(5):623–626, 2002.