Low-Memory Algorithms for Online Edge Coloring

Prantar Ghosh
Georgetown University, Washington, DC, USA

Manuel Stoeckl
Dartmouth College, Hanover, NH, USA

Abstract

For edge coloring, the online and the W-streaming models seem somewhat orthogonal: the former needs edges to be assigned colors immediately after insertion, typically without any space restrictions, while the latter limits memory to be sublinear in the input size but allows an edge's color to be announced any time after its insertion. We aim for the best of both worlds by designing small-space online algorithms for edge coloring.

Our online algorithms significantly improve upon the memory used by prior ones while achieving an $O(1)$-competitive ratio. We study the problem under both (adversarial) edge arrivals and vertex arrivals. Under vertex arrivals of any n-node graph with maximum vertex-degree Δ, our online $O(\Delta)$-coloring algorithm uses only semi-streaming space (i.e., $\tilde{O}(n)$ space, where the $\tilde{O}(.)$ notation hides polylog(n) factors). Under edge arrivals, we obtain an online $O(\Delta)$-coloring in $\tilde{O}(n\sqrt{\Delta})$ space. We also achieve a smooth color-space tradeoff: for any $t = O(\Delta)$, we get an $O(\Delta t(\log^2 \Delta))$-coloring in $\tilde{O}(n\sqrt{\Delta/t})$ space, improving upon the state of the art that used $\tilde{O}(n\Delta/t)$ space for the same number of colors.

The improvements stem from extensive use of random permutations that enable us to avoid previously used colors. Most of our algorithms can be derandomized and extended to multigraphs, where edge coloring is known to be considerably harder than for simple graphs.

2012 ACM Subject Classification Mathematics of computing → Graph coloring; Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Edge coloring, streaming model, online algorithms

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.71

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2304.12285

Funding *Prantar Ghosh*: Supported in part by NSF under award 1918989. Part of this work was done while the author was at DIMACS, Rutgers University, supported in part by a grant (820931) to DIMACS from the Simons Foundation.
Manuel Stoeckl: This work was supported in part by the National Science Foundation under award 2006589.

1 Introduction

A proper edge-coloring of a graph or a multigraph colors its edges such that no two adjacent edges share the same color. The goal is to use as few colors as possible. Any graph with maximum vertex-degree Δ trivially requires Δ colors to be properly edge-colored. Vizing's celebrated theorem [46] asserts that $\Delta + 1$ colors suffice for any simple graph.[1] Constructive polynomial-time algorithms exist for $(\Delta + 1)$-edge-coloring in the classical offline setting [39], and these are likely to be optimal with respect to the number of colors: distinguishing between whether the edge-chromatic number (i.e., the minimum number of colors needed to edge-color a graph) of a simple graph is Δ or $\Delta + 1$ is NP-hard [33].

[1] For multigraphs, $3\Delta/2$ colors are necessary and sufficient. [44]

The edge-coloring problem finds applications in various practical scenarios, including switch routing [2], round-robin tournament scheduling [34], call scheduling [27], optical networks [42], and link scheduling in sensor networks [30]. In applications like switch routing, where the underlying graph evolves through edge insertions, the color assignments must be made instantly and irrevocably. This is modeled by the *online* edge coloring problem. Due to its restrictions, an online algorithm cannot obtain a $(\Delta+1)$-coloring [11]. Nevertheless, the simple greedy algorithm, which colors every edge with the first available color not assigned to any neighbor, obtains a $(2\Delta-1)$-coloring since each edge can have at most $2\Delta-2$ neighboring edges. Thus, the competitive ratio achieved is $2-o(1)$ (since the optimum is Δ or $\Delta+1$). Bar-Noy, Motwani, and Naor [11] showed that no online algorithm can outperform this greedy algorithm. However, they proved this only for graphs with max-degree $\Delta = O(\log n)$. They conjectured that for $\Delta = \omega(\log n)$, it is possible to get better bounds – specifically, a $(1+o(1))\Delta$-coloring. For this regime of Δ, several works [2, 10, 23, 18, 43, 35, 41] have studied online edge coloring with the goal of surpassing the greedy algorithm and, even further, of resolving the said conjecture. Additionally, other variants of the online problem have been investigated [29, 38, 28]. However, all prior works assume full storage of all graph edges and their colors in memory.

With the ubiquity of big data in the modern world, the assumption of storing entire graphs in memory becomes impractical. Even graphs like communication and internet routing networks that motivate the study of edge coloring often turn out to be large-scale networks. This challenge has given rise to big-graph processing models such as *graph streaming* which, like the online model, sequentially accesses the graph edges, but only retains a small summary. There is an immediate barrier for edge coloring in this setting: reporting all edge colors at the end of the stream would use space linear in the number of edges. To remedy this, a natural extension of the model, called the *W-streaming model*, allows reporting the output in streaming fashion. Here, an algorithm with limited working memory stores information about both the input graph and the output coloring and periodically streams or announces edge colors. Unlike the online model, here we don't need to assign a color to an incoming edge right away, and can defer it. However, due to space constraints, we are not able to remember all the previously announced colors. Note that this makes even the greedy $(2\Delta-1)$-coloring algorithm hard, if not impossible, to implement in this model.

In this work, we aim to get the best of both worlds of the online and the streaming models by designing *low-memory* online algorithms for edge coloring. This addresses the need for immediate color assignment in modern scenarios while optimizing space usage. We achieve an $O(1)$-competitive ratio, i.e., a color bound of $O(\Delta)$. Note that no prior work in the sublinear-space setting has achieved an $O(\Delta)$-coloring W-streaming algorithm, let alone an online algorithm. For adversarial edge-arrival streams, we get an online $O(\Delta)$-coloring in $\widetilde{O}(n\sqrt{\Delta})$ space[2], significantly reducing the space compared to prior online algorithms with only a constant factor increase in colors. We can smoothly trade off space with colors, obtaining an $\widetilde{O}(\Delta t)$-coloring in $\widetilde{O}(n\sqrt{\Delta/t})$ space. This improves upon the state of the art [21, 5] which used $\widetilde{O}(n\Delta/t)$ space for the same color bound. Furthermore, for natural and well-studied settings of vertex-arrival in general graphs and one-sided vertex arrival in bipartite graphs, we enhance the space usage to $\widetilde{O}(n)$, the prevalent *semi-streaming* memory regime for graph streaming problems. Most of our algorithms generalize to multigraphs and can be made deterministic.

[2] Throughout the paper, the $\widetilde{O}(.)$ notation hides polylog(n) and polylog(Δ) factors.

We remark that apart from being interesting online algorithms on their own, our results can be also viewed as strengthening W-streaming algorithms with the guarantee of online output-reporting. In particular, they contribute an important conceptual message: the state-of-the-art W-streaming space bound for $O(\Delta)$ edge coloring can be matched *without* the exclusive power of the W-streaming model (allowing delayed assignment of edge colors); we can report edge colors online.

1.1 Our Results and Contributions

We study edge-coloring in the online model with sublinear (i.e., $o(n\Delta)$) memory, under (adversarial) edge-arrivals as well as vertex-arrivals. These results are summarized in Table 1, and their corollaries for the W-streaming model in Table 2. The tables mention the state of the art, for comparison.

Table 1 Our results in the online model. Here, $t \leq \Delta$ is any positive integer. Algorithms marked with a $\star$ require oracle randomness for randomized algorithms and an advice string computable in exponential time for deterministic.

Arrival	Algorithm	Colors	Space	Graph	Reference
Edge	Randomized	$\left(\frac{e}{e-1}+o(1)\right)\Delta$	$\widetilde{O}(n\Delta)$	Simple	[35]
	Randomized	$O(\Delta)$	$\widetilde{O}(n\sqrt{\Delta})$	Simple	Theorem 1
	Deterministic	$(2\Delta-1)t$	$O(n\Delta/t)$	Multigraph	[5]
	Deterministic	$\widetilde{O}(\Delta t)$	$\widetilde{O}(n\sqrt{\Delta/t})\star$	Multigraph	Theorem 3 + Lemma 5
Vertex	Randomized	$(1.9+o(1))\Delta$	$\widetilde{O}(n\Delta)$	Simple	[43]
	Randomized	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Theorem 6
	Deterministic	$2\Delta-1$	$O(n\Delta)$	Multigraph	Greedy folklore
	Deterministic	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Theorem 7
One sided vertex	Randomized	$(1+o(1))\Delta$	$\widetilde{O}(n\Delta)$	Simple	[23]
	Randomized	1.533Δ	$\widetilde{O}(n\Delta)$	Multigraph	[41]
	Randomized	5Δ	$\widetilde{O}(n)\star$	Multigraph	Theorem 6
	Deterministic	$2\Delta-1$	$O(n\Delta)$	Multigraph	Greedy folklore
	Deterministic	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Theorem 7

Table 2 Our results in the W-streaming edge-arrival model. Here, $s \leq \Delta/2$ is any positive integer. Results marked with $\star$ require oracle randomness for randomized algorithms and an advice string computable in exponential time for deterministic.

Algorithm	Colors	Space	Graph	Reference
Randomized	$O(\Delta^2/s)$	$\widetilde{O}(ns)$	Simple	[21]
Randomized	$O(\Delta)$	$\widetilde{O}(n\sqrt{\Delta})$	Simple	Corollary 2
Deterministic	$(1-o(1))\Delta^2/s$	$O(ns)$	Simple	[5]
Deterministic	$\widetilde{O}(\Delta^2/s)$	$\widetilde{O}(n\sqrt{s})\star$	Multigraph	Corollary 4 + Lemma 5

Edge-arrival model

Here we design both online and W-streaming algorithms.

▶ **Theorem 1.** *Given any adversarial edge-arrival stream of a simple graph, there is a randomized algorithm for online $O(\Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

Previously, there was no sublinear space online algorithm known for $O(\Delta)$-coloring. As observed in Table 1, all prior algorithms need $\Theta(n\Delta)$ space in the worst case to achieve a color bound of $O(\Delta)$.

Note that Theorem 1 immediately implies a randomized W-streaming algorithm with the same space and color bounds. Although immediate, we believe that it is important to note it as a corollary.

▶ **Corollary 2.** *Given an adversarially ordered edge stream of any simple graph, there is a randomized W-streaming algorithm for $O(\Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

The above result improves upon the state of the art algorithms of [21, 5] which, as implied by Table 2, only obtain $\omega(\Delta)$-colorings for $o(n\Delta)$ space (the non-trivial memory regime in W-streaming). In fact, we improve upon them by a factor of $\Omega(\sqrt{\Delta})$ in space for $O(\Delta)$-coloring.

Further, we prove that we can make the above algorithms deterministic, and able to handle multigraphs, at the cost of only a polylogarithmic factor in the number of colors used. Once again, the online algorithm is also a W-streaming algorithm.

▶ **Theorem 3.** *Given an adversarially ordered edge-arrival stream of any multigraph, there is a deterministic algorithm for online $O(\Delta \log^2 \Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

▶ **Corollary 4.** *Given an adversarially ordered edge stream of any multigraph, there is a deterministic W-streaming algorithm for $O(\Delta \log^2 \Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

Furthermore, in each case, we can achieve a smooth tradeoff between the number of colors and the memory used. This is implied by a framework captured in the following lemma.

▶ **Lemma 5.** *Suppose that we are given an online $f(n, \Delta)$-space streaming algorithm $\mathcal{A}$ for $O(\Delta)$-coloring any n-node multigraph with max-degree Δ under adversarial edge arrivals. Then, for any $t \geq 1$, there is a online streaming algorithm $\mathcal{B}$ for $O(\Delta t)$-coloring the same kind of graphs under adversarial edge arrivals using $f(n/t, \Delta t) + \widetilde{O}(n)$ bits of space.*

For the online model, the above lemma combined with Theorem 3 immediately gives the tradeoff of $\widetilde{O}(\Delta t)$ colors and $\widetilde{O}(n\sqrt{\Delta/t})$ space for any $t = O(\Delta)$, as claimed in Table 1. In other words, combined with Corollary 4, it implies the W-streaming bounds of $\widetilde{O}(\Delta^2/s)$ colors and $O(n\sqrt{s})$ space for any $s = O(\Delta)$, as claimed in Table 2.[3] Note that our results match the tradeoff obtained by the state of the art for $t = \Theta(\Delta)$ and $s = O(1)$, and strictly improve upon them for $t = o(\Delta)$ and $s = \omega(1)$.

[3] We use the parameter t for online algorithms and s for W-streaming, where $t \cdot s = \Delta$, so that a reader can easily compare our bounds with the "ideal" bounds of $O(\Delta)$ colors and $\widetilde{O}(n)$ space in the respective models by smoothly growing t or s from 1 to Δ.

Vertex-arrival Model

We now turn to the weaker vertex-arrival model. The online edge-coloring problem has been widely studied in this setting as well (see Section 1.2 for a detailed discussion). Our online algorithms obtain significantly better space bounds than the edge-arrival setting.

▶ **Theorem 6.** *Given any adversarial vertex-arrival stream of a multigraph, there is a randomized online $O(\Delta)$-edge coloring algorithm using $\widetilde{O}(n)$ bits of space. It works even against an adaptive adversary and uses $\widetilde{O}(n\Delta)$ oracle random bits.*[4] *In particular, for one-sided bipartite vertex arrivals, there is an algorithm using 5Δ colors.*

Thus, at the cost of only a constant factor in the number of colors, we can improve the memory usage from $\widetilde{O}(n\Delta)$ to $\widetilde{O}(n)$ for vertex-arrival streams. Since this algorithm immediately implies a W-streaming algorithm with the same bounds, we see that for vertex-arrival streams, $O(\Delta)$-coloring can be achieved in semi-streaming space, the most popular space regime for graph streaming. Behnezhad et al. [12] mentioned that "a major open question is whether [the number of colors for W-streaming edge-coloring] can be improved to $O(\Delta)$ while also keeping the memory near-linear in n." Our results answer the question in the affirmative for the widely studied model of vertex-arrival streams.

Further, we show that the algorithm can be made deterministic using $\widetilde{O}(n)$ bits of *advice* instead of $\widetilde{O}(n\Delta)$ bits of oracle randomness. By picking a uniformly random advice string, the updated algorithm can alternatively be used as a robust algorithm (see Definition 12) with $1/\operatorname{poly}(n)$ error; the advice can also be computed in exponential time.

▶ **Theorem 7.** *Given any adversarial vertex-arrival stream of a multigraph, there is a deterministic online $O(\Delta)$-edge-coloring algorithm using $\widetilde{O}(n)$ bits of space, using $\widetilde{O}(n)$ bits of advice.*

An interesting special case of the vertex-arrival model is the one-sided vertex-arrival setting for bipartite graphs. Here, the vertices on one side of the bipartite graph are fixed, while the vertices on the other side arrive in a sequence along with their incident edges. A couple of works [23, 41] have studied online edge-coloring specifically in this model. We design low-memory online algorithms in this setting and use them as building blocks for our algorithms in the more general settings of vertex-arrival and edge-arrival. These algorithms may be of independent interest due to practical applications of the one-sided vertex-arrival model; moreover, the randomized algorithm here uses only 5Δ colors (as opposed to our other algorithms where the hidden constants in $O(\Delta)$ are rather large).

Finally, we present a lower bound on the space requirement of a deterministic online edge-coloring algorithm. To the best of our knowledge, this is the first non-trivial space lower bound proven for an online edge-coloring algorithm.

▶ **Theorem 8.** *For $\Delta \leq \varepsilon n$ for a sufficiently small constant ε, any deterministic online algorithm that edge-colors a graph using $(2 - o(1))\Delta$ colors requires $\Omega(n)$ space.*

[4] The use of oracle randomness here is not a big deal. In practice (where we assume cryptographic pseudo-random number generators exist), it is straightforward to generate the bits of the oracle random string on demand, ensuring that computationally bounded systems essentially cannot produce hard inputs for the algorithm. Also, against oblivious adversaries, one can easily modify the randomized vertex arrival algorithm to use only $\widetilde{O}(n)$ random bits, by choosing its random permutations from almost $O(\log n)$-wise independent families, but we skip this to keep the proof simple.

1.2 Related Work

Online model

Online edge-coloring has a rich literature [2, 5, 11, 10, 18, 19, 23, 28, 29, 37, 38, 41, 35, 43]. The seminal work of Bar-Noy, Motwani, and Naor [11] ruled out any online algorithm outperforming the $(2\Delta - 1)$-coloring greedy algorithm that assigns each edge the first available color not used by any adjacent edge. However, this lower bound applies only to graphs with $\Delta = O(\log n)$. They conjectured that for $\Delta = \omega(\log n)$, there exist online $(1 + o(1))\Delta$-coloring algorithms. Although this conjecture remains unresolved, there has been significant progress on it over the years. A number of works [2, 10, 18] considered the problem under *random-order* edge arrivals: Aggarwal et al. [2] showed that if $\Delta = \omega(n^2)$, then a tight $(1 + o(1))\Delta$-coloring is possible. For $\Delta = \omega(\log n)$ (the bound in the said conjecture), Bahmani et al. [10] obtained a 1.26Δ-coloring. Bhattacharya et al. [18] then achieved the best aspect of each result as they attained the tight color bound of $(1 + o(1))\Delta$ for the broad range of $\Delta = \omega(\log n)$, essentially resolving the conjecture for random-order arrivals.

More relevant to our work is the setting of *adversarial-order* edge arrivals. Cohen et al. [23] were the first to make progress on [11]'s conjecture in this regard, obtaining a $(1 + o(1))\Delta$-coloring for bipartite graphs under one-sided vertex arrivals (i.e., the nodes on one side are fixed, and the nodes on the other side arrive one by one with all incident edges). Their algorithm assumes a priori knowledge of the value of Δ. For unknown Δ, they rule out any online algorithm using fewer than $(e/(e-1))\Delta$ colors and also complement this result with a $(e/(e-1) + o(1))\Delta$-coloring algorithm. For bipartite *multigraphs* with one-sided vertex arrivals, Naor et al. [41] very recently proved that 1.533Δ colors suffice, while at least 1.207Δ colors are necessary even for $\Delta = 2$. Saberi and Wajc [43] showed that it is possible to beat the greedy algorithm for $\Delta = \omega(\log n)$ under vertex arrivals in general graphs: they designed a $(1.9 + o(1))\Delta$-coloring algorithm. Recently, Kulkarni et al. [35] made the first progress on the said conjecture for fully general adversarial edge arrivals: they obtained a $(e/(e-1) + o(1))\Delta$-coloring in this model. Note that the focus of all these works was on resolving [11]'s conjecture without any space limitations. Our focus is on designing low-memory online algorithms while staying within a constant competitive ratio. The only prior sublinear-space online edge-coloring algorithm we know was given by Ansari et al. [5]: a (deterministic) online $2\Delta t$-coloring in $O(n\Delta/t)$ space for any $t \leq \Delta$.

Several works [29, 26, 28] have addressed the variant where, given a fixed number of colors, the goal is to color as many edges as possible. Mikkelsen [37, 38] considered online edge coloring with limited advice for the future.

W-Streaming model

The W-streaming model [25] is a natural extension of the classical streaming model for studying problems where the output size is very large, possibly larger than our memory. While the W-streaming literature has considered several graph problems [25, 24, 36, 31], we are aware of only three past works [12, 21, 5] that have studied edge-coloring here. Behnezhad et al. [12] initiated the study of W-streaming edge coloring. They considered the problem for both adversarial-order and random-order streams: using $\widetilde{O}(n)$ bits of working memory, they gave an $O(\Delta^2)$-coloring in the former setting, and a $(2e\Delta)$-coloring in the latter setting. Charikar and Liu [21] improved these results: for adversarial-order streams, for any $s \in [\Omega(\log n), \Delta]$, they presented an $O(\Delta^2/s)$-coloring algorithm that uses $\widetilde{O}(ns)$ space; and for random-order streams, they gave a $(1 + o(1))\Delta$-coloring algorithm using $\widetilde{O}(n)$ space. Both of the said adversarial-order streaming algorithms are, however, randomized. Ansari

et al. [5] designed simple deterministic algorithms achieving the same bounds of $O(\Delta^2/s)$ colors and $\widetilde{O}(ns)$ space. Their algorithm can also be made online at the cost of a factor of 2 in the number of colors. Note that, parameterizing our results in Table 2 appropriately, our algorithms achieve $O(\Delta^2/s)$-colorings in $\widetilde{O}(n\sqrt{s})$ space, matching the state of the art for $s = O(1)$, and strictly improving upon it for $s = \omega(1)$.

The related problem of vertex coloring has a more mature literature in the streaming model [1, 3, 6, 7, 8, 9, 15, 16, 17, 20, 32]. However, due to fundamental differences between the classical streaming and W-streaming models and between the two problems, not many techniques seem to carry over.

Concurrent work

In an independent and parallel work, Behnezhad and Saneian [13] have designed a randomized $\widetilde{O}(n\sqrt{\Delta})$-space W-streaming algorithm for $O(\Delta)$-edge-coloring for edge-arrival streams in simple general graphs. This matches our Corollary 2. Their result generalizes to give, for any $s \in [\sqrt{\Delta}]$, an $O(\Delta^{1.5}/s)$ coloring algorithm in $\widetilde{O}(ns)$ space, while we achieve an $O(\Delta^2/s^2)$-coloring in the same space. They also get an $O(\Delta)$-edge-coloring algorithm for vertex-arrival streams using $\widetilde{O}(n)$ space, similar to our Theorem 6. Note that a crucial difference between the two papers is that most of our algorithms have the additional strong feature of being online, while it is not clear if their W-streaming edge-arrival algorithm can also be implemented in the online setting. Thus, conceptually, our results affirm that to obtain an $O(\Delta)$-coloring in sublinear space, the only advantage of W-streaming over online – enabling delay of color assignment – is not necessary.

In terms of techniques, while both works have some high level ideas in common (e.g., using random offsets/permutations to keep track of colors, or designing a one-sided vertex-arrival algorithm first and building on it to obtain the edge-arrival algorithm), the final algorithms and analyses in the two papers are fairly different.

Another independent work by Chechik, Mukhtar, and Zhang [22] obtains a randomized W-streaming algorithm that colors an edge-arrival stream on general multigraphs using $O(\Delta^{1.5}\log\Delta)$ colors in expectation[5], and $\widetilde{O}(n)$ space in expectation. Unlike us, they make no claims in the online model.

2 Notation and Definitions

Throughout the paper, logarithms are in base 2. The notation $[t]$ indicates the set of integers $\{1, \ldots, t\}$. The notation $\widetilde{O}(x)$ ignores $\mathrm{poly}(\log(n), \log(\Delta))$ factors in x. $A \sqcup B$ gives the disjoint union of A and B. $\mathbb{S}_t$ is the set of permutations over $[t]$, and for any permutation $\sigma \in \mathbb{S}_t$ and $X \subseteq [t]$, we denote $\sigma[X] := \{\sigma_i : i \in X\}$. For any set X, $\binom{X}{k}$ denotes the set of all k-sized subsets of X.

If not otherwise stated, n is the number of vertices in a graph G, V the set of vertices (or $A \sqcup B$ if the graph is bipartite), E the (multi-)set of edges, and Δ is the maximum degree of the graph (counting multiplicity, for multigraphs).

[5] While [22] does not claim this, one can prove their algorithm uses $O(\Delta^{1.5}\log\Delta)$ colors with $\geq 1 - 1/\mathrm{poly}(n)$ probability.

2.1 Models

We consider the following models for presenting edges (to be colored) to an algorithm. In all cases, the set of vertices for the graph is known in advance.

▶ **Definition 9** (Edge-Arrival Model). *With an* edge-arrival stream*, the algorithm is given a sequence of edges in the graph. Each edge is provided as an unordered pair $\{x, y\}$ of vertices in V. In this paper, online algorithms processing edge arrival streams will implement a method* Process$(\{x, y\})$ *which returns the color assigned to the edge. W-streaming algorithms may assign the color for an edge at any time, although all edges must be given a color at the end of the stream. We permit algorithms to output not just integers but also tuples of integers as "colors", since users of the algorithms can easily remap these colors to whatever space they wish.*

▶ **Definition 10** (Vertex-Arrival Model). *In a* vertex-arrival stream*, the algorithm is given a sequence of (vertex, edge-set) pairs (v, M_v), where the edge (multi-)set M_v contains all edges from v to vertices that have been seen earlier in the stream. Online algorithms should report colors for all edges in M_v when (v, M_v) is processed.*

A one-sided vertex-arrival stream *on a bipartite graph with partite sets A, B is a vertex arrival stream where the vertices in one partite set (B) are fixed, and then all the (vertex, edge-set) pairs for the other partite set (A) are given. This means that the stream consists of pairs (v, M_v), where each $v \in A$, and M_v contains all edges from v to vertices in B.*

▶ Remark 11 (Assumption of prior knowledge of Δ). We assume that the maximum degree Δ of G is known in advance. An edge-coloring algorithm for which Δ is not known in advance can be converted to one which is, although one way to do this conversion (by running a new 2Δ-coloring algorithm with a fresh set of colors whenever the maximum degree of graph formed by the input stream doubles) increases the total number of colors used by a constant factor, and uses $O(n \log \Delta)$ bits of space to keep track of the maximum degree. Since the algorithms in this paper already have large constant factors on number of colors used, it is not worth it to optimize the algorithms for the case where Δ is not known in advance.

▶ **Definition 12** (Robust Streaming). *An algorithm is said to be* robust *or* adversarially robust *if it works with $\geq 1 - \delta$ probability even when its input streams are adaptively generated. By "adaptively generated", we mean that the input is produced by an adaptive adversary that sees all outputs of the online (or W-streaming) algorithm, and repeatedly chooses the next element of the stream based on what the algorithm has output so far. See [14] for a more detailed explanation.*

2.2 Basic Definitions

▶ **Definition 13** ((ϵ, k)-wise independent permutation). *A random permutation σ is (ϵ, k)-wise independent if for all distinct $a_1, \ldots, a_k$ in $[n]$, the distribution of σ on $a_1, \ldots, a_k$ has total variation distance $\leq \epsilon$ from uniform. In other words,*

$$\frac{1}{2} \sum_{\text{distinct } b_1, \ldots, b_k \in [n]} \left| \Pr\left[\bigwedge_{i \in [k]} \{\sigma(a_i) = b_i\} \right] - \frac{1}{\prod_{i \in [k]} (n - i + 1)} \right| \leq \epsilon \,.$$

Per [4], while it is not known if there are nontrivial $(0, k)$-wise independent families of permutations for large k and n, one can always construct weighted distributions which have support of size $n^{O(k)}$ and provide $(0, k)$-wise independence. However, sampling from these may not be efficient.

We say a random permutation is almost k-wise independent *when it is (ϵ, k)-wise independent for sufficiently small ϵ.*

We, using a result by [40] on switching networks, give a short lemma describing an efficient construction of (ϵ, k)-wise independent permutations.

▶ **Lemma 14** (Random permutations through switching networks). *Let C be a power of 2, let s be a natural number $\leq C$, and let $\epsilon > 0$. For $r = O(s(\log C)^4 \log \frac{1}{\epsilon})$, there is a map f from $\{0,1\}^r$ to $\mathbb{S}_C$ so that, if U is a uniformly random string of r bits, then $\sigma = f(U)$ is an (ϵ, s) wise independent random permutation.*

Furthermore, for any $i \in [C]$ and $u \in \{0,1\}^r$, with $\sigma = f(u)$, we can evaluate $\sigma(i)$ and $\sigma^{-1}(i)$ in $O(s(\log C)^4 \log \frac{1}{\epsilon} \log C)$ time.

3 Technical Overview

In this section, we give a high-level overview of our algorithms and techniques. We see these techniques as a major contribution of the paper since many of them are used for the first time in the context of W-streaming and online edge coloring. The proofs of our results are given in the full version of this paper.

3.1 General Reductions

Reducing to bipartite case

We show that it is essentially enough to consider bipartite graphs. Suppose that we can partition a general graph into $O(\log n)$ bipartite graphs, each of which has max-degree roughly $\Delta/\log n$, where Δ is the max-degree of the original graph. Then, if we run our algorithm on these bipartite graphs with disjoint palettes, we use colors roughly proportional to Δ. It is known (see [21] or for a similar result, [43, Lemma 2.1]) that such a partition can be done in a randomized way, incurring a multiplicative overhead of just $1 + o(1)$ in the number of colors. In this work, we show that if we are willing to tolerate an $O(1)$ blowup in the number of colors, then this partition can be done deterministically. Since such a primitive is used in multiple edge-coloring algorithms, this deterministic version might be of independent interest. One advantage of this version is that it works against adaptive adversaries, unlike the randomized version which can be shown to be breakable by such an adversary.

We construct this partition using appropriate binary codes for the vertices: the codes are of length $O(\log n)$ and we have a bipartite graph corresponding to each bit, where the bipartition is given by whether the bit is 0 or 1. Now, we need to ensure that (a) every edge goes to "some" bipartition, and (b) the max-degree of a single bipartite graph is not much higher than $\Delta/\log n$. This can be done using codes with constant rate and relative distance, like the expander codes described by [45]. Now we can focus on getting $O(\Delta)$-colorings for bipartite graphs, which would give us asymptotically same number of colors for general graphs.

Note that for the vertex-arrival model, we can go one step farther and assume "one-sided" vertex arrival, i.e., vertices along with their incident edges arrive on only one side of the bipartite graph. This is because we can run two copies of the algorithm, one each for the vertices arriving on either side, with disjoint palettes. This incurs only a factor of 2 in the number of colors.

Space-color tradeoff for multigraphs

We show with Lemma 5 that if an algorithm can handle multigraphs, then we can smoothly tradeoff colors with space. This is one of our motivations behind extending our algorithms to multigraphs. Recall that we reduce the problem to just bipartite graphs. Now the idea for the tradeoff is simple: we arbitrarily group t nodes (for some parameter t) from the same partite set together as a single supernode. Since the vertices on the same partite set do not share any edges, there are no edges inside a supernode. Then, the resulting multigraph has no self-loops, but any pair of supernodes can have multiple edges between them. Observe that the max-degree can now increase to Δt, where Δ is the max-degree of the original graph. Thus, if we have an $S(n, \Delta)$-space $f(\Delta)$-coloring algorithm for multigraphs, then we can turn it into an $S(n/t, \Delta t)$-space $f(\Delta t)$-coloring algorithm. In particular, our $\widetilde{O}(n\sqrt{\Delta})$-space $\widetilde{O}(\Delta)$-coloring algorithm from Theorem 3 generalizes to an $\widetilde{O}(n\sqrt{\Delta/t})$-space $\widetilde{O}(\Delta t)$-coloring for any $1 \leq t \leq \Delta$. We generalize most of our algorithms to multigraphs and establish such a tradeoff.

3.2 Randomized Online Algorithms

Randomized online algorithm for vertex arrivals

Recall the simple greedy algorithm for online $(2\Delta - 1)$-coloring: we assign each incoming edge a color that is not already taken up by any of its adjacent edges. However, even in the one-sided vertex arrival model, to naively implement this algorithm, we need to remember the colors assigned to edges incident on *each* vertex on the "fixed" side, and hence, essentially colors assigned to all previous edges. This needs $O(n\Delta)$ space, and hence, the greedy algorithm doesn't seem to help in getting low-memory algorithms.

Algorithm 1 Randomized algorithm for 5Δ-coloring under one sided vertex arrivals.

Input: Stream of vertex arrivals of n-vertex graph $G = (A \sqcup B, E)$

Initialize:

Let $C = 5\Delta$.
for $v \in B$ **do**
    Let σ_v be a uniformly random permutation over $[C]$ ▷ *constructed on demand from random oracle bits*
    $h_v \leftarrow 1$. ▷ *counter for vertex v*

Process(vertex $x \in A$ with multiset M_x of edges)

Let $S_x \leftarrow \emptyset$ ▷ *set of colors M_x will have used so far*
for $e = \{x, y\}$ in M_x, in arbitrary order **do**
    ▷ *Increase counter for y to next value with color $\sigma_y[h_y]$ not already used, if possible*
    while $(h_y \leq C) \wedge (\sigma_y[h_y] \in S_x)$ **do**
        $h_y \leftarrow h_y + 1$ ▷ *increment counter for y*
    if $h_y > C$ **then** ▷ *counter exceeded 5Δ*
        abort
    Assign color $\sigma_y[h_y]$ to e ▷ *if this line is reached, current color must be unused; assign to e*
    $S_x \leftarrow S_x \cup \{\sigma_y[h_y]\}$ ▷ *add assigned color to set of used colors*
    $h_y \leftarrow h_y + 1$ ▷ *increment counter for y*

We observe, however, that for the vertex-arriving side, it is enough to remember only the colors assigned to edges on the "current" vertex so as to ensure no conflict among these edges. We shoot for a semi-streaming, i.e., $\widetilde{O}(n)$ space algorithm, and hence can afford to store the entire edge set of the current vertex with the assigned colors. To ensure that there is no color-conflict on the fixed side, we resort to random permutations. On each such vertex v, we have a random permutation σ_v of $[5\Delta]$ and a counter h_v. When an edge $\{a, b\}$ arrives with b on the fixed side, we look at the color at the h_bth index of σ_b. If that color is already taken by any edge incident on a (whose colors we explicitly store), then we increment the counter h_b. We continue this until we find an available color and increment the counter. The random permutations ensure (i) no color will be repeated on any fixed vertex (since a permutation takes distinct counter values to distinct colors) and (ii) with high probability, none of the counters can exceed 5Δ.[6] Intuitively, the slack in the number of colors ensure that for a single edge, an available color is reached within a constant counter increment in expectation. Hence, the Δ edges incident on a vertex can increase its counter to at most $O(\Delta)$ in expectation. Since we only store a counter for each vertex whose value can go up to $O(\Delta)$, the space usage is $\widetilde{O}(n)$. Thanks to the reductions discussed above, we can extend this to a semi-streaming[7] $O(\Delta)$-coloring for the general vertex arrival case, even for general graphs.

Randomized algorithm for online edge arrivals

When handling edge arrivals for simple graphs, our goal on receiving an edge $\{u, v\}$ is to assign a color that has not been used by any edge incident on u or on v. As precisely tracking the set of available colors for any given vertex requires $\Omega(\Delta)$ space, we will instead keep track of a subset of the available colors for a vertex, and choose colors for edges in a way that limits the amount of information we must store.

Algorithm 2 Storing free regions from a permutation.

$F \leftarrow$InitFreeTracker(C,s,Δ,σ)**:** ▷ *Assume C,s,Δ are powers of two, $C \geq \Delta$, $C \geq s$, and $\sigma \in S_C$*

$H \leftarrow [s]$ be a subset of $[s]$

$b \leftarrow 1$ be a counter with range from 1 to C/s

Interpreting F as subset of $[C]$

return $\sigma[H + (b-1)s]$

F**.RemoveAndUpdate**(c) ▷ *This will only be called with $c \in F$*

$H \leftarrow H \setminus \{\sigma^{-1}(c)\}$

if $|H| \leq s - s\Delta/C$ **then** ▷ *Switch to next block*

    $H \leftarrow [s]$

    $b \leftarrow b + 1$

Specifically, for each vertex v in the graph, we associate a uniform and independently chosen random permutation σ_v over the set $[C]$ of colors, where $C = O(\Delta)$ is the number of colors used. When edges incident on v arrive, we will choose colors by very roughly increasing

[6] This algorithm essentially discards colors that it skips over. We suspect one can obtain a semi-streaming $(2\Delta - 1)$-edge-coloring, or better, by retaining and promptly using the skipped colors in some fashion.

[7] Not counting random oracle bits. To implement without oracle randomness, choose each permutation independently from an almost $O(\log n)$-wise independent distribution over permutations.

order in σ_v. Specifically, we split σ_v up into a sequence of disjoint blocks $P_{v,1}, \ldots, P_{v,C/s}$ of size $s = O(\sqrt{\Delta \log n})$ each. At any given time, the algorithm have an integer b_v so that, all colors in blocks from $P_{v,1}$ up to P_{v,b_v-1} are assumed unsafe to use; none of the colors in $P_{v,b_v+1}, \ldots, P_{v,C/s}$ have been used; and the set F_v of colors in P_{v,b_v} that are still available is tracked exactly. Pseudocode for updating this state is given in Algorithm 2; it will ensure that $|F_v| \geq s(1 - \Delta/C)$. Whenever an edge $\{x, y\}$ arrives, the algorithm (Algorithm 3) will assign it a random color in $F_x \cap F_y$. We will prove that, since F_x and F_y have size close to s, and only a Δ/C fraction of (biased) random elements from P_{x,b_x} and P_{y,b_y} will have been used/excluded from F_x and F_y, the intersection of $F_x \cap F_y$ will have size $\Omega(s^2/C) = \Omega(\log n)$ with high probability. Thus, our algorithm will w.h.p. be able to pick a color for the edge $\{x, y\}$.

Algorithm 3 Randomized algorithm for $O(\Delta)$ edge coloring for simple graph edge arrival streams.

Input: Stream of vertex arrivals n-vertex graph $G = (V, E)$
Assume Δ is a power of two, and $\Delta = \Omega(\log(n/\delta))$
δ is the maximum failure probability over all input streams

Initialize:
Let $C = 128\Delta$ and $\epsilon = \frac{\delta}{16n^3}$
Let s be the least power of two which is $\geq 128\sqrt{\Delta \log(n/\delta)}$
for $v \in V$ **do**
    $\sigma_v \leftarrow$ permutation drawn from the (ϵ, s)-wise independent distribution over $\mathbb{S}_C$ from Lemma 14
    $F_v \leftarrow$ InitFreeTracker(C, s, Δ, σ_v)

Process(edge $\{x, y\}$) $\rightarrow$ **color**
if $F_x \cap F_y = \emptyset$ **then**
    abort
Let c be chosen uniformly at random from $F_x \cap F_y$.
F_x.RemoveAndUpdate(c)
F_y.RemoveAndUpdate(c)
return color c

Keeping track of the available colors for each vertex will use $\widetilde{O}(\sqrt{\Delta})$ space per vertex. We will choose the random permutations from an (ε, s)-wise independent distributions (see Definition 13). We show in a technical lemma that there is a particular such distribution whose permutations can be encoded using $\widetilde{O}(\sqrt{\Delta})$ space each, and for which sampling and permutation evaluation are efficient. In total over all vertices, we will use $\widetilde{O}(n\sqrt{\Delta})$ space for our algorithm.

3.3 Deterministic Online Algorithms

Derandomization of online vertex arrival

When a vertex a and its neighboring edges are processed, the randomized online vertex arrival algorithm is greedily finding an assignment for each edge incident on a to the next few available colors on the "fixed" vertex at the other endpoint. While this greedy selection works acceptably in the *average* case, it does not provide strong worst-case guarantees: with $\Omega(1/\operatorname{poly}(n))$ probability, *some* vertex adjacent to a will increase its counter by $\Omega(\log n)$,

thereby marking ("consuming") $\Omega(\log n)$ colors from its random permutation as unavailable. Consequently, there is small but nonzero risk that a vertex will use up too many colors from its random permutation and run out. In contrast, our deterministic algorithm for online edge coloring for one-sided vertex arrivals (Algorithm 4) is designed to ensure that, for any fixed vertex, the *amortized* number of colors consumed per edge incident on a fixed vertex is always bounded by a constant.

First, instead of greedily coloring the edges incident to a, we explicitly construct a matching in a bipartite graph between the set M_a of edges incident to a and the set of all colors, where each edge $\{a, b\}$ in M_a is linked to some of the colors that are known to be still available for vertex b. This avoids the problem of the greedy color selection, where there was always a small risk that for some vertex, all the next few colors in its random permutation were used; although it has the risk that the matching might not exist. Fortunately, as the number of color candidates per edge in M_a increases, the probability of there being no matching shrinks rapidly. If there are no repeated edges in M_a, and if each edge has t uniformly randomly chosen color candidates, the probability of there being no matching is $\exp(-\Omega(t|M_a|))$.

Second, instead of using random permutations, the deterministic algorithm uses a certain "good" array of permutations for which we are guaranteed that whenever we look for a matching, one will exist. This relies on an additional modification: instead of having each "fixed" vertex v keep track of a single counter h_v, we store a set of $\geq t = \Omega(\log n)$ colors known to still be available for vertex v, and periodically replace this set with a range of new colors from the permutation σ_v. The exact details of the encoding ensure that each fixed vertex has only $O(\operatorname{poly} n)$ possible states. Now, consider what happens when a vertex a (with incident edge set M_a) arrives, assuming for simplicity that M_a has no repeated edges. The number of possible configurations of states of vertices in $N(a)$ will be $\exp(|M_a| \log n)$. If each vertex had a random permutation, the probability of there being no matching would be $\exp(-\Omega(|M_a| \log n)$. By carefully adjusting parameters, we can ensure the product of these two is exponentially *small*. Then by a union bound we can show that the probability (over the random permutations) of *any* matching failing, for *any* set M_a and any associated vertex state, is at most $\frac{1}{2}$. In other words, for a good choice of permutations, our algorithm would *always* find a matching, in any state.

A notable problem is that storing each permutation would require $\widetilde{O}(\Delta)$ bits each.[8] To avoid this, we select the permutations from small, almost-k-wise independent families of permutations, instead. This works, but requires a more careful analysis to prove correct, and a large fraction of our proof is spent dealing with the interaction of this and support for multigraphs.

Derandomization of online edge arrival

We did not find a way to directly derandomize the randomized algorithm for online edge arrival. Instead, we created an algorithm (Algorithm 5) that manages to *partially* solve the edge coloring problem, only assigning a color to a $\geq 1/3$ fraction of the incoming edges, and leaving the rest uncolored. Now, say we run $O(\log \Delta)$ instances of this algorithm in parallel, each using a distinct palette of $O(\Delta)$ colors. When an edge arrives, we pass it to the first instance of the algorithm, and if it wasn't assigned a color, pass it to the second instance;

[8] One could recompute individual "good" permutations on demand, instead of storing all of the permutations, but this has the risk of the computation itself requiring $\widetilde{O}(n\Delta)$ bits of scratch space.

Algorithm 4 Deterministic $O(\Delta)$ edge coloring algorithm for one sided bipartite vertex arrivals.

Input: Stream of vertex arrivals for n-vertex graph $G = (A \sqcup B, E)$ of max degree Δ, where Δ is a power of 2

Initialize(δ):
Let $C = 2^{18}\Delta$.
Let $s = \lceil 2^{18} \log \frac{n\Delta}{\delta} \rceil$.
▷ *To get a deterministic algorithm, fix "good" values of these permutations*
▷ *When the permutations are chosen randomly, with probability* $\geq 1 - \delta$ *the algorithm will succeed on* all *inputs*
Let $(\sigma_v)_{v \in B}$ be permutations drawn from an $(\epsilon = C^{-s-1}, s)$-wise independent distribution over $\mathbb{S}_C$, using Lemma 14

for $v \in B$ **do**
  $b_v \leftarrow 1$.
  $Q_v \leftarrow [s]$.

Process(vertex x with multiset M_x of edges to B)
Let $d_{x,y}$ be the number of times edge $\{x, y\}$ is in M_x
for each $y \in B$ with $d_{x,y} > 0$ **do**
  if $d_{x,y} < \frac{1}{16}s$ **then**
    Let $F_y = (b_y - 1)s + Q_y$
  else
    Let $F_y = ((b_y - 1)s + Q_y) \sqcup \left[b_y s + 1, b_y s + \left\lceil \frac{64 d_{x,y}}{s} \right\rceil s\right]$
Construct bipartite graph H from M_x to $[C]$, edge $e \in M_x$ is linked to all $c \in \sigma_y[F_y]$.
Compute an M_x-saturating matching P of H.
for each $e \in M_x$ **do**
  Assign color $P(e)$ to e
  if $d_{x,y} < \frac{1}{16}s$ **then**
    Remove $\sigma_y^{-1}(P(e)) - (b_y - 1)s$ from Q_y
for each $y \in B$ with $d_{x,y} > 0$ **do**
  if $d_{x,y} < \frac{1}{16}s$ **then**
    if $|Q_y| \leq s - \frac{1}{2^{17}}s$ **then**
      $b_y \leftarrow b_y + 1$
      $Q_y \leftarrow [s]$
  else
    $Q_y \leftarrow [s]$
    $b_y \leftarrow b_y + \left\lceil \frac{64 d_{x,y}}{s} \right\rceil + 1$

and if that didn't assign a color, pass it to the third instance, and so on. All in all, only a $\leq 1/\operatorname{poly}(\Delta)$ fraction of the input stream will fail to be colored by this process, and there are few enough of these leftover edges that one can store them all and color them using $O(\Delta)$ colors. This approach uses $O(\Delta \log \Delta)$ colors in total.

The partial edge coloring algorithm itself uses an interesting trick. Let C be the number of colors allowed. Each vertex v has an associated permutation $\sigma_v \in S_C$, which is partitioned into a number of blocks $P_{v,1}, P_{v,2}, \ldots$ of $\widetilde{O}(\sqrt{\Delta})$ colors each. Whenever an edge $\{u, v\}$ arrives, it must be colored using a color in the set $P_{v,i} \cap P_{u,j}$, where i and j depend on the degrees of vertices v and u at the time. Parameters are set up so this set has size $\Omega(\log n)$, and the

Algorithm 5 (1/3)-partial $O(\Delta)$ edge coloring algorithm for graph edge arrival streams.

Input: Stream of edge arrivals in n-vertex graph $G = (V, E)$ of max degree Δ, where Δ is a power of two

Initialize(Δ,C,s):
▷ *C: number of colors used; s: block size parameter*
▷ *As with Algorithm 4, a "good" set of permutations will exist, if C/Δ is large*
$(\sigma_v)_{v\in V}$ are specific s-wise almost independent permutations
1: **for** $v \in V$ **do**
2: $F_v \leftarrow$ InitFreeTracker(C, s, Δ, σ_v). ▷ *Defined in: Algorithm 2*

Process(edge $\{x, y\}$) $\rightarrow$ **Option<color>** $\in \{\perp\} \cup [C]$
3: **if** $F_x \cap F_y \neq \emptyset$ **then**
4: Choose $c \in F_x \cap F_y$ arbitrarily
5: F_x.RemoveAndUpdate(c, $\{x, y\}$)
6: F_y.RemoveAndUpdate(c, $\{x, y\}$)
7: **return** color c
8: **else**
9: **return** $\perp$

algorithm knows which colors in $P_{v,i}$ and $P_{u,j}$ have been used so far. We prove that, if the algorithm could preview the future of the stream, it could always pick the "right" color in $P_{v,i} \cap P_{u,j}$, and thereby find a valid edge coloring. On the other hand, without being able to look at future edges, if one just greedily picks valid colors in $P_{v,i} \cap P_{u,j}$ that don't conflict with colors chosen earlier – assuming there are any – then the algorithm will color at least a 1/3 fraction of the edges.

Handling multigraphs in online edge arrival

Let us now turn to multigraphs. Note that the degree of a vertex takes into consideration the multiplicity of each incident edge. Both the randomized online edge arrival algorithm and what was described so far of the deterministic algorithm will fail when faced with input streams that repeat edges. But looking more closely, both can tolerate some amount of repeated edges – a given edge e could be repeated up to $\widetilde{O}(\sqrt{\Delta})$ times, as long as it doesn't arrive too frequently. Specifically, as long as, for a given endpoint x of e, the substream of edges incident on x does not contain e more than $\widetilde{O}(1)$ times in any interval of $\widetilde{O}(\sqrt{\Delta})$ edges. This is a consequence of the way the edge arrival algorithms rotate between blocks of colors for each vertex.

On the other hand, the edges that would break the sketch, which repeat many times within the last $\widetilde{O}(\sqrt{\Delta})$ edges incident on a given vertex, are easy to detect using $\widetilde{O}(n\sqrt{\Delta})$ space. All one must do is keep track of the edges which recently arrived at a vertex, and detect duplicates.

To handle the "bad" type of repeated edges, we maintain $O(\log \Delta)$ modified instances of an edge arrival algorithm. (In this paper, we use the basic deterministic online edge arrival algorithm, but the same argument would work for the randomized one.) The first instance processes all edges in the stream, filtering out the edges which it detects to have repeated at least once. The repeated edges are sent to the second instance, which filters out edges that it finds to have repeated twice, and sends those to the third instance. In general, the

ith instance will receive edges which have been seen $\Theta(2^i)$ times in the stream. (The exact condition is a bit more complicated.) The ith instance is also modified to handle edges with high repetition rates, assigning batches of colors to each edge type that it processes.

3.4 A Lower Bound

Space lower bound for online deterministic edge coloring

Our last notable result is a lower bound on the space needed for deterministic online edge coloring algorithms, which use $\beta\Delta$ colors, for a constant $\beta < 2$. It applies in the one sided bipartite vertex arrival setting, and thus automatically gives a lower bound for general vertex and edge arrivals. Let B be the "fixed" set of vertices, and A the "arriving" set of vertices. We prove the lower bound by reducing a deterministic, Δ-player, one way, communication game to (deterministic, one-sided, bipartite, vertex arrival) online edge coloring.

In this game, each player receives a set of edges to color, and must immediately output a coloring for the edges, before sending a message to the next player. Say that there are only $2^{o(n)}$ possible messages. Each message corresponds to a collection of inputs which the player could have received. One can show that each message "rules out" some of the colors for each vertex v in B, so that, if the protocol is correct, future players cannot mark edges going to v with one of the ruled out colors. Furthermore, there must be some message which has a large number of associated inputs, and which rules out $> \beta$ colors for each fixed vertex, on average. As this can happen for each of the Δ players, by the end of the protocol it is possible to have ruled out $> \beta\Delta$ colors per vertex, on average, which contradicts the assumption that the algorithm uses at most $\beta\Delta$ colors. Thus, there must be $2^{\Omega(n)}$ possible messages.

References

1. Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller cuts, higher lower bounds. *CoRR*, abs/1901.01630, 2019. doi:10.48550/arXiv.1901.01630.
2. Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu. Switch scheduling via randomized edge coloring. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2003,*, pages 502–512. IEEE, 2003. doi:10.1109/SFCS.2003.1238223.
3. Noga Alon and Sepehr Assadi. Palette sparsification beyond (Δ+1) vertex coloring. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 6:1–6:22, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.6.
4. Noga Alon and Shachar Lovett. Almost k-wise vs. k-wise independent permutations, and uniformity for general group actions. In *Proc. 16th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 350–361. Springer, 2012. doi:10.1007/978-3-642-32512-0_30.
5. Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh. Simple Streaming Algorithms for Edge Coloring. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:4, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.8.
6. Sepehr Assadi, Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Coloring in graph streams via deterministic and adversarially robust algorithms. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 141–153. ACM, 2023. doi:10.1145/3584372.3588681.
7. Sepehr Assadi, Andrew Chen, and Glenn Sun. Deterministic graph coloring in the streaming model. In *Proc. 54th Annual ACM Symposium on the Theory of Computing*, pages 261–274, 2022. doi:10.1145/3519935.3520016.

8 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for (Δ+ 1) vertex coloring. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 767–786, 2019. doi:10.1137/1.9781611975482.48.

9 Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: a single-pass semi-streaming algorithm for Δ-coloring. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 234–247. ACM, 2022. doi:10.1145/3519935.3520005.

10 Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. Online graph edge-coloring in the random-order arrival model. *Theory of Computing*, 8(1):567–595, 2012. doi:10.4086/toc.2012.v008a025.

11 Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, 44(5):251–253, 1992. doi:10.1016/0020-0190(92)90209-E.

12 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and massively parallel algorithms for edge coloring. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 15:1–15:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ESA.2019.15.

13 Soheil Behnezhad and Mohammad Saneian. Streaming edge coloring with asymptotically optimal colors. *arXiv preprint arXiv:2305.01714*, 2023. doi:10.48550/arXiv.2305.01714.

14 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proc. 39th ACM Symposium on Principles of Database Systems*, pages 63–80, 2020. doi:10.1145/3375395.3387658.

15 Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 11:1–11:21, 2020. doi:10.4230/LIPIcs.ICALP.2020.11.

16 Suman Kalyan Bera and Prantar Ghosh. Coloring in graph streams. *CoRR*, abs/1807.07640, 2018. doi:10.48550/arXiv.1807.07640.

17 Anup Bhattacharya, Arijit Bishnu, Gopinath Mishra, and Anannya Upasana. Even the easiest(?) graph coloring problem is not easy in streaming! In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 15:1–15:19, 2021. doi:10.4230/LIPIcs.ITCS.2021.15.

18 Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2830–2842. SIAM, 2021. doi:10.1137/1.9781611976465.168.

19 Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Simple and asymptotically optimal online bipartite edge coloring. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 331–336, 2024. doi:10.1137/1.9781611977936.30.

20 Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Adversarially robust coloring for graph streams. In *Proc. 13th Conference on Innovations in Theoretical Computer Science*, pages 37:1–37:23, 2022. doi:10.4230/LIPIcs.ITCS.2022.37.

21 Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the W-streaming model. In *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 181–183. SIAM, 2021. doi:10.1137/1.9781611976496.20.

22 Shiri Chechik, Doron Mukhtar, and Tianyi Zhang. Streaming edge coloring with subquadratic palette size. *arXiv preprint arXiv:2305.07090*, 2023. doi:10.48550/arXiv.2305.07090.

23 Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1–25. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00010.

24 Camil Demetrescu, Bruno Escoffier, Gabriel Moruz, and Andrea Ribichini. Adapting parallel algorithms to the W-stream model, with applications to graph problems. *Theoretical Computer Science*, 411(44):3994–4004, 2010. doi:10.1016/j.tcs.2010.08.030.

25 Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 714–723. ACM Press, 2006. doi:10.1145/1644015.1644021.

26 Martin R. Ehmsen, Lene M. Favrholdt, Jens S. Kohrt, and Rodica Mihai. Comparing first-fit and next-fit for online edge coloring. *Theor. Comput. Sci.*, 411(16-18):1734–1741, 2010. doi:10.1016/j.tcs.2010.01.015.

27 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.

28 Lene M. Favrholdt and Jesper W. Mikkelsen. Online edge coloring of paths and trees with a fixed number of colors. *Acta Informatica*, 55(1):57–80, 2018. doi:10.1007/s00236-016-0283-0.

29 Lene M. Favrholdt and Morten N. Nielsen. On-line edge-coloring with a fixed number of colors. *Algorithmica*, 35(2):176–191, 2003. doi:10.1007/s00453-002-0992-3.

30 S. Gandham, M. Dawande, and R. Prakash. Link scheduling in sensor networks: distributed edge coloring revisited. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 4, pages 2492–2501 vol. 4, 2005. doi:10.1109/INFCOM.2005.1498534.

31 Christian Glazik, Jan Schiemann, and Anand Srivastav. A one pass streaming algorithm for finding euler tours. *Theory of Computing Systems*, 67(4):1–23, December 2022. doi:10.1007/s00224-022-10077-w.

32 Magnus M. Halldorsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonayan. Near-optimal distributed degree+1 coloring. In *Proc. 54th Annual ACM Symposium on the Theory of Computing*, pages 450–463, 2022. doi:10.1145/3519935.3520023.

33 Ian Holyer. The np-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981. doi:10.1137/0210055.

34 Tiago Januario, Sebastián Urrutia, Celso C. Ribeiro, and Dominique de. Werra. Edge coloring: A natural model for sports scheduling. *European Journal of Operational Research*, 254(1):1–8, 2016. doi:10.1016/j.ejor.2016.03.038.

35 Janardhan Kulkarni, Yang P. Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 104–116. ACM, 2022. doi:10.1145/3519935.3519986.

36 Luigi Laura and Federico Santaroni. Computing strongly connected components in the streaming model. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 193–205. Springer Berlin Heidelberg, 2011. doi:10.1007/978-3-642-19754-3_20.

37 Jesper W. Mikkelsen. Optimal online edge coloring of planar graphs with advice. In *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, volume 9079 of *Lecture Notes in Computer Science*, pages 352–364. Springer, 2015. doi:10.1007/978-3-319-18173-8_26.

38 Jesper W. Mikkelsen. Randomization can be as helpful as a glimpse of the future in online computation. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 39:1–39:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.39.

39 Jayadev Misra and David Gries. A constructive proof of Vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992. doi:10.1016/0020-0190(92)90041-S.

40 Ben Morris. Improved mixing time bounds for the thorp shuffle. *Combinatorics, Probability and Computing*, 22(1):118–132, 2013. doi:10.1017/S0963548312000478.

41 Joseph Naor, Aravind Srinivasan, and David Wajc. Online dependent rounding schemes. *CoRR*, abs/2301.08680, 2023. doi:10.48550/arXiv.2301.08680.

42 Prabhakar Raghavan and Eli Upfal. Efficient routing in all-optical networks. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 134–143, 1994. doi:10.1145/195058.195119.

43 Amin Saberi and David Wajc. The greedy algorithm is not optimal for on-line edge coloring. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 109:1–109:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.109.

44 Claude E. Shannon. A theorem on coloring the lines of a network. *Journal of Mathematics and Physics*, 28(1-4):148–152, 1949. doi:10.1002/sapm1949281148.

45 Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. doi:10.1109/18.556667.

46 V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964.