Coloring in Graph Streams via Deterministic and Adversarially Robust Algorithms[*]

Sepehr Assadi[†]
sepehr.assadi@rutgers.edu
Rutgers University
Piscataway, New Jersey, USA

Amit Chakrabarti
amit.chakrabarti@dartmouth.edu
Dartmouth College
Hanover, New Hampshire, USA

Prantar Ghosh[‡]
prantar.ghosh@gmail.com
DIMACS, Rutgers University
Piscataway, New Jersey, USA

Manuel Stoeckl
manuel.stoeckl.gr@dartmouth.edu
Dartmouth College
Hanover, New Hampshire, USA

ABSTRACT

Graph coloring is a fundamental problem with wide reaching applications in various areas including data mining and databases, e.g., in parallel query optimization. In recent years, there has been a growing interest in solving various graph coloring problems in the streaming model. The initial algorithms in this line of work are all crucially randomized, raising natural questions about how important a role randomization plays in streaming graph coloring. A couple of very recent works prove that deterministic or even adversarially robust coloring algorithms (that work on streams whose updates may depend on the algorithm's past outputs) are considerably weaker than standard randomized ones. However, there is still a significant gap between the upper and lower bounds for the number of colors needed (as a function of the maximum degree Δ) for robust coloring and multipass deterministic coloring. We contribute to this line of work by proving the following results.

- In the deterministic semi-streaming (i.e., $O(n \cdot \text{polylog } n)$ space) regime, we present an algorithm that achieves a combinatorially optimal $(\Delta + 1)$-coloring using $O(\log \Delta \log \log \Delta)$ passes. This improves upon the prior $O(\Delta)$-coloring algorithm of Assadi, Chen, and Sun (STOC 2022) at the cost of only an $O(\log \log \Delta)$ factor in the number of passes.
- In the adversarially robust semi-streaming regime, we design an $O(\Delta^{5/2})$-coloring algorithm that improves upon the previously best $O(\Delta^3)$-coloring algorithm of Chakrabarti, Ghosh, and Stoeckl (ITCS 2022). Further, we obtain a smooth colors/space tradeoff that improves upon another algorithm of the said work: whereas their algorithm uses $O(\Delta^2)$ colors and $O(n\Delta^{1/2})$ space, ours, in particular, achieves (i) $O(\Delta^2)$ colors in $O(n\Delta^{1/3})$ space, and (ii) $O(\Delta^{7/4})$ colors in $O(n\Delta^{1/2})$ space.

CCS CONCEPTS

• Mathematics of computing → Graph coloring; **• Theory of computation → Streaming models**; **Sketching and sampling**; *Communication complexity.*

KEYWORDS

data streams, adversarial robustness, graph coloring

ACM Reference Format:
Sepehr Assadi, Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. 2023. 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 '23), June 18–23, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3584372.3588681

1 INTRODUCTION

In the graph coloring problem, we are given an undirected graph and the goal is to assign colors to the nodes of the graph such that adjacent nodes receive different colors. Graph coloring is a fundamental problem in graph theory with numerous applications in computer science, including in databases, data mining, register allocation, and scheduling [17, 28, 29]; see, e.g., the application to parallel query optimization by Hasan and Motwani [24] in VLDB'95. The emergence of massive graphs in many of these application domains has necessitated the study of graph coloring algorithms that are capable of handling such graphs efficiently on modern architecture. Of particular interest is the family of *graph streaming* algorithms: such an algorithm computes its solution using only a small number of sequential passes over the edges of the input graph, while using a sublinear amount of memory.

Several graph coloring problems have been studied in the streaming setting, typically with the goal of achieving a palette size (total number of colors used) proportional to the graph's chromatic number [1, 21], maximum vertex-degree [2–5, 12], arboricity [12], or degeneracy [11]. Also studied is the closely-related problem of (degree+1)-list-coloring [23] (see also [2]). Furthermore, graph coloring has been considered under different streaming paradigms such as random stream order and the vertex-arrival model [13].

[*]This work was supported in part by NSF under awards CCF-1907738 and CCF-2006589.
[†]Research supported in part by a NSF CAREER Grant CCF-2047061, a Google Research gift, and a Fulcrum award from Rutgers Research Council.
[‡]Research supported in part by a grant (820931) to DIMACS from the Simons Foundation. Part of the work was done while the author was at Dartmouth College.

Most of these works consider the *semi-streaming* regime, where we are restricted to $O(n \cdot \text{polylog}\, n)$ space for processing an n-vertex graph. Since even just storing the output coloring can require $\Omega(n \log n)$ space, this is close to optimal for the problem. We study semi-streaming graph coloring, focusing on *the* most popular color parameter in this line of work, namely the maximum degree Δ of the graph: we call this "Δ-based coloring."

A trivial greedy algorithm achieves a $(\Delta + 1)$-coloring in the offline setting. However, obtaining this color bound in the *streaming* model is fairly challenging. A breakthrough work by Assadi, Chen, and Khanna [4] did achieve such a coloring in semi-streaming space. An aspect of this algorithm, shared with almost all subsequent streaming coloring algorithms, is that it is inherently randomized. This raises the natural question: to what extent is *randomization* necessary for Δ-based coloring? Indeed, a derandomized version can be advantageous in multiple scenarios, e.g., having low or zero error even when the algorithm is rerun a huge (maybe exponential) number of times, or for robustness against input streams generated based on the algorithm's past outputs or internal states.

Two recent works have addressed this question. On the one hand, Assadi, Chen, and Sun [3] ruled out non-trivial single-pass deterministic algorithms for Δ-based coloring: any such algorithm requires $\exp(\Delta^{\Omega(1)})$ colors for semi-streaming space (and $\Delta^{\Omega(1/\alpha)}$ colors for $O(n^{1+\alpha})$ space). They further showed that allowing *multiple* semi-streaming passes over the stream makes better tradeoffs possible: one can get an $O(\Delta^2)$-coloring in two passes, and an $O(\Delta)$-coloring in $O(\log \Delta)$ passes. On the other hand, Chakrabarti, Ghosh, and Stoeckl [18], considered a "middle ground" between deterministic and randomized algorithms, namely the *adversarially robust* algorithms introduced by [9], recipient of the PODS'20 best paper award. These algorithms are required to work even when stream updates are generated by an adaptive adversary, depending on the algorithm's previous outputs (and thus implicitly on its internal randomness; observe that deterministic algorithms are always robust). They showed that a (possibly randomized) robust semi-streaming coloring algorithm requires $\Omega(\Delta^2)$ colors, while an $O(\Delta)$-coloring admits no $o(n^2)$-space robust algorithm. The same work also gave a robust semi-streaming algorithm achieving $O(\Delta^3)$ colors. Thus, the results in [3, 18] establish a neat trichotomy for single-pass semi-streaming graph coloring: (i) a $(\Delta + 1)$-color palette suffices for standard randomized streaming; (ii) poly(Δ) colors are necessary and sufficient for robust streaming; and (iii) $\exp(\Delta)$ colors are needed for deterministic algorithms.

Many questions in this line of work, however, remain unresolved. Here are two particular ones:

(i) *For deterministic algorithms, how many passes are needed to achieve a tight* $(\Delta + 1)$*-coloring?*
(ii) *For robust algorithms, where in the range* $[\Delta^2, \Delta^3]$ *does the above* "poly(Δ)" *number of colors lie?*

This paper takes steps towards resolving both these questions.

1.1 Our Contributions

The Deterministic Setting. In this setting, our main result is a multi-pass semi-streaming algorithm that achieves a $(\Delta+1)$-coloring, thus signicantly improving the $O(\Delta)$-color bound of [3] to the combinatorially optimal $\Delta + 1$, while paying only an additional $O(\log \log \Delta)$ factor in the number of passes.

Theorem 1.1. *There is a deterministic* $O(\log \Delta \log \log \Delta)$*-pass semi-streaming algorithm to* $(\Delta + 1)$*-color a graph with maximum degree* Δ*, where the graph edges arrive in any adversarial order.*[1]

To fully appreciate this improvement, note that in streaming, as well as several other computational models, it is known that $O(\Delta)$-coloring is an "algorithmically *much* easier" problem than $(\Delta + 1)$-coloring. For instance, there are quite simple single-pass randomized algorithms known for $O(\Delta)$-coloring [4, 12], whereas the only known streaming $(\Delta + 1)$-coloring algorithm, due to [4], uses sophisticated tools and a combinatorially involved analysis.[2]

Our algorithm in Theorem 1.1 uses a variety of novel ideas and techniques. It is inspired by a recent distributed algorithm of Ghaffari and Kuhn [22] that solves $(\Delta + 1)$-coloring in the CONGEST model of distributed computation; which in turn was inspired by earlier algorithms of [7] and [27]. In building on these works, we must contend with the limitation that the semi-streaming model does not allow enough space for a typical vertex to "know" much of its neighborhood; this is in sharp contrast to distributed computing models (including CONGEST). The algorithm of [22] asks each vertex to progressively whittle down its space of candidate colors by using full information about its neighborhood. Since we cannot do this in a streaming setting, we make novel use of pairwise-independent hashing to collect the necessary information "approximately." Another innovation in our algorithm is the introduction of a stream-computable quantity we call "slack" (inspired by, but different from, a similar notion used in distributed coloring algorithms [23]) that measures roughly how many free colors each vertex has. The precise definition of slack is a key contribution of this work, since it is crucial for the eventual analysis of our algorithm. Moreover, our algorithm achieves roughly $O(\log \Delta)$ passes, whereas that of [22] uses $O(\log^2 \Delta \log n)$ distributed rounds; this quantitative difference stems, in part, from our delicate tuning of parameters in an iterative process that colors vertices in batches.

As a by-product of the technology developed for establishing Theorem 1.1, we also find an efficient streaming algorithm for the more general problem of (degree + 1)-list-coloring. In this problem, the input stream specifies a graph G as usual and, for each vertex x, a list L_x of at least $\deg(x) + 1$ allowed colors for x; the goal is to properly color G subject to these lists. We get the following result.

Theorem 1.2. *Let* C *be a set of colors of size* $O(n^2)$*. There is a deterministic semi-streaming algorithm for* (*degree* + 1)*-list-coloring a graph* G *given arbitrarily interleaved stream of the graph edges and* (x, L_x) *pairs specifying the list* L_x *of allowed colors for a vertex* x*, where* $L_x \subseteq C$ *with* $|L_x| \geq \deg(x)+1$*. The algorithm uses* $O(n \log^2 n)$ *bits of space and runs in* $O(\log \Delta \log \log \Delta)$ *passes.*

[1] If Δ is not known in advance, it can be determined in $O(n \log n)$ space using a single pass.
[2] Similar examples of this difference appear in the (randomized) LOCAL algorithms [19, 30], (deterministic) dynamic graph algorithms [14], or even provable separations for the "palette sparsification" technique [2, 4]. Yet another example is the closely related problem of O(degeneracy)-coloring versus (degeneracy + 1)-coloring studied by [11] who proved that the former admits a (randomized) single-pass semi-streaming algorithm while the latter does not.

The Adversarially Robust Setting. In this setting, our algorithm needs to be correct against an adversary who constructs the input graph adaptively by inserting edges based on the past colorings returned by the algorithm. This is inherently a single-pass setting. However, we are now allowed to use randomness. Observe that the stream elements might depend on past outputs, which in turn depend on the random bits used by the algorithm. While [4] gave a semi-streaming $(\Delta + 1)$-coloring algorithm in the "non-robust" setting where the stream is fixed in advance, [18] showed that a *robust* semi-streaming algorithm must use $\Omega(\Delta^2)$ colors. Our main result in the robust setting is the following.

Theorem 1.3. *There is an adversarially robust semi-streaming algorithm to $O(\Delta^{5/2})$-color a graph, given oracle access to $O(n\Delta)$ bits of randomness.*

The above result improves a robust algorithm of [18], which runs in a similar semi-streaming amount of space and accesses as many random bits, but only gives an $O(\Delta^3)$-coloring. Further, our robust algorithm admits a smooth tradeoff between the number of colors and the memory used: we can get an $O(\Delta^{5/2-3\beta/2})$-coloring in $\widetilde{O}(n\Delta^{\beta})$ space[3] for any $\beta \in [0, 1]$. While [18] gave a robust algorithm giving an $O(\Delta^2)$-coloring using $\widetilde{O}(n\Delta^{1/2})$ space, our result implies robust $O(\Delta^2)$-coloring is possible with $\widetilde{O}(n\Delta^{1/3})$ space, and $O(\Delta^{7/4})$-coloring with $\widetilde{O}(n\Delta^{1/2})$ space.

Our algorithm overcomes the challenges posed by the adaptive adversary by crucially exploiting the graph structure and using enhanced versions of some known techniques on subgraphs of the input graph. These techniques include those in the adversarially robust literature, such as *sketch switching* [9, 18], as well as those in the coloring literature, such *graph partitioning* [12] and *degeneracy-based coloring* [11].

One caveat of the above result is the need for a large number of random bits. The same caveat applies to the aforementioned robust $O(\Delta^3)$-coloring algorithm of [18]. We note three points in defense of this "lenient" model, where random bits are "free" and do not count towards space usage. First, the lower bound of $\Omega(\Delta^2)$ colors for robust semi-streaming algorithms does apply even in this lenient model. Second, in practice, one might reduce the randomness usage via a cryptographic pseudorandom generator. But perhaps the best response is our next result, wherein we design a fresh algorithm that removes this caveat while achieving the same color bound as [18]. Note that this is the first non-trivial robust coloring algorithm in the strict semi-streaming model that accounts for all random bits in space usage.

Theorem 1.4. *There is an adversarially robust $O(\Delta^3)$-coloring algorithm that runs in semi-streaming space, even when one charges for the random bits used by the algorithm.*

1.2 Related work

Prior works most relevant to our paper are [3], [18], and [22]. We have already discussed their results and comparisons with our work in the sections above. Other important related work include the growing literature on streaming graph coloring [1, 2, 4, 5, 11–13, 23]. However, all these works study the problem in the "static" streaming model. Starting with [9], there has been a rapidly growing interest in adversarially robust streaming, leading to a long line of papers on the topic in the last couple of years [6, 8, 10, 15, 18, 20, 25, 26, 31, 32]. To the best of our knowledge, [18] is the only prior work to study graph coloring in the robust model. For a detailed account on these works, please see Appendix A.

2 PRELIMINARIES

Notation. Throughout the paper, "log" denotes the base-2 logarithm; $[n]$ denotes the set $\{1, \ldots, n\}$; $\mathbb{F}_p$ is the finite field with p elements; $\mathbb{1}_{\mathsf{cond}}$ is the indicator function for condition cond, i.e., it takes the value 1 when cond is true, and 0 otherwise; and the notation $a \in_R A$ means that a is drawn uniformly at random from the finite set A.

A graph $G = (V, E)$ typically has $n = |V|$ vertices. We may identify G with its set E of edges, and write $\{u, v\} \in G$ to mean that $\{u, v\} \in E$. For $B \subseteq E$, $\deg_B(x)$ denotes the degree of x in the graph formed by the edges in B. For $X \subseteq V$, $G[X]$ denotes the subgraph of G induced by X.

The Classical Streaming Model. In the *static* or *classical* streaming setting, an algorithm operates on a long sequence $\langle e_1, e_2, \ldots \rangle$ of elements, fixed in advance. It may make multiple passes over the stream. For a given parameter δ, we typically aim to design a streaming algorithm with parameter S as low as possible so that, for all possible input streams, it uses $\leq S$ bits of space and errs with probability $\leq \delta$. If the algorithm is deterministic, then $\delta = 0$.

The Adversarially Robust Streaming Model. In the *adversarial* setting, we view the algorithm as one party in a game it plays with an *adversary*; the adversary produces a sequence $\langle e_1, e_2, \ldots \rangle$ of elements, and can ask the algorithm to report an output o_i after each new element e_i. Unlike the static setting, the next element e_{i+1} produced by the adversary may depend (possibly randomly[4]) on the transcript $\langle e_1, o_1, \ldots, e_i, o_i \rangle$ of the game. The algorithm is said to err if at least one of its outputs is incorrect for the problem at hand. In this setting, we typically aim to find streaming algorithms minimizing S, δ, where the algorithm (a) never exceeds S bits of space and (b) errs with probability $\leq \delta$ for all possible adversaries.

Colorings. A *partial coloring* of a graph $G = (V, E)$ using a palette C (any nonempty finite set) is a tuple (U, χ) where $U \subseteq V$ is the set of uncolored vertices and $\chi \colon V \to C \cup \{\bot\}$ is a function such that $\chi(x) = \bot \Leftrightarrow x \in U$ (we may also simply refer to χ as the partial coloring). The coloring is said to be *proper* if, for all $\{u, v\} \in E$ such that $u \notin U$ and $v \notin U$, we have $\chi(u) \neq \chi(v)$. A proper coloring of G is a partial coloring where $U = \emptyset$.

Given a graph-theoretic parameter ψ, the ψ-coloring (algorithmic) problem asks one to determine a proper coloring of an input graph G using a palette of size $|C| \leq \psi$. This paper focuses first on $(\Delta+1)$-coloring and later on $\mathrm{poly}(\Delta)$-coloring. We also consider the *list-coloring* problem, wherein each $x \in V$ has an associated list (really a set) $L_x \subseteq C$ and we are to find a coloring satisfying $\chi(x) \in L_x$ for all x. Specifically, we study the problem of $(\deg + 1)$-list-coloring, in which $|L_x| = \deg(x) + 1$ for each x.

[3]The $\widetilde{O}(\cdot)$ notation suppresses polylogarithmic factors.

[4]However, by Yao's lemma, there is always a deterministic adversary at least as effective as any randomized one at making the algorithm fail.

Hash Functions. We will use the following standard properties of families of hash functions. A hash family $\mathcal{H}$ of functions $A \to B$ is *k-independent* if, for all distinct $a_1, \ldots, a_k \in A$, and arbitrary $b_1, \ldots, b_k \in B$, $\Pr_{h \in_R \mathcal{H}}\left[\bigwedge_{i=1}^{k} h(a_i) = b_i\right] = 1/|B|^k$. The family is 2-*universal* if, for all $a_1 \neq a_2 \in A$, $\Pr_{h \in_R \mathcal{H}}\left[h(a_1) = h(a_2)\right] \leq 1/|B|$.

Useful Lemmas. We use the following variations of standard lemmas, proofs of which are given in the full version of the paper.

Lemma 2.1 (Constructive variant of Turán's theorem). *Given a graph with n vertices and m edges, one can find an independent set of size $\geq n^2/(2m+n)$ in deterministic polynomial time.* □

Lemma 2.2 (Mix of Chernoff bound and Azuma's inequality). *Let $X_1, \ldots, X_k$ be a sequence of $\{0,1\}$ random variables, and $c \in [0,1]$ a real number for which, for all $i \in k$, $\mathbb{E}[X_i \mid X_1, \ldots, X_{i-1}] \leq c$. Then $\Pr\left[\sum_{i \in [k]} X_i \geq (1+t)kc\right] \leq 2^{-tkc}$, assuming $t \geq 3$.* □

3 A (MULTIPASS) DETERMINISTIC ALGORITHM

This section presents our first main result, giving a multipass deterministic semi-streaming algorithm for $(\Delta+1)$-coloring, proving Theorem 1.1. As usual, the input graph $G = (V, E)$ has $n = |V|$ vertices and maximum degree Δ. Later, we shall extend our algorithm to the $(\deg+1)$-list-coloring problem, so it will be helpful to think of each vertex $x \in V$ being associated with a set L_x of allowed colors; for our main algorithm, $L_x = [\Delta+1]$ for each $x \in V$.

3.1 High-Level Organization

The algorithm's passes are organized as follows. The algorithm proceeds in *epochs*, where each epoch starts with a partial coloring χ that has a certain subset $U \subseteq V$ uncolored and ends with a new partial coloring that extends χ by coloring at least a third of the vertices in U, thereby shrinking $|U|$ to $\leq \frac{2}{3}|U|$. In the beginning, $U = V$. After at most $\lceil \log_{3/2} \Delta \rceil$ such epochs, we will have $|U| \leq n/\Delta$: at this point, the algorithm makes a final pass to collect all edges incident to U and greedily extend χ to a full coloring of G.

Each epoch of the algorithm is divided into *stages*, where each stage whittles down a set of proposed colors for each uncolored vertex. The following definition is crucial to the logic of an epoch and is an important conceptual contribution of this work.

Definition 3.1 (Partial commitment, slack, potential). A *partially committed coloring* (PCC) of G is an assignment of colors and lists to the vertices satisfying the following conditions.

- Every vertex outside a subset $U \subseteq V$ of uncolored vertices is assigned a specific color $\chi(x) \in L_x$; the resulting χ is a proper partial coloring.
- Each $x \in U$ has an associated set P_x of proposed colors, defining a collection $\mathcal{P} = \{P_x\}_{x \in U}$.
- For all vertices $x, y \in U$, either $P_x = P_y$ or $P_x \cap P_y = \emptyset$.

We shall denote such a PCC by the tuple $(U, \chi, \mathcal{P})$. Given such a PCC, define the *slack* of a vertex with respect to a set T of colors by

$$\operatorname{slack}(x \mid T) = \max\{0,\ |T \cap L_x| - |\{y \in \mathrm{N}(x) \setminus U : \chi(y) \in T\}|\}, \tag{1}$$

and further define $s_x = \operatorname{slack}(x \mid P_x)$; that is, s_x is the number of colors in P_x that are available to x in L_x minus the number of *times* the colors in P_x have appeared in the already colored neighbors of x. Define the *potential* of the PCC to be

$$\Phi = \Phi(U, \chi, \mathcal{P}) = \sum_{\{x,y\} \in E} \mathbb{1}_{x \in U \wedge y \in U} \cdot \mathbb{1}_{P_x = P_y} \cdot \left(\frac{1}{s_x} + \frac{1}{s_y}\right) \tag{2}$$

which sums the quantity $(1/s_x + 1/s_y)$ over all edges $\{x, y\}$ inside U with $P_x = P_y$. □

Intuitively, the slack defined here is a lower bound on the number of unused colors available to a vertex. Our definition differs from the "slack" defined by [23], where the number of colors used by the neighbors is known exactly. It turns out that such a lower bound on the number of unused colors is sufficient for our algorithm to progressively refine a PCC. The advantage of this lower bound—equivalently, of the upper bound $|\{y \in \mathrm{N}(x) \setminus U : \chi(y) \in T\}|$ on the number of used colors, instead of the exact quantity $|T \cap \{\chi(y) : y \in \mathrm{N}(x) \setminus U\}|$—is that the former is a linear function of the data stream, and can be easily computed in $O(\log n)$ space by storing a single counter which is incremented each time an edge $\{x, y\}$ with $y \notin U$ and $\chi(y) \in T$ is encountered. Meanwhile, as a consequence of the set disjointness lower bound in communication complexity, determining the latter can require up to $\Omega(\Delta)$ space. In the LOCAL and CONGEST models, each vertex can easily store and maintain a list of all its available colors (equivalently, colors used by its neighborhood), so the algorithms of [7, 22, 23] do not need such a modified notion of "slack."

The set $\operatorname{Free}(T, x) := T \cap L_x \setminus \{\chi(y) : y \in \mathrm{N}(x) \setminus U\}$ is the set of all colors in T that are available for x, in light of the local constraints imposed by L_x and χ. Notice that $|\operatorname{Free}(T, x)| \geq \operatorname{slack}(x \mid T)$, since a color in T might be used more than once in the neighborhood of x, thus reducing the LHS only once, but the RHS more than once. Hence, if we extend χ to a full coloring by choosing, independently for each $x \in U$, a uniformly random color in $\operatorname{Free}(P_x, x)$, the only monochromatic edges we might create are within U and the number, $m_{\text{mono}}(U, \chi, \mathcal{P})$, of such edges satisfies

$$\begin{aligned}
\mathbb{E}\, m_{\text{mono}}(U, \chi, \mathcal{P}) &= \sum_{\substack{\{x,y\} \in E(G[U]) \\ P_x = P_y}} \frac{|\operatorname{Free}(P_x, x) \cap \operatorname{Free}(P_y, y)|}{|\operatorname{Free}(P_x, x)| \cdot |\operatorname{Free}(P_y, y)|} \\
&\leq \sum_{\substack{\{x,y\} \in E(G[U]) \\ P_x = P_y}} \left(\frac{1}{s_x} + \frac{1}{s_y}\right) = \Phi\,.
\end{aligned} \tag{3}$$

3.2 The Logic of an Epoch: Extending a Partial Coloring

Returning to the algorithm outline, at the start of an epoch, the current partial coloring χ and its corresponding set U of uncolored vertices define a trivial PCC where $P_x = L_x = [\Delta+1]$ for each x. We shall eventually show that the resulting potential $\Phi \leq |U|$. Each stage in the epoch shrinks these sets P_x in such a way that the potential Φ does not increase much. After several stages, each P_x in the PCC becomes a singleton and the bound on Φ, together with eq. (3), ensures that assigning each $x \in U$ the sole surviving color in P_x would not create too many monochromatic edges. Now, Lemma 2.1 allows us to commit to these proposed colors for at least $\frac{1}{3}|U|$ of the uncolored vertices; this defines a new partial coloring and ends the epoch.

We now describe how to shrink the sets P_x. For this, view each color as a b-bit vector where $b = \lceil \log(\Delta+1) \rceil$ according to some canonical mapping, e.g., $\mathbf{a} \in \{0,1\}^b \mapsto 1 + \sum_{i=1}^{b} a_i 2^{i-1}$. Each set P_x will correspond to a subcube of $\{0,1\}^b$ where the first several bits have been fixed to particular values.[5] Each stage of the rth epoch (except perhaps the last, due to divisibility issues) will shrink each P_x by fixing an additional k bits of its subcube, thus reducing the dimension of the subcube. We choose $k := 1 + \lfloor \log(n/|U|) \rfloor$, so that $|U|2^k \le 2n$; this bound will be important when we analyze the space complexity. The epoch ends when all bits of each P_x have been fixed, making each P_x a singleton; clearly, this happens after $\lceil b/k \rceil$ stages.

This brings us to the heart of the algorithm: we need to describe, for each $x \in U$ and the particular value of k for the current epoch, how to fix the next k bits for P_x. Let $P_{x,\mathbf{j}}$ be the subset of P_x where the k lowest-indexed free bits are set to $\mathbf{j} \in \{0,1\}^k$: this partitions P_x into 2^k subcubes. Define

$$w_{x,\mathbf{j}} = \frac{\text{slack}(x \mid P_{x,\mathbf{j}})}{\sum_{\mathbf{i} \in \{0,1\}^k} \text{slack}(x \mid P_{x,\mathbf{i}})} \,. \tag{4}$$

An easy calculation shows that if, for each x, we choose $\mathbf{j}$ at *random* according to the distribution given by $(w_{x,\mathbf{j}})_{\mathbf{j} \in \{0,1\}^k}$ to obtain a new random collection $\widetilde{\mathcal{P}}$ of proposed color sets for each vertex, then

$$\mathbb{E}\, \Phi(U, \chi, \widetilde{\mathcal{P}}) = \Phi(U, \chi, \mathcal{P}) \,. \tag{5}$$

Therefore, there *exists* a particular realization $\mathcal{P}'$ of $\widetilde{\mathcal{P}}$ such that $\Phi(U,\chi,\mathcal{P}') \le \Phi(U,\chi,\mathcal{P})$. However, it is not clear how to identify such a $\mathcal{P}'$ deterministically and space-efficiently in a stream.

A key idea that enables a space-efficient derandomization is to choose the $\mathbf{j}$ values for the vertices $x \in U$ in a pseudorandom fashion, using a 2-independent family $\mathcal{H}$ of hash functions $V \mapsto [p]$ for a value p which is $\Omega(n \log n)$. By using a map $g_{\mathbf{w}} \colon U \times [p] \to \{0,1\}^k$, as specified in the following lemma (proved in the full version), we can use a uniform random value in $[p]$ to sample from a distribution *close enough* to the $(w_{x,\mathbf{j}})$ distribution.

Lemma 3.2. *There is a function $g_{\mathbf{w}} \colon U \times [p] \to \{0,1\}^k$ satisfying*

$$\frac{|g_{\mathbf{w}}^{-1}(x,\mathbf{j})|}{p} \le w_{x,\mathbf{j}} \left(1 + \frac{1}{8 \log n}\right), \quad \forall\, \mathbf{j} \in \{0,1\}^k \,. \quad \square$$

Then, for each x, we shrink P_x to $P_{x,\mathbf{j}(x)}$ where $\mathbf{j}(x) = g_{\mathbf{w}}(x, h(x))$ and $h \in_R \mathcal{H}$. Let $\mathcal{P}_h$ denote the resulting collection of proposed color sets.

By choosing (e.g.) the Carter–Wegman family of affine functions on $\mathbb{F}_p$ for prime p, we can take $|\mathcal{H}| = O(n^2 \log^2 n)$. This lets us identify a specific function $h \in \mathcal{H}$ for which $\Phi(U,\chi,\mathcal{P}_h)$ is not much larger than the average over all $h \in \mathcal{H}$, using two streaming passes and $\widetilde{O}(n)$ space. We then show that the new potential is at most $1 + O(1/\log n)$ times the old. Repeating this argument for each of the $O(\log n)$ stages in the epoch shows that at the end of the epoch, the potential will have increased by at most a constant factor, which allows us to shrink U by a factor $2/3$.

The above outline suggests $O(\log n)$ epochs, each using $O(\log n)$ stages, each of which uses $O(1)$ passes. A more careful analysis bounds the number of passes by $O(\log \Delta \log\log \Delta)$.

[5] If $\Delta + 1$ is not a power of 2, P_x might contain elements not in L_x, but this doesn't matter because $\text{Free}(T,x) \subseteq L_x$ always.

3.3 Detailed Algorithm and Analysis

We now describe the algorithm more formally, by fleshing out the precise logic of an epoch. Let $Q^{(i)}$ denote the partition of the color space $\{0,1\}^b$ into subcubes $Q_{\mathbf{j}}^{(i)}$ defined by setting the ith k-bit block to each of the 2^k possible patterns $\mathbf{j}$; i.e.,

$$Q_{\mathbf{j}}^{(i)} := \left\{\mathbf{a} \in \{0,1\}^b : (a_{ki-k+1}, \ldots, a_{ki}) = \mathbf{j}\right\}$$
$$Q^{(i)} := \left\{Q_{\mathbf{j}}^{(i)}\right\}_{\mathbf{j} \in \{0,1\}^k} \,. \tag{6}$$

If k does not divide b, we must make an exception for the $\lceil b/k \rceil$th partition, for which the relevant bit patterns $\mathbf{j}$ would be shorter; for clarity of presentation, we shall ignore this edge case in what follows. The full logic of the algorithm is given in Algorithm 1.

Algorithm 1 Deterministic Semi-Streaming $(\Delta+1)$-Coloring

```
1:  procedure Deterministic-Coloring(streamed $n$-vertex graph $G = (V,E)$ with max degree $\Delta$)
2:      $U \leftarrow V$; $\chi(x) \leftarrow \perp$ for all $x \in V$        ▷ all vertices uncolored
3:      repeat
4:          Coloring-Epoch$(G, U, \chi)$ ▷ shrinks $|U|$ to at most $\frac{2}{3}|U|$
5:      until $|U| \le n/\Delta$
6:      In one pass, collect every edge incident to a vertex in $U$
7:      Use these edges to greedily complete $\chi$ to a proper coloring of $G$

8:  procedure Coloring-Epoch(graph $G$, partial coloring $(U,\chi)$)
9:      $b \leftarrow \lceil \log(\Delta+1) \rceil$        ▷ each color is a $b$-bit vector
10:     $k \leftarrow 1 + \lfloor \log(n/|U|) \rfloor$ ▷ number of bits fixed in each stage
11:     for each $x \in U$ do $P_x \leftarrow \{0,1\}^b$  ▷ the initial, trivial PCC
12:     for each stage $i$, from 1 through $\lceil b/k \rceil$ do
13:         $\mathcal{P} \leftarrow$ Refine-PCC$((U,\chi,\mathcal{P}), i, b, k)$

14:     end-of-epoch pass:        ▷ each $P_x$ is now a singleton
15:         Collect $F \leftarrow \{\{u,v\} \in E : u \in U, v \in U$, and $P_u = P_v\}$
            ▷ (We will prove that $|F| = O(|U|)$)
16:         In the graph $(V,F)$, find an independent set $I$ with $|I| \ge \frac{1}{3}|U|$, using Lemma 2.1
17:         for each $x \in I$ do        ▷ extend $\chi$ by coloring $x$
18:             $U \leftarrow U \setminus \{x\}$
19:             $\chi(x) \leftarrow$ the sole element in $P_x$
```

The most important aspect of the analysis is to quantify the progress made in each epoch and establish that the colors proposed at the end of each stage do not produce too many monochromatic edges (i.e., those in F.) This analysis will demonstrate the utility of the potential defined in eq. (2).

Given a PCC $(U,\chi,\mathcal{P})$ where $\mathcal{P} = \{P_x\}_{x \in U}$, define the "conflict degree" of each $x \in U$ by

$$d_{\text{conf}}(x) = d_{\text{conf}}(x; U, \chi, \mathcal{P}) := |\{y \in N(x) \cap U : P_y = P_x\}| \,, \tag{7}$$

which counts the neighbors of x that could potentially form monochromatic edges with x, were we to assign colors from $\mathcal{P}$ to the uncolored vertices. Recall the quantities $s_x = \text{slack}(x \mid P_x)$ from Definition 3.1.

Lemma 3.3. *For every PCC, $\Phi(U,\chi,\mathcal{P}) = \sum_{x \in U} d_{\text{conf}}(x)/s_x$.*

Algorithm 2 Refining a Partially Committed Coloring

```
1: procedure Refine-PCC(streamed $n$-vertex graph $G = (V, E)$ with max degree $\Delta$; PCC $(U, \chi, \mathcal{P})$; stage number $i$; parameters $b, k$)
2:     pass 1:
3:         for each $x \in U$ and $Q \in \mathcal{Q}^{(i)}$ do compute slack$(x \mid P_x \cap Q)$ by using eq. (1)
4:     Determine all $w_{x,\mathbf{j}}$ values using eq. (4), noting that $P_{x,\mathbf{j}} = P_x \cap Q^{(i)}_{\mathbf{j}}$
5:     Let $p \leftarrow$ a prime in $[8n \log n, 16n \log n]$
6:     Let $\mathcal{H} := \{z \mapsto az + b :\ a, b \in \mathbb{F}_p\}$ be a family of hash functions $U \mapsto [p]$    ▷ Carter–Wegman hashing
7:     Implicitly construct $g_{\mathbf{w}} : U \times [p] \to \{0,1\}^k$ as per Lemma 3.2
8:     Define $\mathcal{P}_h := (P_{x,h})_{x \in U}$ where $P_{x,h} := P_x \cap Q^{(i)}_{g_{\mathbf{w}}(x,h(x))}$
       ▷ Identify a specific $h^\star \in \mathcal{H}$ for which $\Phi(U, \chi, \mathcal{P}_{h^\star})$ is not much larger than average, as follows:
9:     pass 2:
10:        Split $\mathcal{H}$ into $\sqrt{|\mathcal{H}|}$ parts of equal size
11:        Estimate $\sum_h \Phi(U, \chi, \mathcal{P}_h)$ for each part, up to $(1 + 1/(8 \log n))$ relative error
12:        Pick the part minimizing the estimated sum
13:    pass 3:
14:        Estimate $\Phi(U, \chi, \mathcal{P}_h)$ for each $h$ within the chosen part, up to $(1 + 1/(8 \log n))$ relative error
15:        Choose $h^\star$ as the (approximate) minimizer
16:    $\mathcal{P} \leftarrow \mathcal{P}_{h^\star}$    ▷ constrain the PCC more tightly
```

Proof. From the definitions in eqs. (1) and (2), using some straightforward algebra,

$$\Phi(U, \chi, \mathcal{P}) = \sum_{\substack{\{u,v\} \in E(G[U]) \\ P_u = P_v}} \left(\frac{1}{s_u} + \frac{1}{s_v} \right) = \sum_{x \in U} \frac{|\{y \in U :\ \{x, y\} \in E \wedge P_x = P_y\}|}{s_x} = \sum_{x \in U} \frac{\mathrm{d_{conf}}(x)}{s_x} . \qquad \square$$

Lemma 3.4. *For all x and disjoint sets T_1, T_2: slack$(x \mid T_1 \cup T_2) \le$ slack$(x \mid T_1)$ + slack$(x \mid T_2)$.*

Proof. This follows from eq. (1) and the fact that $\max\{0, a_1 + a_2\} \le \max\{0, a_1\} + \max\{0, a_2\}$. $\square$

The following major lemma addresses the core of the algorithm; it is proven in Appendix B.

Lemma 3.5 (Restated as Lemma B.1). *Suppose we start a particular epoch with the partial coloring (U, χ) and the initial, trivial PCC $(U, \chi, \mathcal{P}_0)$. Suppose there are ℓ stages in this epoch and the ith stage begins with the PCC $\mathcal{P}_i$. Let $\Phi_i := \Phi(U, \chi, \mathcal{P}_i)$ be the corresponding potential, for $0 \le i \le \ell$. Then $\Phi_0 \le |U|$ and $\Phi_\ell \le 2|U|$.*

The crucial combinatorial property of the $(\Delta + 1)$-coloring problem is that given any proper partial coloring, every uncolored vertex is guaranteed to have a free color not in use by its colored neighbors. The next lemma argues that even as we gradually tighten constraints in our PCC during the stages of an epoch, a similar guarantee is maintained.

Lemma 3.6. *In each epoch, for all $x \in U$, the stages maintain the invariant that $s_x \ge 1$ and after the last stage we have $s_x = 1$.*

Proof. At the start of the epoch, $s_x \ge |L_x| - |\mathrm{N}(x)| = (\Delta + 1) - \deg(x) \ge 1$.

Consider a particular stage, which begins with a PCC $(U, \chi, \mathcal{P})$, where $\mathcal{P} = \{P_x\}_{x \in U}$. Fix a vertex $x \in U$. In the next PCC formed at the end of the stage, P_x shrinks down to $P_{x,\mathbf{j}} = P_x \cap Q^{(i)}_{\mathbf{j}}$ for a pattern $\mathbf{j} \in \{0,1\}^k$ satisfying $w_{x,\mathbf{j}} > 0$: the way $g_{\mathbf{w}}$ is defined (Lemma 3.2) ensures this. By Lemma 3.4,

$$\sum_{\mathbf{i} \in \{0,1\}^k} \mathrm{slack}(x \mid P_{x,\mathbf{i}}) \ge \mathrm{slack}(x \mid P_x) = s_x \ge 1 ,$$

so there exists $\mathbf{j} \in \{0,1\}^k$ for which slack$(x \mid P_{x,\mathbf{j}}) \ge 1$. One such $\mathbf{j}$ must be picked as the chosen pattern for x, because $w_{x,\mathbf{j}} > 0$ implies slack$(x \mid P_{x,\mathbf{j}}) > 0$. Consequently, the new value of P_x chosen at the end of the stage will satisfy the invariant $s_x \ge 1$.

After the last stage in the epoch, every set P_x is a singleton because, in the corresponding subcube of $\{0,1\}^b$, all bits have been fixed. P_x cannot be empty, because $|P_x \cap L_x| \ge s_x \ge 1$. Thus $|P_x \cap L_x| = s_x = 1$. $\square$

Lemma 3.7. *The set F collected at the end of an epoch satisfies $|F| \le |U|$.*

Proof. At the end of an epoch, we have

$$2|U| \overset{\text{lemma 3.5}}{\ge} \Phi_\ell \overset{\text{lemma 3.3}}{=} \sum_{x \in U} \frac{\mathrm{d_{conf}}(x)}{s_x} \overset{\text{lemma 3.6}}{=} \sum_{x \in U} \frac{|\{y \in \mathrm{N}(x) \cap U :\ P_x = P_y\}|}{1} = 2|F| . \qquad \square$$

Lemma 3.8. *Each epoch maintains the invariant that (U, χ) is a proper partial coloring and shrinks the set of uncolored vertices U to a smaller U' with $|U'| \le \frac{2}{3}|U|$.*

Proof. At the end of the epoch, each set P_x is a singleton and the sole color $c_x \in P_x$ is not used in $\mathrm{N}(x)$ because $s_x \neq 0$ (Lemma 3.6). Therefore, the set F collected at the end is precisely the set of edges that would be monochromatic *if* we colored each $x \in U$ with c_x. It follows that the end-of-epoch logic in the algorithm, which commits to these colors only on an *independent* set in the graph (V, F), maintains the invariant of a proper partial coloring.

By Lemma 2.1, (V, F) contains an independent set I of size

$$|I| \geq \frac{|U|^2}{2|F| + |U|} \overset{\text{lemma 3.7}}{\geq} \frac{|U|}{3}$$

and one can compute I in polynomial time. Therefore, $|U'| = |U| - |I| \leq \frac{2}{3}|U|$. □

The full version of the paper addresses the space and pass complexity of the algorithm in greater detail. The conclusions are given in the following lemmas.

LEMMA 3.9. *Algorithm 1 runs in $O(n \log^2 n)$ bits of space.* □

LEMMA 3.10. *Algorithm 1 runs in $O(\log \Delta \cdot \log\log \Delta)$ streaming passes.*

PROOF SKETCH. The number of passes used is dominated by the number of times the subroutine REFINE-PCC is called; for the ith epoch, this is $O(b/k) = O(\log \Delta / i)$. Summing over all $O(\log n)$ epochs gives a total of $O(\log \Delta \log\log \Delta)$ passes. See following lemma. □

This concludes the proof of our first major algorithmic result.

THEOREM 3.11 (FORMAL VERSION OF THEOREM 1.1). *There is a deterministic $O(\log \Delta \log\log \Delta)$-pass semi-streaming algorithm to $(\Delta + 1)$-color an n-vertex graph using $O(n \log^2 n)$ bits of space.* □

We can extend Algorithm 1 to obtain a deterministic (degree+1)-list coloring using essentially the same memory and number of passes, thus establishing Theorem 1.2 (see Appendix C for proof). Further, see the full version of the paper for implications of our result in communication complexity.

4 COLORING ROBUSTLY AGAINST AN ADAPTIVE ADVERSARY

We now turn to the adversarially robust streaming setting. See Section 2 to recall the definition of this model.

We require the following graph-theoretic concept.

Definition 4.1 (degeneracy). The *degeneracy* of a graph G is the least integer value κ for which every induced subgraph of G has a vertex of degree $\leq \kappa$.

Equivalently, it is the least value κ for which there is an acyclic orientation of the graph where the maximum out-degree of any vertex is $\leq \kappa$. By greedily assigning colors to the vertices of this orientation of G in reverse topological order, one obtains a proper $(\kappa + 1)$-coloring of G; we call this a (degeneracy + 1)-coloring.

We set up some terminology to help us outline our algorithm.

- **Buffer.** As the stream arrives, we explicitly store a buffer B of at most n edges. When the buffer is full (i.e., has reached its capacity of n edges), we empty it completely, and move on to storing the next batch of n edges.
- **Epoch.** We say we are in the ith epoch when we are storing the ith batch of n edges in our buffer.
- **Level.** We define levels for the vertices with respect to their degree in the (entire) graph seen so far. At the point of query, we say that a vertex is in level ℓ if its degree in the current graph is in $\left((\ell - 1)\sqrt{\Delta}, \ell\sqrt{\Delta}\right]$.
- **Zone (fast and slow).** We define zones (fast or slow) for the vertices with respect to their degree *in the buffer B*. At the time of query, we say that a vertex v is in the *fast* zone (or simply, a *fast vertex*) if $\deg_B(v) > \sqrt{\Delta}$; otherwise, we say that it is in the *slow* zone (or a *slow vertex*).
- **Block.** We have multiple independent coloring functions, denoted by h_i and g_i, that assign each node a color uniformly at random from a palette of suitable size.[6] As a result, we obtain a partition of the nodes into monochromatic classes that we call *blocks*. Each block produced by a coloring function f is called an f-*block*. More formally, for each c in the range of f, the set of nodes $\{v \in V : f(v) = c\}$ is an f-block.
- **f-Monochromatic.** An edge $\{u, v\}$ with $f(u) = f(v)$ is called f-monochromatic.
- **f-Sketches.** For a function f, we call the underlying sketch of the algorithm, which stores only f-monochromatic edges among the ones it receives, as an f-sketch.

We are now ready to present our algorithm. We give the pseudocode in Algorithm 3 and describe its analysis on a high-level below. The rigorous analysis is deferred to the full version of the paper.

First, we describe why Algorithm 3 colors the slow nodes using $O(\Delta^{5/2})$ colors while storing only $\widetilde{O}(n)$ edges in $\cup_{i=1}^{\Delta} A_i$. Next, we explain why it colors the fast nodes using $O(\Delta^{5/2})$ colors while storing $\widetilde{O}(n)$ edges in $\cup_{i=1}^{\sqrt{\Delta}} C_i$. Additionally, B stores $O(n)$ edges. These collectively imply the desired result.

Coloring slow vertices. Consider breaking the edge stream into Δ "chunks" of size n each. As described above, our buffer B basically stores a chunk from start to end, and then deletes it entirely to move on to the next chunk. We initialize Δ independent coloring functions $h_1, \ldots, h_\Delta$. For each i, the function h_i assigns each node a color from $[\Delta^2]$ uniformly at random. An h_i-sketch then processes the prefix of the stream until the end of chunk i. By definition (see above), it stores a received edge (u, v) in the set A_i only if it is h_i-monochromatic.

Suppose a query arrives in the current epoch $\texttt{curr} > 1$ (in the first epoch, we have the entire graph in store, and can deterministically return an optimal coloring). Fix a subgraph induced by *only* the slow vertices in an arbitrary $h_{\texttt{curr}-1}$-block on the edge set $A_{\texttt{curr}-1} \cup B$. Recolor this subgraph using an offline $(\Delta' + 1)$-coloring algorithm where Δ' is its max-degree. Now do this for each $h_{\texttt{curr}-1}$-block, using fresh palettes for the distinct blocks. The resultant coloring is our output for the slow nodes. We now argue that the number of edges stored in $\cup_{i=1}^{\Delta} A_i$ is roughly $O(n)$ and the number of colors used is $O(\Delta^{5/2})$.

Observe that for each i, the h_i-sketch processes the prefix of the stream until the end of chunk i. But note that, until that point, we based our outputs only on A_js for $j < i$, which are independent of h_i in particular. Therefore, we ensure that each h_i-sketch processes

[6]Thus, the functions are likely to generate *improper* colorings.

Algorithm 3 Robust Semi-Streaming $O(\Delta^{5/2})$-Coloring

```
Input: Edge insertion stream for an n-vertex graph $G = (V, E)$

Initialize:
1:  $d(v) \leftarrow 0$ for each $v \in V$                       ▷ degree counters
2:  for $i$ from 1 to $[\Delta]$ do                         ▷ for each of $\Delta$ possible epochs
3:      Let $h_i : V \rightarrow [\Delta^2]$ be uniformly random
4:      $A_i \leftarrow \emptyset$                                ▷ edges stored by $h_i$-sketch
5:  for $i$ from 1 to $\lceil\sqrt{\Delta}\rceil$ do                    ▷ for each of $\sqrt{\Delta}$ possible levels
6:      Let $g_i : V \rightarrow [\Delta^{3/2}]$ be uniformly random
7:      $C_i \leftarrow \emptyset$                                ▷ edges stored by $g_i$-sketch
8:  $B \leftarrow \emptyset$                                      ▷ buffer
9:  curr $\leftarrow 1$                                    ▷ current epoch number

Process(edge $\{u, v\}$):
10: if $|B| = n$ then                                 ▷ when buffer is full...
11:     $B \leftarrow \emptyset$; curr $\leftarrow$ curr $+ 1$          ▷ ...reset it and start next epoch
12: $B \leftarrow B \cup \{\{u, v\}\}$;                          ▷ store edge in buffer
13: $d(u) \leftarrow d(u) + 1$; $d(v) \leftarrow d(v) + 1$      ▷ increase deg($u$), deg($v$)
14: for $i$ from curr to $\Delta$ do               ▷ for epochs curr and above...
        ▷ store $h_i$-monochromatic edges in $A_i$
15:     if $h_i(u) = h_i(v)$ then $A_i \leftarrow A_i \cup \{\{u, v\}\}$
16: for $i$ from $\left\lceil \frac{\max\{d(u),d(v)\}}{\sqrt{\Delta}} \right\rceil + 1$ to $\Delta$ do ▷ for levels $\geq$ both $u, v$
        ▷ store $g_i$-monochromatic edges in $C_i$
17:     if $g_i(u) = g_i(v)$ then $C_i \leftarrow C_i \cup \{\{u, v\}\}$.

Query():
18: if curr $= 1$ then
19:     Return (degree+1)-coloring of $B$
    ▷ Partition vertices into fast ($F$) and slow ($S$) zones
20: $F \leftarrow \{v \in V : \deg_B(v) > \sqrt{\Delta}\}$; $S \leftarrow V \setminus F$
21: for $c$ from 1 to $[\Delta^2]$ do
        ▷ consider each $h_{\mathsf{curr}-1}$-block among slow vertices
22:     $S_{\mathsf{curr}-1}(c) \leftarrow \{w \in S : h_{\mathsf{curr}-1}(w) = c\}$
23:     Using fresh colors, (degree+1)-color subgraph induced by
        $S_{\mathsf{curr}-1}(c)$ on edge set $A_{\mathsf{curr}-1} \cup B$
24: for $\ell$ from 1 to $\lceil\sqrt{\Delta}\rceil$ do                       ▷ for each level $\ell$...
25:     for $c$ from 1 to $\lceil\Delta^{3/2}\rceil$ do
            ▷ consider each $g_\ell$-block among fast vertices in level $\ell$
26:         $F(\ell, c) \leftarrow \left\{w \in F : \left\lceil \frac{d(w)}{\sqrt{\Delta}} \right\rceil = \ell, \text{ and } g_\ell(w) = c\right\}$
27:         Using fresh colors, (degeneracy+1)-color subgraph in-
            duced by $F(\ell, c)$ on edge set $C_\ell \cup B$
28: Return resultant coloring for $S \cup F = V$
```

a part of the stream independent of their randomness. Thus, since h_is have range $[\Delta^2]$, an edge $\{u, v\}$ received by an h_i-sketch is h_i-monochromatic with probability $1/\Delta^2$. Since it receives at most $n\Delta$ edges, it *stores* only $O(n\Delta/\Delta^2) = O(n/\Delta)$ edges in expectation in A_i. By a Chernoff Bound argument, the actual value is tightly concentrated around this expectation w.h.p. Then, the Δ sets $A_1, \ldots, A_\Delta$ store roughly $O(n/\Delta \cdot \Delta) = O(n)$ edges in total w.h.p.

Now, we first verify that we *properly* color the subgraph of G induced by the slow nodes. Every slow node is in some $h_{\mathsf{curr}-1}$-block. Fix an arbitrary edge e of G in any $h_{\mathsf{curr}-1}$-block of slow vertices. By definition, e is $h_{\mathsf{curr}-1}$-monochromatic. Observe that e must be stored in $A_{\mathsf{curr}-1} \cup B$: if $e \in B$, then it is definitely stored, and otherwise, it was in an epoch $\leq$ curr $- 1$; therefore, the $h_{\mathsf{curr}-1}$-sketch received it and must have stored it in $A_{\mathsf{curr}-1}$. This means each such intra-block edge e is properly colored by the offline algorithm. Again, each inter-block edge is also properly colored since we use distinct palettes for distinct blocks.

Now we argue the color bound. For each slow node, an h_i-sketch receives at most Δ edges incident to it and hence, A_i stores $O(\Delta \cdot 1/\Delta^2) = O(1/\Delta)$ edges incident to it in expectation (by the previous argument). By a Chernoff Bound argument and taking union bound over all nodes, we get that each of them has degree roughly $O(\log n)$ in A_i w.h.p. Further, since these nodes are slow, they have degree at most $\sqrt{\Delta}$ in B. Therefore, the degree of each slow node in the edge set $A_{\mathsf{curr}-1} \cup B$ is $O(\sqrt{\Delta} + \log n) = O(\sqrt{\Delta})$ since we can assume that $\Delta = \Omega(\log^2 n)$ (if not, we can store the entire graph in semi-streaming space and then color it optimally). Hence, each $h_{\mathsf{curr}-1}$-block of slow nodes induced on $A_{\mathsf{curr}-1} \cup B$ is colored with a fresh palette of $O(\sqrt{\Delta})$ colors by the offline algorithm. There are Δ^2 many $h_{\mathsf{curr}-1}$-blocks, which means we use $O(\Delta^2 \cdot \sqrt{\Delta}) = O(\Delta^{5/2})$ colors.

Coloring fast vertices. To handle these, we use another $\sqrt{\Delta}$ independent coloring functions $g_1, \ldots, g_{\sqrt{\Delta}}$. Each g_i assigns each node a color from $[\Delta^{3/2}]$ uniformly at random. When an edge $\{u, v\}$ arrives, let ℓ be the maximum between the two levels of u and v. We send it to the g_i-sketches for all $i \geq \ell + 1$. Recall that a g_i-sketch then stores the edge in the set C_i only if it is g_i-monochromatic.

We prove that each g_i-sketch processes edges independent of their randomness. This is the tricky part. Intuitively, for each edge $\{u, v\}$ that a g_i-sketch receives, the levels of u and v were strictly smaller than i when it was inserted. The only values $g_j(u)$ and $g_j(v)$ that were used to return outputs until then were for $j < i$. Hence, $g_i(u)$ and $g_i(v)$ were completely unknown to the adversary when $\{u, v\}$ was inserted. Thus, the edge stream received by each g_i-sketch is independent of the randomness "that matters" in processing it. Hence, since the probability of each edge being g_i-monochromatic is $1/\Delta^{3/2}$, each g_i-sketch stores roughly $O(n\Delta/\Delta^{3/2}) = O(n/\sqrt{\Delta})$ edges in C_i. The total number of edges stored in $C_1, \ldots, C_{\sqrt{\Delta}}$ is then roughly $O(n/\sqrt{\Delta} \cdot \sqrt{\Delta}) = O(n)$.

When a query arrives, for each level ℓ, consider only the fast vertices in each g_ℓ-block. Then consider the subgraph induced by these vertices on the edge set $C_\ell \cup B$. Color it using a (degeneracy+1)-coloring offline algorithm. We prove that this colors the fast vertices properly with $O(\Delta^{5/2})$ colors.

To verify that it is a proper coloring, we need to show that the subgraph of G induced on each g_ℓ-block of fast vertices is stored in $C_\ell \cup B$. This follows from the "fastness" property of the nodes: if any such edge $\{u, v\}$ is not in B, then, since the degrees of nodes u and v increased by at least $\sqrt{\Delta}$ in the buffer, they must have been at levels lower than ℓ when $\{u, v\}$ was inserted. Therefore, it was fed to the g_ℓ-sketch, which stored it in C_ℓ since it is g_ℓ-monochromatic. Hence, each intra-block edge of fast vertices is properly colored by the offline algorithm. Also, each inter-block edge is properly colored since we use distinct palettes for distinct blocks.

Finally, for the color bound, we prove that the degeneracy of each g_ℓ-block induced on $C_\ell \cup B$ is $\leq \sqrt{\Delta}$. Then, each block is colored with $O(\sqrt{\Delta})$ colors, and there are $\Delta^{3/2}$ g_ℓ-blocks for each of the $\sqrt{\Delta}$ values of ℓ. Hence, the total number of colors used is $O(\sqrt{\Delta} \cdot \Delta^{3/2} \cdot \sqrt{\Delta}) = O(\Delta^{5/2})$.

Thus, we get the following result.

Theorem 4.2 (Formal version of Theorem 1.3). *There is an $O(\Delta^{5/2})$-coloring algorithm which is robust against adaptive adversaries with total error probability $\leq \delta$, and runs in $O(n \log^{O(1)} n \cdot \log \delta^{-1})$ bits of space, given oracle access to $O(n\Delta)$ bits of randomness.*

By adjusting parameters, we can obtain a corollary giving smooth colors/space tradeoff (see Appendix D for details).

Corollary 4.3 (Restated as Corollary D.1). *By adjusting parameters of Algorithm 3, we can obtain a robust $O(\Delta^{(5-3\beta)/2})$-coloring algorithm using $O(n\Delta^{\beta})$ space. This algorithm still requires oracle access to $O(n\Delta^{1-\beta})$ bits of randomness.*

Finally, we give a robust algorithm achieving semi-streaming space even in the "strict" model where all random bits do count to the space usage. This algorithm processes the stream in $O(\Delta)$ epochs, like Algorithm 3, but does not have a "fast zone", and uses a coloring function per epoch to split vertices into blocks. To reduce the number of random bits required, we randomly pick the coloring functions from 4-independent hash families; it turns out that this guarantees that the number of monochromatic edges is bounded with constant probability. This can be amplified to high probability. See Appendix E for more details.

Theorem 4.4 (Formal version of Theorem 1.4). *There is an adversarially robust $O(\Delta^3)$-coloring algorithm that runs in $\widetilde{O}(n)$ space and uses $\widetilde{O}(\Delta)$ random bits, with high probability.*

REFERENCES

[1] Abboud, A., Censor-Hillel, K., Khoury, S., and Paz, A. Smaller cuts, higher lower bounds. *CoRR abs/1901.01630* (2019).

[2] Alon, N., and Assadi, S. 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* (2020), vol. 176 of *LIPIcs*, pp. 6:1–6:22.

[3] Assadi, S., Chen, A., and Sun, G. Deterministic graph coloring in the streaming model. In *Proc. 54th Annual ACM Symposium on the Theory of Computing* (2022), pp. 261—-274.

[4] Assadi, S., Chen, Y., and Khanna, S. Sublinear algorithms for (Δ+ 1) vertex coloring. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms* (2019), pp. 767–786.

[5] Assadi, S., Kumar, P., and Mittal, P. 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* (2022), ACM, pp. 234–247.

[6] Attias, I., Cohen, E., Shechner, M., and Stemmer, U. A framework for adversarial streaming via differential privacy and difference estimators. *CoRR abs/2107.14527* (2021).

[7] Bamberger, P., Kuhn, F., and Maus, Y. Efficient deterministic distributed coloring with small bandwidth. In *Proc. 39th ACM Symposium on Principles of Distributed Computing* (2020), Y. Emek and C. Cachin, Eds., ACM, pp. 243–252.

[8] Ben-Eliezer, O., Eden, T., and Onak, K. Adversarially robust streaming via dense–sparse trade-offs. *CoRR abs/2109.03785* (2021).

[9] Ben-Eliezer, O., Jayaram, R., Woodruff, D. P., and Yogev, E. A framework for adversarially robust streaming algorithms. In *Proc. 39th ACM Symposium on Principles of Database Systems* (2020), p. 63–80.

[10] Ben-Eliezer, O., and Yogev, E. The adversarial robustness of sampling. In *Proc. 39th ACM Symposium on Principles of Database Systems* (2020), ACM, pp. 49–62.

[11] Bera, S. K., Chakrabarti, A., and Ghosh, P. 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)* (2020), vol. 168 of *LIPIcs*, pp. 11:1–11:21.

[12] Bera, S. K., and Ghosh, P. Coloring in graph streams. *CoRR abs/1807.07640* (2018).

[13] Bhattacharya, A., Bishnu, A., Mishra, G., and Upasana, A. 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* (2021), vol. 185 of *LIPIcs*, pp. 15:1–15:19.

[14] Bhattacharya, S., Chakrabarty, D., Henzinger, M., and Nanongkai, D. Dynamic algorithms for graph coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018* (2018), SIAM, pp. 1–20.

[15] Braverman, V., Hassidim, A., Matias, Y., Schain, M., Silwal, S., and Zhou, S. Adversarial robustness of streaming algorithms through importance sampling. *CoRR abs/2106.14952* (2021).

[16] Carter, L., and Wegman, M. N. Universal classes of hash functions. *J. Comput. Syst. Sci. 18*, 2 (1979), 143–154.

[17] Chaitin, G. J. Register allocation & spilling via graph coloring. In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, Boston, Massachusetts, USA, June 23-25, 1982* (1982), J. R. White and F. E. Allen, Eds., ACM, pp. 98–105.

[18] Chakrabarti, A., Ghosh, P., and Stoeckl, M. Adversarially robust coloring for graph streams. In *Proc. 13th Conference on Innovations in Theoretical Computer Science* (2022), pp. 37:1–37:23.

[19] Chang, Y., Li, W., and Pettie, S. An optimal distributed (Δ+1)-coloring algorithm? In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018* (2018), I. Diakonikolas, D. Kempe, and M. Henzinger, Eds., ACM, pp. 445–456.

[20] Cohen, E., Lyu, X., Nelson, J., Sarlós, T., Shechner, M., and Stemmer, U. On the robustness of countsketch to adaptive inputs. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA* (2022), vol. 162 of *Proceedings of Machine Learning Research*, PMLR, pp. 4112–4140.

[21] Cormode, G., Dark, J., and Konrad, C. Independent sets in vertex-arrival streams. In *Proc. 46th International Colloquium on Automata, Languages and Programming* (2019), pp. 45:1–45:14.

[22] Ghaffari, M., and Kuhn, F. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science* (2021), pp. 1009–1020.

[23] Halldorsson, M. M., Kuhn, F., Nolin, A., and Tonayan, T. Near-optimal distributed degree+1 coloring. In *Proc. 54th Annual ACM Symposium on the Theory of Computing* (2022), pp. 450–463.

[24] Hasan, W., and Motwani, R. Coloring away communication in parallel query optimization. In *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland* (1995), U. Dayal, P. M. D. Gray, and S. Nishio, Eds., Morgan Kaufmann, pp. 239–250.

[25] Hassidim, A., Kaplan, H., Mansour, Y., Matias, Y., and Stemmer, U. Adversarially robust streaming algorithms via differential privacy. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual* (2020).

[26] Kaplan, H., Mansour, Y., Nissim, K., and Stemmer, U. Separating adaptive streaming from oblivious streaming using the bounded storage model. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III* (2021), vol. 12827 of *Lecture Notes in Computer Science*, Springer, pp. 94–121.

[27] Kuhn, F. Faster deterministic distributed coloring through recursive list coloring. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms* (2020), S. Chawla, Ed., SIAM, pp. 1244–1259.

[28] Lotfi, V., and Sarin, S. A graph coloring algorithm for large scale scheduling problems. *Comput. Oper. Res. 13*, 1 (1986), 27–32.

[29] Peng, Y., Choi, B., He, B., Zhou, S., Xu, R., and Yu, X. Vcolor: A practical vertex-cut based approach for coloring large graphs. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)* (2016), IEEE, pp. 97–108.

[30] Schneider, J., and Wattenhofer, R. A new technique for distributed symmetry breaking. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010* (2010), A. W. Richa and R. Guerraoui, Eds., ACM, pp. 257–266.

[31] Stoeckl, M. Streaming algorithms for the missing item finding problem. In *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, To appear* (2023).

[32] Woodruff, D. P., and Zhou, S. Tight bounds for adversarially robust streams and sliding windows via difference estimators. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science* (2021), p. to appear.

A DETAILED ACCOUNT OF RELATED WORK

The study of graph coloring in the classical streaming model was initiated parallelly and independently by Bera and Ghosh [12] and Assadi, Chen, and Khanna [4]. The former work obtained an $O(\Delta)$-coloring algorithm in semi-streaming space, while the latter achieved a tight $(\Delta+1)$-coloring in the same amount of space. The latter work uses an elegant framework called *palette sparsification*: each node samples a list of roughly $\log n$ colors from the palette of size $\Delta+1$, and it is shown that w.h.p. there exists a proper list-coloring where each node uses a color only from its list. This immediately gives a semi-streaming $(\Delta+1)$-coloring algorithm since one can store only "conflicting" edges that can be shown to be only $\widetilde{O}(n)$ many w.h.p.[7] This framework implying semi-streaming coloring algorithms was then explored by Alon and Assadi [2] under various palette sizes (based on multiple color parameters) as well as list sizes. Their results also implied interesting algorithms for coloring triangle-free graphs and for (degree+1)-list coloring.

Very recently, Assadi, Chen, and Sun [3] studied deterministic Δ-based coloring and showed that for a single pass, no non-trivial streaming algorithm can be obtained. For semi-streaming space, any deterministic algorithm needs $\exp(\Delta^{\Omega(1)})$ colors, whereas for $O(n^{1+\alpha})$ space, $\Delta^{\Omega(1/\alpha)}$ colors are needed. Observe that these bounds are essentially matched by the trivial algorithm that stores the graph when $\Delta \le n^{\alpha}$ in order to $(\Delta+1)$-color it at the end; or just color the graph trivially with $n = \Delta^{1/\alpha}$ colors, without even reading the edges, when $\Delta > n^{\alpha}$. In light of this, a natural approach is to consider the problem allowing multiple passes over the input stream. They show that in just one additional pass, an $O(\Delta^2)$-coloring can be obtained deterministically, while with $O(\log\Delta)$ passes, we can have a deterministic $O(\Delta)$-coloring algorithm. Another very recent work on Δ-based coloring is that of Assadi, Kumar, and Mittal [5], who surprisingly proved Brooks's theorem in the semi-streaming setting: any (connected) graph that is not a clique or an odd cycle can be colored using exactly Δ colors in semi-streaming space.

Other works on streaming coloring include the work of Abboud, Censor-Hillel, Khoury, and Paz [1] who show that coloring an n-vertex graph with the optimal chromatic number of colors requires $\Omega(n^2/p)$ space in p passes. They also show that deciding c-colorability for $3 \le c < n$ (that might be a function of n) needs $\Omega((n-c)^2/p)$ space in p passes. Another notable work is that of Bera, Chakrabarti, and Ghosh [11], who considered the problem with respect to the *degeneracy* parameter that often yields more efficient colorings, especially for sparse graphs. They designed a semi-streaming $\kappa(1+o(1))$-coloring algorithm for graphs of degeneracy κ. They also proved that a combinatorially tight $(\kappa+1)$-coloring is not algorithmically possible in sublinear space. In particular, semi-streaming coloring needs $\kappa+\Omega(\sqrt{\kappa})$ colors. Bhattacharya, Bishnu, Mishra, and Upasana [13] showed that verifying whether an input vertex-coloring of a graph is proper is hard in the vertex-arrival streaming model where each vertex arrives with its color and incident edges. Hence, they consider a relaxed version of the problem that asks for a $(1\pm\epsilon)$-estimate of the number of conflicting edges. They prove tight bounds for this problem on adversarial-order streams and further study it on random-order streams. Recently, Halldorsson, Kuhn, Nolin, and Tonayan [23] gave a palette-sparsification-based semi-streaming algorithm for (degree + 1)-list-coloring for any arbitrary list of colors assigned to the nodes, improving upon the work of [2] whose algorithm works only when the color-list of each vertex v is $\{1,\dots,\deg(v)+1\}$. Note that all the works mentioned above are in the "static" streaming model and all their algorithms, except those in [3], are randomized and non-robust.

Starting with the work of Ben-Eliezer, Jayaram, Woodruff, and Yogev [9], the adversarially robust streaming model has seen a flurry of research in the last couple of years [6, 8, 10, 15, 18, 20, 25, 26, 31, 32]. Chakrabarti, Ghosh, and Stoeckl [18] were the first to study graph coloring in this model. They showed a separation between standard and robust streaming coloring algorithms by establishing lower bounds of (i) $\Omega(\Delta^2)$ colors for robust semi-streaming coloring, and (ii) $\Omega(n\Delta)$ space for robust $O(\Delta)$-coloring. In fact, they prove a smooth colors/space tradeoff: a robust K-coloring algorithm requires $\Omega(n\Delta^2/K)$ space. On the upper bound side, they design an $O(\Delta^3)$-coloring robust algorithm in semi-streaming space, with oracle access to $\widetilde{O}(n\Delta)$ many random bits. They also obtain an $O(\Delta^2)$-coloring in $\widetilde{O}(n\sqrt{\Delta})$ space (including random bits used).

B PROOF OF LEMMA 3.5

Lemma B.1 (Restatement of Lemma 3.5). *Suppose we start a particular epoch with the partial coloring (U,χ) and the initial, trivial PCC $(U,\chi,\mathcal{P}_0)$. Suppose there are ℓ stages in this epoch and the ith stage begins with the PCC $\mathcal{P}_i$. Let $\Phi_i := \Phi(U,\chi,\mathcal{P}_i)$ be the corresponding potential, for $0 \le i \le \ell$. Then $\Phi_0 \le |U|$ and $\Phi_\ell \le 2|U|$.*

Proof. Recalling that each $L_x \cap P_x = L_x = [\Delta+1]$ for the initial PCC, we use eqs. (1) and (7) to derive

$$\begin{aligned} s_x - \mathrm{d}_{\mathrm{conf}}(x) &= \max\{0, \Delta+1-|\mathrm{N}(x)\smallsetminus U|\} - |\mathrm{N}(x)\cap U| \\ &= \Delta+1-\deg(x) \\ &\ge 1\,. \end{aligned}$$

Thus, $\mathrm{d}_{\mathrm{conf}}(x)/s_x \le 1$ (and is not "0/0") for all $x \in U$. Lemma 3.3 now implies $\Phi_0 \le |U|$.

We now argue that, between each pair of successive stages, the potential Φ_i does not increase by much. First observe that when h is drawn uniformly at random from $\mathcal{H}$, and $u \ne v$,

$$\begin{aligned} &\Pr\left[g_{\mathbf{w}}(u,h(u)) = g_{\mathbf{w}}(v,h(v)) = \mathbf{j}\right] \qquad (8)\\ &= \Pr\left[g_{\mathbf{w}}(u,h(u)) = \mathbf{j}\right]\cdot\Pr\left[g_{\mathbf{w}}(v,h(v)) = \mathbf{j}\right]\\ &= \Pr\left[h(u)\in g_{\mathbf{w}}^{-1}(u,\mathbf{j})\right]\cdot\Pr\left[h(v)\in g_{\mathbf{w}}^{-1}(v,\mathbf{j})\right]\\ &\le w_{u,\mathbf{j}}w_{v,\mathbf{j}}\left(1+\frac{1}{8\log n}\right)^2\\ &\le e^{1/(4\log n)}w_{u,\mathbf{j}}w_{v,\mathbf{j}}\,. \qquad (9) \end{aligned}$$

To keep the rest the derivation compact, let us abbreviate "slack" to "sk." Also let $E' = E(G[u])$ be the set of edges in G between vertices in U, and let $C = e^{1/4\log n}$. The candidate PCCs $\mathcal{P}_h$ defined in line 8 are tightenings of the current PCC in which we pick subcubes according to the specific hash function h. With h chosen uniformly at random from $\mathcal{H}$:

$\mathbb{E}\,\Phi(U,\chi,\mathcal{P}_h)$

[7]The algorithm that is immediately implied is an exponential-time one where one can store the conflicting edges and obtain the list-coloring by brute force. An elaborate method was then needed to implement it in polynomial time.

$$
\begin{aligned}
&= \sum_{\{u,v\}\in E} \mathbb{E}\, \mathbb{1}_{u\in U \wedge v \in U \wedge P_{u,h}=P_{v,h}} \cdot \left(\frac{1}{\mathrm{sk}(u \mid P_{u,h})} + \frac{1}{\mathrm{sk}(v \mid P_{v,h})} \right) \\
&= \sum_{\{u,v\}\in E'} \sum_{\mathbf{j}\in\{0,1\}^k} \Pr\left[P_{u,h} = P_{u,\mathbf{j}} = P_{v,\mathbf{j}} = P_{v,h}\right] \cdot \\
&\qquad \left(\frac{1}{\mathrm{sk}(u \mid P_{u,\mathbf{j}})} + \frac{1}{\mathrm{sk}(v \mid P_{v,\mathbf{j}})} \right) \\
&\overset{\text{line 8}}{=} \sum_{\{u,v\}\in E'} \sum_{\mathbf{j}\in\{0,1\}^k} \mathbb{1}_{P_u=P_v} \Pr\left[g_{\mathbf{w}}(u,h(u)) = g_{\mathbf{w}}(v,h(v)) = \mathbf{j}\right] \cdot \\
&\qquad \left(\frac{1}{\mathrm{sk}(u \mid P_{u,\mathbf{j}})} + \frac{1}{\mathrm{sk}(v \mid P_{v,\mathbf{j}})} \right) \\
&\overset{\text{eq. (9)}}{\le} \sum_{\substack{\{u,v\}\in E' \\ P_u=P_v}} \sum_{\mathbf{j}\in\{0,1\}^k} C w_{u,\mathbf{j}} w_{v,\mathbf{j}} \cdot \left(\frac{1}{\mathrm{sk}(u \mid P_{u,\mathbf{j}})} + \frac{1}{\mathrm{sk}(v \mid P_{v,\mathbf{j}})} \right) \\
&\overset{\text{eq. (4)}}{=} C \sum_{\substack{\{u,v\}\in E' \\ P_u=P_v}} \sum_{\mathbf{j}\in\{0,1\}^k} \frac{\mathrm{sk}(u \mid P_{u,\mathbf{j}})}{\sum_{\mathbf{i}} \mathrm{sk}(u \mid P_{u,\mathbf{i}})} \cdot \frac{\mathrm{sk}(v \mid P_{v,\mathbf{j}})}{\sum_{\mathbf{i}} \mathrm{sk}(v \mid P_{v,\mathbf{i}})} \cdot \\
&\qquad \left(\frac{1}{\mathrm{sk}(u \mid P_{u,\mathbf{j}})} + \frac{1}{\mathrm{sk}(v \mid P_{v,\mathbf{j}})} \right) \\
&= C \sum_{\substack{\{u,v\}\in E' \\ P_u=P_v}} \sum_{\mathbf{j}\in\{0,1\}^k} \frac{\mathrm{sk}(u \mid P_{u,\mathbf{j}}) + \mathrm{sk}(v \mid P_{v,\mathbf{j}})}{\sum_{\mathbf{i}} \mathrm{sk}(u \mid P_{u,\mathbf{i}}) \cdot \sum_{\mathbf{i}} \mathrm{sk}(v \mid P_{v,\mathbf{i}})} \\
&= C \sum_{\substack{\{u,v\}\in E' \\ P_u=P_v}} \left(\frac{1}{\sum_{\mathbf{j}} \mathrm{sk}(u \mid P_{u,\mathbf{j}})} + \frac{1}{\sum_{\mathbf{j}} \mathrm{sk}(v \mid P_{v,\mathbf{j}})} \right) \\
&\overset{\text{lemma 3.4}}{\le} C \sum_{\substack{\{u,v\}\in E' \\ P_u=P_v}} \left(\frac{1}{\mathrm{sk}(u \mid P_u)} + \frac{1}{\mathrm{sk}(v \mid P_u)} \right) \\
&= C\Phi_i \,.
\end{aligned} \tag{10}
$$

Thus, picking $h^\star$ with $\Phi(U, \chi, \mathcal{P}_{h^\star})$ below average would ensure $\Phi_{i+1} \le e^{1/(4\log n)}\Phi_i$. However, due to precision constraints, each of lines 11 and 14 could contribute a relative error of $(1 + 1/(8\log n))$, so the $h^\star$ actually picked by the algorithm gives only the following weaker guarantee:

$$\Phi_{i+1} \le \left(1 + \frac{1}{8\log n}\right)^2 e^{1/(4\log n)}\Phi_i \le e^{1/(2\log n)}\Phi_i \,.$$

Since the number of stages in the epoch is $\ell \le \lceil b/k \rceil \le \log(\Delta + 1) \le \log n$, we have

$$\Phi_\ell \le \left(e^{1/(2\log n)}\right)^\ell \Phi_0 \le e^{1/2}|U| \le 2|U| \,. \qquad \square$$

C EXTENSION: DETERMINISTIC LIST COLORING

We can extend Algorithm 1 to handle the more general problem of (deg +1)-list-coloring. This requires a new technical lemma and a careful refinement of some of the low-level details of the previous algorithm.

Theorem C.1 (Restatement of Theorem 1.2). *Let C be a set of colors of size $O(n^2)$. There is a deterministic semi-streaming algorithm for (degree + 1)-list-coloring a graph G given a stream consisting of, in any order, the edges of G and (x, L_x) pairs specifying the list L_x of allowed colors for a vertex x, where $L_x \subseteq C$. The algorithm uses $O(n\log^2 n)$ bits of space and runs in $O(\log\Delta \log\log\Delta)$ passes.*

Here is a technical lemma that is key to the proof of the above.

Lemma C.2. *Let $s \ge 1$ be an integer, and let C be a set. There exists a family $\mathcal{F}$ of $O(|C|^2)$ partitions of C so that, for every collection $L_1, \dots, L_t$ of subsets of C:*

$$\frac{1}{|\mathcal{F}|} \sum_{\mathcal{R}\in\mathcal{F}} \sum_{i\in[t]} \max_{S\in\mathcal{R}} (|L_i \cap S| - 1) \le \frac{1}{\sqrt{s}} \sum_{i\in[t]} (|L_i| - 1)\,, \tag{11}$$

In particular, there must exist $Q \in \mathcal{F}$ where $\sum_{i\in[t]} \max_{S\in Q}(|L_i \cap S| - 1)$ is less than the right hand side.

Proof. Let $\mathcal{H}$ be a 2-universal hash family $C \to [s]$, with $|\mathcal{H}| = O(|C|^2)$. (For example, $\mathcal{H} = \{(x \mapsto (ax + b \bmod p) \bmod s) : a, b \in \mathbb{Z}_p, a \ne 0\}$ for p prime and $\ge |C|$, as per [16].) Let h be a randomly chosen element of $\mathcal{H}$, and let $\mathcal{R} = \{R_1, \dots, R_s\}$ be the random partition for which $R_i = \{x \in C : h(x) = i\}$. Consider the function $f(x) = x(x + 1)/2$ defined on $[0,\infty)$; because it is convex and increasing on $[0,\infty)$, $f^{-1}(x) = \sqrt{2x + 1/4} - 1/2$ is concave and increasing on $[0,\infty)$. Because for all $z \ge 1$, $z - 1 = f^{-1}(\binom{z}{2})$, we have for any $i \in [t]$ that:

$$\max_{j\in[s]} (|L_i \cap R_j| - 1) \le f^{-1}\left(\max_{j\in[s]} \binom{L_i \cap R_j}{2}\right) \le f^{-1}\left(\sum_{j\in[s]} \binom{L_i \cap R_j}{2}\right).$$

Taking expectations and using the concavity of f to apply Jensen's inequality:

$$
\begin{aligned}
\mathbb{E} \max_{j\in[s]} (|L_i \cap R_j| - 1) &\le \mathbb{E} f^{-1}\left(\sum_{j\in[s]} \binom{L_i \cap R_j}{2}\right) \\
&\le f^{-1}\left(\mathbb{E} \sum_{j\in[s]} \binom{L_i \cap R_j}{2}\right).
\end{aligned}
$$

Expressing the sum under the inverse function in terms of h lets us apply the universality of the hash family:

$$
\begin{aligned}
\mathbb{E} \sum_{j\in[s]} \binom{L_i \cap R_j}{2} &= \mathbb{E} \sum_{x,y\in L_i : x\ne y} \mathbb{1}_{h(x)=h(y)} \\
&= \sum_{x,y\in L_i : x \ne y} \Pr[h(x) = h(y)] \le \binom{|L_i|}{2} \frac{1}{s}\,.
\end{aligned}
$$

We briefly detour to prove an inequality for f, holding for all $z \ge 1$:

$$
\begin{aligned}
f\left(\frac{1}{\sqrt{s}}(z-1)\right) &= \frac{\frac{1}{\sqrt{s}}(z-1) \cdot (\frac{1}{\sqrt{s}}(z-1) + 1)}{2} \\
&= \frac{1}{s} \frac{(z-1)(z + \sqrt{s} - 1)}{2} \ge \frac{1}{s}\binom{z}{2},
\end{aligned}
$$

which implies $f^{-1}(\frac{1}{s}\binom{z}{2}) \le \frac{1}{\sqrt{s}}(z - 1)$. Thus:

$$\mathbb{E} \max_{j\in[s]} (|L_i \cap R_j| - 1) \le f^{-1}\left(\binom{|L_i|}{2}\frac{1}{s}\right) \le \sqrt{\frac{1}{s}}(|L_i| - 1)\,.$$

By linearity of expectation, it follows

$$\mathbb{E} \sum_{i\in[t]} \max_{j\in[s]} (|L_i \cap R_j| - 1) \le \sqrt{\frac{1}{s}} \sum_{i\in[t]} (|L_i| - 1)\,.$$

This is equivalent to Eq. 11, if we let $\mathcal{F}$ be the set of possible values of $\mathcal{R}$. □

Proof of Theorem 1.2. There are two main changes to the algorithm in Theorem 1.1. First, because the color lists L_x are no longer fixed, computing slack$(x \mid P_x \cap Q)$ for each $x \in U$ and $Q \in \mathcal{Q}^{(i)}$ requires counting both $|\{y \in \mathrm{N}(x) \setminus U : \chi(x) \in (P_x \cap Q)\}|$ as before, and $|P_x \cap Q \cap L_x|$. As both quantities are integers in $[0, \ldots, \Delta+1]$, and can be computed by incrementing counters each time an edge or (vertex, list of colors) pair arrives, the total space usage from this stage is still $O(\log \Delta)|U|2^k$.

The other change is that we now adaptively pick the sequence of partitions $\mathcal{Q}^{(1)}, \ldots, \mathcal{Q}^{(\ell)}$, and use more stages. Instead of letting the number ℓ of stages be $\lceil \log(\Delta+1)/k \rceil$, we use $\ell = \lceil 2\log(\Delta+1)/k \rceil + 1$ stages instead. For the first $\lceil 2\log(\Delta+1)/k \rceil$ stages, we adaptively construct partitions using Lemma C.2 on the L_x with s set to 2^k; the resulting partitions use $O(\ell \log |C|) = O(\log \Delta \log n)$ space to store in total.

Finding the best partition from Lemma C.2 is complicated by the fact that the algorithm can not exactly store the color lists L_x for each vertex. Let $\mathcal{F}$ be the family of partitions from Lemma C.2. At the start of each stage, we use four passes over the stream to identify a partition $\mathcal{R} \in \mathcal{F}$ for which the quantity $\sum_{x \in U} a_{\mathcal{R}}(P_x \cap L_x)$ is below average, for $a_{\mathcal{R}}(S) = \max_{R \in \mathcal{R}}(|S \cap R| - 1)$. This can be done using the same method as was used to identify an approximately sub-average hash function $h^\star$ in Algorithm 1. In the first pass, we split $\mathcal{F}$ into $O(|\mathcal{F}|^{1/4})$ parts, and compute $\sum_{\mathcal{R}} \sum_{x \in U} a_{\mathcal{R}}(P_x \cap L_x)$ for each part; after the pass completes, we pick the part with the least value of this sum, split it into $O(|\mathcal{F}|^{1/4})$ smaller parts, and repeat the process. The fourth pass will compute $\sum_{x \in U} a_{\mathcal{R}}(P_x \cap L_x)$ for individual partitions $\mathcal{R}$ of the family $\mathcal{F}$; we let $\mathcal{Q}^{(i)}$ be the best partition from this pass. All this is possible because the value of $a_{\mathcal{R}}(P_x \cap L_x)$ can be computed as soon as the pair (x, L_x) arrives in the stream. Consequently, it is possible to compute, for any family $\mathcal{F}$ of partitions, $\sum_{\mathcal{R} \in \mathcal{F}} \sum_{x \in U} a_{\mathcal{R}}(P_x \cap L_x)$ in a single pass over the stream, using $O(\log n)$ bits of space. (These sums have integer values, so no approximation is necessary.) As $|\mathcal{H}| = O(|C|^2) = O(n^4)$, each individual pass requires storing only $O(n \log n)$ bits worth of counters.

At the start of the first stage, since all $|L_x| \leq \Delta + 1$, we have $\sum_{x \in U}(|L_x \cap P_x| - 1) \leq \Delta |U|$. Letting j_x be the index of $P_{x,j} = P_x \cap Q_j^{(i)}$ chosen to succeed P_x, we have (due to Lemma C.2).

$$\begin{aligned}\sum_{x \in U}(|L_x \cap P_{x,j_x}| - 1) &\leq \sum_{x \in U} \max_{j \in [s]}(|L_x \cap P_x \cap Q_j^{(i)}| - 1) \\ &\leq 2^{-k/2} \sum_{x \in U}(|L_x \cap P_x| - 1)\end{aligned}$$

Each stage reduces $\sum_{x \in U}(|L_x \cap P_x| - 1)$ by a factor of $2^{-k/2}$, so after $\ell - 1 = \lceil 2\log(\Delta+1)/k \rceil$ stages, we have

$$\sum_{x \in U}(|L_x \cap P_x| - 1) \leq \Delta |U|(2^{-k/2(\ell-1)}) \leq \frac{\Delta}{\Delta+1}|U| \leq |U|$$

In the last stage, we set $\mathcal{Q} = \{\{x\} : x \in C$, where $C = \bigcup_{x \in U} L_x$. Unlike the other stages, where $|\mathcal{Q}| \leq 2^k$, we need to run an additional pass to record, for each $x \in U$, the values of $|L_x \cap P_x|$. This requires only $O(|U| \log n)$ bits. In the following pass to compute slack$(x \mid P_x \cap Q)$ for each $x \in U$ and $Q \in \mathcal{Q}$, we use the fact that slack$(x \mid P_x \cap Q)$ will only be one if $Q \subseteq P_x \cap L_x$ and there is no $y \in \mathrm{N}(x) \setminus U$ satisfying $\chi(y) \in Q$ to save space; instead of tracking sums for every $(x, Q) \in U \times C$ combination, we store a $\{0,1\}$ value for each $(x, Q) \in \cup_{x \in U}\{(x, Q) : Q \in L_x \cap P_x\}$ which is initialized to 1 and set to 0 if the stream contains an edge to a neighboring $y \in [n] \setminus U$ with color in Q. After this stage, the condition $|L_x| \leq 1$ holds, as required for the proof of Theorem 1.1 to work.

Despite the less efficient partitioning scheme, the algorithm still uses roughly the same amount of space; for all but the last stage, it still uses $2^k|U|$ counters. The last stage requires one bit for each element in a list L_x – but since $\sum_{x \in U}(|L_x| - 1) \leq |U|$, we have $\sum_{x \in U}|L_x| \leq 2|U|$, which implies only $2|U|$ bits are needed.

Storing the per vertex partitions P_x requires only $\ell k + \log(|C|) = O(\log n)$ bits, each, at a given point in the algorithm. As in the original algorithm, each partition P_x can be determined using the sequence of sets from $\mathcal{Q}^{(1)}, \ldots, \mathcal{Q}^{(\ell)}$ that contain it.

The analysis to prove that the potential does not increase by much requires no adjustment. □

D EXTENSION: SPACE-COLOR TRADEOFFS

Corollary D.1 (Restatement of Corollary 4.3). *By adjusting parameters of Algorithm 3, we can obtain a robust $O(\Delta^{(5-3\beta)/2})$-coloring algorithm using $O(n\Delta^{\beta})$ space and $O(n\Delta^{1-\beta})$ bits of oracle randomness.*

Proof. These parameter changes do not significantly affect the proofs of correctness for Algorithm 3.

As before, we assume that the powers of Δ given here are integers, and that $\Delta = \Omega(\log^2 n)$:

- Change the buffer replacement frequency (Line 10) from n to $n\Delta^{\beta}$. Because a graph stream with maximum degree Δ contains at most $n\Delta/2$ edges, reduce the number of epochs from Δ to $\Delta^{1-\beta}$. The `for` loops initializing, updating, and querying the variables h_i and A_i should have bounds adjusted accordingly.
- Reduce the range of the functions h_i from $[\Delta^2]$ to $[\Delta^{2-2\beta}]$. The expected number of edges stored in all of the sets A_i will now be roughly:

$$\frac{\text{\# epochs} \times |G|}{\text{\# slow blocks}} = \frac{\Delta^{1-\beta} \cdot O(n\Delta)}{\Delta^{2-2\beta}} = O(n\Delta^{\beta}),$$

  and with high probability, the space usage should not exceed this by more than a logarithmic factor.
- Increase the threshold for a vertex to be considered "fast" from $\sqrt{\Delta}$ to $\Delta^{(1+\beta)/2}$. To match this, the level of a vertex will now be computed as $\left\lceil \frac{d(v)}{\Delta^{(1+\beta)/2}} \right\rceil$, and the number of levels reduced from $\sqrt{\Delta}$ to $\Delta^{(1-\beta)/2}$. Again, all of the `for` loops related to the fast zone of the algorithm need to have their bounds adjusted.
- Reduce the range of the functions g_ℓ from $[\Delta^{3/2}]$ to $[\Delta^{(1-\beta)3/2}]$. The expected number of edges stored in all of the sets C_ℓ will now be roughly:

$$\frac{\text{\# levels} \times |G|}{\text{\# fast blocks}} = \frac{\Delta^{(1-\beta)/2} \cdot O(n\Delta)}{\Delta^{(1-\beta)3/2}} = O(n\Delta^{2\beta}).$$

Algorithm 4 Randomness-Efficient Adversarially Robust $O(\Delta^3)$-Coloring in Semi-Streaming Space

```
**Input**: Stream of edge insertions of an $n$-vertex graph $G = (V, E)$

**Initialize**:
Define $P := \lceil 10 \log n \rceil$, and let $\ell = 2^{\lfloor \log \Delta \rfloor}$ be the greatest power of 2 which is $\leq \Delta$
Let $\mathcal{U}$ be a 4-independent family of hash functions from $V$ to $[\ell^2]$, of size $\mathrm{poly}(n)$
1: for $i \in [\Delta], j \in [P]$ do
2:     $h_{i,j}$ be a uniformly random function from $\mathcal{U}$ mapping $V$ to $[\ell^2]$
3:     $D_{i,j} \leftarrow \emptyset$ ▹ Either a set of $h_{i,j}$-monochromatic edges, or $\perp$ after invalidation
4: $B \leftarrow \emptyset$    ▹ buffer of edges from this epoch
5: curr $\leftarrow 1$    ▹ current epoch number

**Process**(edge $\{u, v\}$):
6: if $|B| = n$ then
       ▹ End current epoch, switch to next
7:     $B \leftarrow \emptyset$; curr $\leftarrow$ curr $+ 1$
8: $B \leftarrow B \cup \{\{u, v\}\}$;    ▹ Update current buffer
9: for $i$ from curr $+ 1$ to $\Delta$, and $j \in [P]$ do
10:    if $h_{i,j}(u) = h_{i,j}(v)$ then ▹ For $h_{i,j}$-monochromatic edges...
11:        if $D_{i,j} \neq \perp \wedge |D_{i,j}| < \frac{7n}{\Delta}$ then
               ▹ Record edge in $D_{i,j}$ if there is space
12:            $D_{i,j} \leftarrow D_{i,j} \cup \{\{u, v\}\}$
13:        else
14:            $D_{i,j} \leftarrow \perp$    ▹ Wipe buffer $D_{i,j}$ if it gets too large

**Query**():
    ▹ This can fail if all $D_{\mathsf{curr},j} = \perp$
15: Let $k = \min\{j \in [P] : D_{\mathsf{curr},j} \neq \perp\}$
16: Let $\chi$ = greedy coloring of $D_{\mathsf{curr},k} \cup B$
17: Output the coloring where $y \in V$ is assigned $(\chi(y), h_{\mathsf{curr},j}(y)) \in [(\Delta + 1)] \times [\ell^2]$
```

The number of colors used by the vertices in the slow zone will be:

$$\begin{aligned}&\#\text{ slow blocks} \times (O(\#\text{ fast threshold}) + O(\log n))\\ &= \Delta^{2-\beta} O(\Delta^{(1+\beta)/2}) = O(\Delta^{(5-3\beta)/2}),\end{aligned}$$

and by the fast zone:

$$\#\text{ levels} \times \#\text{ fast blocks} \times \mathrm{O}(\#\text{ fast threshold}) + \log n))$$

$$= \Delta^{(1-\beta)/2} \Delta^{(1-\beta)3/2} O(\Delta^{(1+\beta)/2}) = O(\Delta^{(5-3\beta)/2}).$$

Combining the two, we find the modified algorithm produces a $O(\Delta^{(5-3\beta)/2})$ coloring with high probability. □

E A RANDOMNESS-EFFICIENT ROBUST ALGORITHM

In this section, we prove Theorem 4.4. The pseudocode is given in Algorithm 4. The lemmas below prove its correctness.

Lemma E.1. *Algorithm 4 is an adversarially robust $O(\Delta^3)$-coloring algorithm, which uses $\widetilde{O}(n)$ bits of space (including random bits used by the algorithm), with high probability.*

Proof. The only step of Algorithm 4 that an adversary could make fail is Line 15. By Lemma E.2, this happens with $1/\mathrm{poly}(n)$ probability. Assuming Line 15 does not fail, Lemma E.3 proves that the output of the algorithm is a valid $(\Delta + 1)\Delta^2$ coloring. Finally, Lemma E.4 verifies that Algorithm 4 uses at most $\widetilde{O}(n)$ bits of space and of randomness. □

We defer the proofs of the next two lemmas to the full version.

Lemma E.2. *Line 15 of Algorithm 4 will execute successfully, with high probability, on input streams provided by an adaptive adversary.* □

Lemma E.3. *If Line 15 does not fail, then Algorithm 4 outputs a valid $(\Delta + 1)(\Delta^2)$ coloring of the input graph.* □

The next lemma bounds the space usage.

Lemma E.4. *Algorithm 4 requires only $\widetilde{O}(n)$ bits of space; this includes random bits.*

Proof. Because $|\mathcal{U}| = O(\mathrm{poly}\, n)$, picking a random hash function from $\mathcal{U}$ requires only $O(\log n)$ random bits. As the algorithm stores $\Delta P = O(\Delta \log n)$ of these hash functions as $(h_{i,j})_{i \in [\Delta], j \in [P]}$, the total space needed by these function is $O(\Delta(\log n)^2)$.

Next, for each of the sets of edges $D_{i,j}$, for $i \in [\Delta]$, $j \in [P]$, Lines 11 through 14 ensure that $|D_{i,j}|$ is always $\leq \frac{7n}{\Delta} + 1$; sets that grow too large are replaced by $\perp$. Since edges can be stored using $O(\log n)$ bits, the total space usage of all the $D_{i,j}$ is $O\left(\frac{n}{\Delta}\right) \Delta P \cdot O(\log n) = O\left(n(\log n)^2\right)$. Similarly, the buffer B never contains more than n edges, since it is reset when the condition of Line 6 is true; thus B can be stored with $O(n \log n)$ bits. The counter curr is negligible.

In total, the algorithm needs $O\left(\Delta(\log n)^2\right) + O\left(n(\log n)^2\right) = \widetilde{O}(n)$ bits of space. □