Improved Streaming Edge Coloring

Shiri Chechik[*] Hongyi Chen[†] Tianyi Zhang[‡]

Abstract

Given a graph, an edge coloring assigns colors to edges so that no pairs of adjacent edges share the same color. We are interested in edge coloring algorithms under the W-streaming model. In this model, the algorithm does not have enough memory to hold the entire graph, so the edges of the input graph are read from a data stream one by one in an unknown order, and the algorithm needs to print a valid edge coloring in an output stream. The performance of the algorithm is measured by the amount of space and the number of different colors it uses.

This streaming edge coloring problem has been studied by several works in recent years. When the input graph contains n vertices and has maximum vertex degree Δ, it is known that in the W-streaming model, an $O(\Delta^2)$-edge coloring can be computed deterministically with $\tilde{O}(n)$ space [Ansari, Saneian, and Zarrabi-Zadeh, 2022], or an $O(\Delta^{1.5})$-edge coloring can be computed by a $\tilde{O}(n)$-space randomized algorithm [Behnezhad, Saneian, 2024] [Chechik, Mukhtar, Zhang, 2024].

In this paper, we achieve polynomial improvement over previous results. Specifically, we show how to improve the number of colors to $\tilde{O}(\Delta^{4/3+\epsilon})$ using space $\tilde{O}(n)$ deterministically, for any constant $\epsilon > 0$. This is the first deterministic result that bypasses the quadratic bound on the number of colors while using near-linear space.

[*]Tel Aviv University, shiri.chechik@gmail.com

[†]State Key Laboratory for Novel Software Technology, New Cornerstone Science Laboratory, Nanjing University, hychener01@gmail.com

[‡]ETH Zürich, tianyi.zhang@inf.ethz.ch

1 Introduction

Let $G = (V, E)$ be an undirected graph on n vertices with maximum vertex degree Δ. An edge coloring of G is an assignment of colors to edges in E such that no pairs of adjacent edges share the same color, and the basic objective is to understand the smallest possible number of colors that are needed in any edge coloring, which is called the edge-chromatic number of G. It is clear that the total number of colors should be at least Δ, and a simple greedy algorithm can always find an edge coloring using $2\Delta - 1$ colors. By the celebrated Vizing's theorem [Viz65] and Shannon's theorem [Sha49], $(\Delta+1)$-edge coloring and $\lfloor 3\Delta/2 \rfloor$-edge coloring always exist in simple and multi-graphs respectively, and these two upper bounds are tight in some hard cases.

The edge coloring problem has been studied widely from the algorithmic perspective. There have been efficient algorithms for finding good edge coloring in various computational models, including sequential [Arj82, GNK+85, Sin19, Ass25, BCC+24, BCSZ25, ABB+24], dynamic [BM17, BCHN18, DHZ19, Chr23, BCPS24, Chr24], online [CPW19, BGW21, SW21, KLS+22, BSVW24, BSVW25, DGS25], and distributed [PR01, EPS14, FGK17, GKMU18, BBKO22, CHL+20, Ber22, Chr23, Dav23] models. In this paper, we are particularly interested in computing edge coloring under the streaming model where we assume the input graph does not fit in the memory of the algorithm and can only be accessed via one pass over a stream of all edges in the graph. Since the output of edge coloring is as large as the graph size, the algorithm cannot store it in its memory. To address this limitation, the streaming model is augmented with an output stream in which the algorithm can write its answers during execution, and this augmented streaming model is thus called the W-streaming model.

Edge coloring in the W-streaming model was first studied in [BDH+19], and improved by follow-up works [CL21, ASZZ22] which led to a deterministic edge coloring algorithm using $O(\Delta^2)$ colors and $O(n)$ space, or $O(\Delta^2/s)$ colors and $O(ns)$ space as a general trade-off. In [GS24], the authors improved the trade-off to $O(\Delta^2/s)$ colors and [1]$\tilde{O}(n\sqrt{s})$ memory, yet it does not break the quadratic bound in the most natural $\tilde{O}(n)$ memory regime; this algorithm is randomized but can be derandomized in exponential time. The quadratic upper bound of colors was subsequently bypassed in [SB24, CMZ24] where it was shown that an $O(\Delta^{1.5})$-edge coloring can be computed by a randomized algorithm using $\tilde{O}(n)$ space. Furthermore, in [SB24] the authors obtained a general trade-off of $O(\Delta^{1.5}/s + \Delta)$ colors and $\tilde{O}(ns)$ space which is an improvement over [GS24].

1.1 Our Results

In this paper, we focus on the basic $\tilde{O}(n)$-memory setting and improve the recent $\Delta^{1.5}$ randomized upper bound to $\Delta^{4/3+\epsilon}$.

Theorem 1.1. *Given a simple graph $G = (V, E)$ on n vertices with maximum vertex degree Δ, for any constant $\epsilon > 0$, there is a randomized W-streaming algorithm that outputs a proper edge coloring of G using $O\left((\log \Delta)^{O(1/\epsilon)} n\right)$ space and $O\left((\log \Delta)^{O(1/\epsilon)} \Delta^{4/3+\epsilon}\right)$ different colors; both upper bounds hold in expectation.*

Furthermore, we also show that our algorithm can be derandomized using bipartite expanders based on error correcting codes at the cost of slightly worse bounds, as stated below.

Theorem 1.2. *Given a simple graph $G = (V, E)$ on n vertices with maximum vertex degree Δ, for any constant $\epsilon > 0$, there is a deterministic W-streaming algorithm that outputs a proper edge coloring of G using $O\left((\log \Delta)^{O(1/\epsilon)} \cdot (1/\epsilon)^{O(1/\epsilon^3)} \cdot \Delta^{4/3+\epsilon}\right)$ colors and $O\left(n \cdot (\log \Delta)^{O(1/\epsilon^4)}\right)$ space.*

[1]$\tilde{O}(f)$ hides $\log f$ factors.

1.2 Technical Overview

Previous Approaches. Using a deterministic general-to-bipartite reduction from [GS24], we can assume the input graph $G = (L \cup R, E)$ is bipartite. Also, it suffices to color only a constant fraction of all edges in G, because we can recurse on the rest $1 - \Omega(1)$ fraction of G which only incurs an extra factor of $O(\log \Delta)$ on the total number of different colors.

Let us begin by recapping the randomized $\tilde{O}(\Delta^{1.5})$-edge coloring from [SB24]. Since the algorithm has $\tilde{O}(n)$ bits of memory, we can assume that input graph is read from stream in batches of size $\Theta(n)$. If the subgraph formed by every batch has maximum degree at most $\Delta^{1/2}$, then we can allocate $O(\Delta^{1/2})$ new colors for each batch, using $O(\Delta^{1.5})$ colors in total. Thus, the main challenge arises when some batches contain subgraphs with maximum degree exceeding $\Delta^{1/2}$.

To simplify the problem, let us assume that in each batch of $O(n)$ edges, every vertex in R has degree $d > \Delta^{1/2}$. To assign colors to edges, organize a table of colors of size $\Delta \times (\Delta/d)$, represented as a matrix $C[i, j]$ with indices $1 \le i \le \Delta$ and $1 \le j \le \Delta/d$. Then, for every vertex $u \in L$, draw a random shift r_u uniformly at random from $[\Delta]$. During the algorithm, each vertex $u \in L$ keeps a counter c_u of its degree in the stream so far, and each vertex $v \in R$ keeps a counter b_v of the number of batches in which it has appeared so far. Then, to color a single batch, for edge (u, v), the algorithm tentatively assigns the color $C[r_u + c_u, b_v]$ (indices are under modulo Δ).

Clearly, edges incident on the same vertex $u \in L$ receive different colors, because the counter c_u is incremented for each edge (u, v); also edges incident on the same vertex $v \in R$ but arriving in different batches are different, because the values of counter b_v are different. Randomization ensures that edges incident on $v \in R$ within the same batch receive mostly different colors from the same column in $C[*, b_v]$. Consider all the d neighbors of v in a single batch $u_1, u_2, \ldots, u_d$. Since all the random shifts r_{u_i} are independent and all the counters c_{u_i} are deterministic (they only depend on the input stream), with high probability, most row indices $(r_{u_i} + c_{u_i}) \bmod \Delta$ will be different.

Bypassing the $\Delta^{1.5}$ Bound. To better understand the bottleneck in the approach of [SB24], consider the following case. If every batch forms a regular subgraph with uniform degree d, then we can reduce the size of the color table from $\Delta \times (\Delta/d)$ to $(\Delta/d) \times (\Delta/d)$, since each vertex $u \in L$ could only appear in Δ/d different batches. So the main difficult case is when the batches are subgraphs of **unbalanced degrees**. As an extreme example, consider when vertices in L have degree 1, while the vertices in R have degree d. For the rest, our main focus will be on this extreme case, and show how to obtain a $\Delta^{1+\epsilon}$-edge coloring which is almost optimal.

The flavor of our approach is similar to [CMZ24]. Divide all Δ batches into Δ/d phases, each phase consisting of d consecutive batches. Let D be a parameter which upper bounds the number of batches in which any vertex $v \in R$ could appear during a single phase. Then, the maximum degree of a single phase would be bounded by Dd, so in principle we should be able to color all edges within this phase using $O(Dd)$ different colors. To implement the coloring procedure, at the beginning of each phase, prepare D fresh palettes $C_1, C_2, \ldots, C_D$, each of size d, and assign each batch in this phase a palette C_i where i is chosen from $[D]$ uniformly at random. To keep track of the colors already used around each vertex, we maintain the following data structures.

- Each vertex $v \in R$ keeps a list $U_v \subseteq \{C_1, C_2, \ldots, C_D\}$ of used palettes.
- Each vertex $u \in L$ initializes a random shift $r_u \in [d]$ at the beginning of the algorithm.

When a batch of edges $F \subseteq E$ arrives, we will use the assigned palette C_i to color this batch. For every edge $(u, v) \in F$, if $C_i \notin U_v$, then tentatively assign the color $(r_u + c_i) \bmod d$ in C_i to edge (u, v), where c_i denotes the number of times that palette C_i has been assigned to previous batches. If $C_i \in U_v$, we mark the edge as uncolored. Since the palettes are assigned to batches randomly, in expectation, a constant fraction of edges will be successfully colored. In the case that multiple edges incident on v are assigned the same color, we retain only one such edge and mark the remaining edges as uncolored. Since all the random shifts r_u's are uniformly random and independent of the randomness of counters c_i, most tentative colors around any vertex $v \in R$ in this batch should be different. Once the batch F is processed, add C_i to all lists $U_v, v \in R$ such that $\deg_F(v) = d$.

To reason about the space usage, we can argue that the total size of the lists $U_v, \forall v \in R$ does not exceed $O(n)$, because a palette C_i appears in the list U_v only when v receives d edges in a batch which uses palette C_i. Since the total number of edges in a phase is $O(dn)$, the total list size should be bounded by $O(n)$. In this way, we are able to store all the used palettes of vertices in R; this is not new to our approach, and a similar argument was also used in [CMZ24]. The new ingredient is the way we store the used colors around vertices in L. Here we have exploited the fact that vertices in L have low degrees in each batch, so their progress in every palette C_i is **synchronized**; that is, we only need to store a single counter c_i which is the number of times that C_i appears, and then the next tentative color of $u \in L$ would be $(r_u + c_i) \bmod d$.

The above scheme would use $O(Dd)$ different colors in a single phase, so $O(\Delta D)$ colors across all Δ/d phases. When $D \leq \Delta^{1/4}$, this bound would be much better than $\Delta^{1.5}$. So, what if $D > \Delta^{1/4}$? To deal with this case, we will apply a two-layer approach. Specifically, let us further group every D consecutive phases as a super-phase, so there are at most $\Delta/Dd < \Delta^{1/4}$ super-phases in total (recall that we were assuming $d > \Delta^{1/2}$). Within a super-phase, we will allocate Δ fresh colors which are divided into Δ/Dd different color packages $P_1, P_2, \ldots, P_{\Delta/Dd}$, where each color package P_i is further divided into D palettes of size d as $P_i = C_{i,1} \cup C_{i,2} \cup \cdots \cup C_{i,D}$. In this way, the total number of colors would be $\Delta^{5/4}$.

Then, like what we did before, for each phase in a super-phase, assign a color package P_i from $P_1, P_2, \ldots, P_{\Delta/Dd}$ uniformly at random. Within each phase, we will stick to its assigned color package P_i and reuse the algorithm within a phase we have described before. Since a color package is shared by multiple phases, each vertex $v \in R$ needs to store a list $U_{v,2}$ which stores all the packages P_i it has used so far, and in any phase where P_i is assigned but P_i is already contained in $U_{v,2}$, we would not color any edges incident on v in this particular phase. By repeating the same argument, we can argue that the total size of all the lists $U_{v,2}, \forall v \in R$ is bounded by $O(n)$ as well.

We can further generalize this two-layer approach to multiple layers and reduce the total number of colors to $\Delta^{1+\epsilon}$. However, this only works with the most unbalanced setting where vertices on L always have constant degrees in each batch of input. In general, when low-degree side has degree d', our algorithm needs $d' \cdot \Delta^{1+\epsilon}$ colors. If d' is large, then we would switch to the color table approach from [SB24]. Balancing the two cases, we end up with $\Delta^{4/3+\epsilon}$ colors overall.

Derandomization using Bipartite Expanders. Randomization is used both in the unbalanced case and in the regular case. To replace the choices of the random shifts r_u and random color package assignments, we will show one can apply unbalanced bipartite expanders [TSUZ01, GUV09, KTS22] in a black box manner. However, for the random shifts used in the regular case where we apply the color table idea from [SB24], it would not be enough to use an arbitrary bipartite expander, because the counters c_u's could be different and possibly damage the expansion guarantee; for example, it is not clear to us how to apply the bipartite expander construction based on Parvaresh–Vardy Codes [GUV09]. To fix this issue, it turns out that it would be most convenient to use the bipartite expander construction based on multiplicity codes [KTS22].

2 Preliminaries

Basic Terminologies. For any integer k, let us conventionally define $[k] = \{1, 2, \ldots, k\}$. For any set S and integer k, let $k \odot S$ be the multi-set that contains exactly k copies of each element in S.

Let $G = (V, E)$ denote the simple input graph on n vertices and m edges with maximum degree Δ. For any subset of edges $F \subseteq E$ and any vertex $u \in V$, let $\deg_F(u)$ be the number of edges in F incident on u. Sometimes we will also refer to the subgraph (V, F) simply as F.

Problem Definition. In the edge coloring problem, we need to find an assignment of colors to edges such that adjacent edges have distinct colors, and the objective is to minimize the total number of different colors.

In the W-streaming model introduced by [DFR09, GSS17], all edges of the graph G arrive one by one in the stream in an arbitrary order, and the algorithm makes one pass over the stream to perform its computation. For the task of edge coloring, since the algorithm has much less space than the total size of the graph, it cannot store all the edge colors in its memory. To output the answer, the algorithm is given a write stream in which it can print all colors.

Next, to set the stage in a convenient way, we will state several reductions for the problem.

General-to-Bipartite Reduction. Let us first simplify the problem by a deterministic reduction from edge coloring in general graphs to bipartite graphs.

Lemma 2.1 (Corollary 3.2 in [GS24])**.** *Given an algorithm $\mathcal{A}$ for streaming edge coloring on bipartite graphs using $f(\Delta)$ colors and $g(n, \Delta)$ space, there is an algorithm $\mathcal{B}$ using $O(f(\Delta))$ colors and $O(g(n, \Delta) \log n + n \log n \log \Delta)$ space in general graphs. Furthermore, this reduction is deterministic.*

Reduction to Fixed Degree Pairs. By the above reduction, we focus on edge coloring for bipartite graphs $G = (V, E)$, where $V = L \cup R$. Since the algorithm has space $\Omega(n)$, we can divide the input stream into at most $O(m/n) = O(\Delta)$ batches, each batch containing n edges. For any vertex $u \in V$ and any batch F, let $\deg_F(u)$ be the number of edges in F incident on u. For every pair of integers $0 \leq l, r \leq \log_2 \Delta$ and every batch F, let $F_{l,r} = \{(u, v) \in F \mid \deg_F(u) \in [2^l, 2^{l+1}), \deg_F(v) \in [2^r, 2^{r+1})\}$, and define $E_{l,r} = \bigcup_F F_{l,r}$ over all batches F in the stream and $m_{l,r} = |E_{l,r}|$.

In the main text, we will devise an algorithm to handle edges in $F_{l,r}$ for any fixed pair of (l, r). The final coloring is obtained by taking the union of the colors over all (l, r), which will blow up the number of colors and space by a factor of $O(\log^2 \Delta)$.

Adapting to an Unknown Δ. So far we have assumed that the maximum vertex degree Δ in G is known in advance. Our algorithm can be adapted to an unknown value Δ in a standard way. Specifically, we can maintain the value Δ_t which is the maximum degree of the subgraph containing the first t edges in the input stream. Whenever $\Delta_t \in (2^{k-1}, 2^k]$, we apply our algorithm with $\Delta = 2^k$ to color all the edges. When k increments at some point, we restart a new instance of the algorithm with a new choice $\Delta = 2^k$ and continue to color the edges with fresh colors. Overall, the total number of colors will not change asymptotically.

3 Randomized $\Delta^{4/3+\epsilon}$ Edge Coloring

This section is devoted to the proof of Theorem 1.1.

Reduction to Partial Coloring. To find an edge coloring of a graph, it was shown in [CMZ24] that it suffices to color a constant fraction of edges in E if we do not care about $O(\log \Delta)$ blow-ups in the number of colors. Roughly speaking, for the uncolored edges, we can view them as another instance of edge coloring, and solve it recursively. The following statement formalizes this reduction.

Lemma 3.1 (implicit in [CMZ24])**.** *Suppose there is a randomized streaming algorithm $\mathcal{A}$ with space $g(n, \Delta)$ space, such that for each edge in $e \in E$, it assigns a color from $[f(\Delta)]$ or marks it as $\perp$ (uncolored) and print this information in the output stream, with the guarantee that there are at least δm edges in expectation which receive actual colors, for some value $\delta > 0$. Then, there is a randomized edge coloring algorithm $\mathcal{B}$ which uses at most $O\left(\frac{\log \Delta}{\delta} f(\Delta)\right)$ colors and $O\left(\frac{\log \Delta}{\delta} g(n, \Delta)\right)$ space in expectation.*

Proof. TOPROVE 0 □

According to the reductions in the preliminaries, we will focus on partial edge coloring in bipartite graphs with fixed degree pairs. More specifically, our algorithm consists of the following two components.

Lemma 3.2. *Fix a parameter $\epsilon > 0$. Given an graph $G = (V, E)$ on n vertices with maximum vertex degree Δ, for any constant $\epsilon > 0$, and fix an integer pair (l, r), there is a randomized W-streaming algorithm that outputs a partial coloring of edges $F_{l,r}$ such that least $\delta m_{l,r}$ edges receive colors in expectation; here $\delta = 2^{-O(1/\epsilon)}$ is also a constant. The algorithm uses $O\left((\log \Delta)^{O(1/\epsilon)} \cdot \Delta^{1+\epsilon} \cdot 2^l\right)$ colors and $O\left((\log \Delta)^{O(1/\epsilon)} \cdot n\right)$ space.*

Lemma 3.3. *Given an graph $G = (V, E)$ on n vertices with maximum vertex degree Δ, and fix an integer pair (l, r), there is a randomized W-streaming algorithm that outputs a partial coloring of edges $F_{l,r}$ such that least $m_{l,r}/2$ edges receive colors in expectation. The algorithm uses $O\left(\Delta + \Delta^2/2^{l+r}\right)$ colors and $O(n)$ space.*

Proof. TOPROVE 1 □

3.1 Proof of Lemma 3.2

3.1.1 Data Structures

Before presenting the details of the data structures we will use in the main algorithm, let us start with a brief technical overview. The data structures consist of three components.

- **Forest structures on batches.** This part organizes edge input batches into parameterized forests in a way similar to B-trees. The height of the forests will be $O(1/\epsilon)$, and the branch size of each level will be an integer power of 2 in the range $[\Delta^\epsilon, \Delta]$, and so the total number of different forests will be bounded by $(\log \Delta)^{O(1/\epsilon)}$.

- **Color allocation on forests.** Each forest will be associated with a separate color set of size roughly $\Delta^{1+\epsilon}$. On each level of a forest, we will randomly partition the color set of this forest into color subsets (packages) and assign them to the tree nodes on this level. We will also make sure that the color packages on tree nodes are nested; that is, the color package of a node is a subset of the color package of its parent node.

  Each vertex will choose colors for its incident edges according to its own frequencies of accumulating edges in the stream. For example, when a vertex is gathering a large number of incident edges in a short interval of batches, it would use color packages on tree nodes with large branch sizes.

- **Vertex-wise data structures.** To ensure that we never assign the same color to adjacent edges, each vertex needs to remember which colors it has already used around it. To efficiently store all the previously used colors, we will show that the used colors are actually concentrated in color packages, so each vertex only needs to store the tree nodes corresponding to those color packages, instead of storing every used colors individually.

Next, let us turn to the details of the data structures we have outlined above.

Forest Structures on Batches. Without loss of generality, assume $l \leq r$, $2^r > \Delta^\epsilon$, and Δ^ϵ is an integer power of two; note that if $2^r \leq \Delta^\epsilon$, then the maximum degree inside each batch $F_{l,r}$ is at most Δ^ϵ, so we can use a fresh palette of size $O(\Delta^\epsilon)$, so the total number of colors would be $O(\Delta^{1+\epsilon})$.

As we have done in the preliminaries, partition the entire input stream into at most $m/n \leq \Delta$ batches of size n. We will create at most $(\log \Delta)^{O(1/\epsilon)}$ different forest structures, where each leaf represents a batch, and the internal tree nodes represent consecutive batches. Each forest is parameterized by an integer vector $\mathbf{f} = (f_1, f_2, \ldots, f_h)$ such that:

- f_i is an integer power of two;
- $2^{r+1} = f_0 \geq f_1 \geq f_2 \geq \cdots \geq f_h = \Delta^\epsilon$, and $2^{r+1} \cdot \prod_{i=1}^{h-1} f_i \leq m/n$.

We can assume the total number of batches in the input stream is an integer multiple of $2^{r+1} \cdot \prod_{i=1}^{h-1} f_i \leq m/n$ by padding empty batches. Given such a vector $\mathbf{f} = (f_1, f_2, \ldots, f_h)$, we will define a forest structure $\mathcal{T}_\mathbf{f}$ with $h+1$ levels by a bottom-up procedure; basically we will build a forest on all the batches with branching factors $2^{r+1}, f_1, \ldots, f_h$ bottom-up. More specifically, consider the following inductive procedure.

- All the batches will be leaf nodes on level 0. Partition the sequence of all batches into groups of consecutive $f_0 = 2^{r+1}$ batches. For each group, create a tree node at level-1 connecting to all leaf nodes in that group.
- Given any $1 \leq i \leq h-1$, assume we have defined all the tree nodes on levels $1 \leq j \leq i$. List all the tree nodes on level i following the same ordering of the batches, and partition this sequence of level-i nodes into groups of size f_i. For each group, create a tree node at level-$(i+1)$ that connects to all nodes in the group.

According to the definition, in general, tree levels up to k have the same topological structure for all frequency vectors which share the same first $k-1$ coordinates $f_1, f_2, \ldots, f_{k-1}$. For any node $N \in V(\mathcal{T}_\mathbf{f})$, let $\mathcal{T}_\mathbf{f}(N)$ be the subtree rooted at node N. By the above definition, for any $1 \leq k \leq h$, for any level-k node N, the set of all leaf nodes contained in the subtree $\mathcal{T}_\mathbf{f}(N)$ form a sub-interval of the batch sequence with length $2^{r+1} \cdot \prod_{i=1}^{k-1} f_i$.

Color Allocation on Forests. Next, we will allocate color packages at each tree node of each forest $\mathcal{T}_\mathbf{f}$. By construction, each forest structure $\mathcal{T}_\mathbf{f}$ is a tree of $h+1$ levels (from level-0 to level-h), and each tree is rooted at a level-h node. Go over each tree $T \subseteq \mathcal{T}_\mathbf{f}$ and we will allocate color packages to tree nodes in a top-down manner.

- **Basis.** At the root N of the tree T, allocate a color package $\mathcal{C}^N$ with $25 \cdot 2^{l+r+2} \cdot \prod_{i=1}^{h} (5f_i)$ new colors. Divide package $\mathcal{C}^N$ evenly into $5f_h = 5\Delta^\epsilon$ smaller packages (colors are ordered alphabetically in a package):

$$\mathcal{C}^N = \mathcal{C}_1^N \sqcup \mathcal{C}_2^N \sqcup \cdots \sqcup \mathcal{C}_{5\Delta^\epsilon}^N$$

Here, symbol $\sqcup$ means disjoint union. By construction of tree T, N has f_{h-1} different children $N_1, N_2, \ldots, N_{f_{h-1}}$. Let sequence $(i_1, i_2, \ldots, i_{5f_{h-1}})$ be a random permutation of $(f_{h-1}/\Delta^\epsilon) \odot [5\Delta^\epsilon]$. For each $1 \leq j \leq f_{h-1}$, define color package $\mathcal{C}^{N_j} = \mathcal{C}_{i_j}^N$.

- **Induction.** In general, suppose we have defined color packages for tree nodes on levels $k, k+1, \ldots, h$. Go over all tree nodes on level k. Inductively, assume $|\mathcal{C}^N| = 25 \cdot 2^{l+r+2} \cdot \prod_{i=1}^{k}(5f_i)$. Divide color package $\mathcal{C}^N$ into $5f_k$ smaller packages (colors are ordered alphabetically in a package):

$$\mathcal{C}^N = \mathcal{C}_1^N \sqcup \mathcal{C}_2^N \sqcup \cdots \sqcup \mathcal{C}_{5f_k}^N$$

  Let $i_1, i_2, \ldots, i_{5f_{k-1}}$ be a random permutation of $(f_{k-1}/f_k) \odot [5f_k]$. For each such level-$k$ node N, by construction, it has f_{k-1} children $N_1, N_2, \ldots, N_{f_{k-1}}$. Then, for each $1 \leq j \leq f_{k-1}$, define color package $\mathcal{C}^{N_j} = \mathcal{C}_{i_j}^N$.

By the above construction, each leaf node is allocated a color package of size $25 \cdot 2^{l+r+2}$. We will call each such smallest color package a **palette**. Notice that, by definition, the same palette may appear at multiple leaf nodes (which represent input batches). For any leaf node N (or equivalently, a batch), let $\mathsf{cnt}(\mathcal{C}^N)$ count the total number of times that palette $\mathcal{C}^N$ is also allocated to previous leaf nodes (batches). Note that the counters $\mathsf{cnt}(*)$ do not depend on the input stream and can be computed in advance.

Vertex-Wise Data Structures. To carry out the streaming algorithm, we will also maintain some data structures for vertices in V. At the beginning of the streaming algorithm, for each vertex $u \in L$, draw a random shift $r_u \in [3 \cdot 2^{r+1}]$ uniformly at random; these random shifts r_u's will remain fixed throughout the entire execution of the algorithm.

The main part is to specify the data structures associated with each vertex $v \in R$. For each forest $\mathcal{T}_{\mathbf{f}}$, we will maintain a set of marked nodes $M_{v,\mathbf{f}} \subseteq V(\mathcal{T}_{\mathbf{f}})$ throughout the streaming algorithm.

Invariant 3.1. *We will ensure the following properties regarding the marked nodes $M_{v,\mathbf{f}}$ throughout the execution of the streaming algorithm.*

(1) No two nodes in $M_{v,\mathbf{f}}$ lie on the same root-to-leaf path in the forest $\mathcal{T}_{\mathbf{f}}$. Furthermore, suppose the current input batch corresponds to a leaf F, and let P be the unique tree path from F to the tree root. Then, any node $N \in M_{v,\mathbf{f}}$ is a child of some node on P.

(2) For any node $N \in \mathcal{T}_{\mathbf{f}}$ on level-k such that $M_{v,\mathbf{f}} \cap V(\mathcal{T}_{\mathbf{f}}(N)) \neq \emptyset$, let $F_1, F_2, \ldots, F_s \subseteq E$ be all the input batches which correspond to leaf nodes in subtree $\mathcal{T}_{\mathbf{f}}(N)$. Take the union of the batches $U = F_1 \cup F_2 \cup \cdots \cup F_s$. Then, we have $\deg_U(v) \geq 2^{r-k} \cdot \prod_{i=1}^{k} f_i$.

(3) For any previous input batch F' before F such that:

- *F and F' are in the same tree component in $\mathcal{T}_{\mathbf{f}}$,*
- *$\deg_{F'}(v) \in [2^r, 2^{r+1})$,*
- *v used some colors in $\mathcal{C}^{F'}$ during the algorithm,*

we guarantee that F' must be contained in some subtree $\mathcal{T}_{\mathbf{f}}(N)$ for some $N \in M_{v,\mathbf{f}}$.

(4) The choices of $M_{v,\mathbf{f}}$ is independent of the randomness of $\{r_u \mid u \in L\}$ and the randomness of color packages $\mathcal{C}^*$, and they only depend deterministically on the input stream.

Let us explain the purpose of the above properties. Invariant 3.1(1)(2) ensures that the algorithm only uses $\tilde{O}(n)$ space in total. Invariant 3.1(3) allows us to rule out all colors used previously, preventing duplicate assignments to edges incident on the same vertex. Invariant 3.1(4) will be technically important for the analysis of randomization.

3.1.2 Algorithm Description

Next, let us turn to describe the coloring procedure. At the beginning, all the marked sets $M_{v,\mathbf{f}}$ are empty for any $v \in R$ and any frequency vector $\mathbf{f}$. Upon the arrival of a new input batch F, we will describe the procedures that update the marked sets and assign edge colors.

Preprocessing Marked Sets. Since the current input batch has changed due to the arrival of F, we may have violated Invariant 3.1(2) as the root-to-leaf tree path may have changed. Therefore, we first need to update all the marked sets with the following procedure named UPDATEMARKSET(F).

Go over every vertex $v \in R$ and every frequency vector $\mathbf{f}$. Consider the position of F in the forest $\mathcal{T}_{\mathbf{f}}$, and let P be the root-to-leaf path in $\mathcal{T}_{\mathbf{f}}$ ending at leaf F. First, remove all marked nodes $N \in M_{v,\mathbf{f}}$ which are not in the same tree as P; note that this may happen when F is the first leaf in a new tree component of $\mathcal{T}_{\mathbf{f}}$.

Next, go over every node W lying on the tree path P, for any child node N of W, if (1) $V(\mathcal{T}_{\mathbf{f}}(N)) \not\ni F$ and (2) $V(\mathcal{T}_{\mathbf{f}}(N)) \cap M_{v,\mathbf{f}} \neq \emptyset$, then remove all nodes in $V(\mathcal{T}_{\mathbf{f}}(N)) \cap M_{v,\mathbf{f}}$ from $M_{v,\mathbf{f}}$ and add N to $M_{v,\mathbf{f}}$. In other words, we elevate the positions of all the marked nodes in $M_{v,\mathbf{f}}$ to their ancestors which are children of P. See Figure 1 for an illustration.

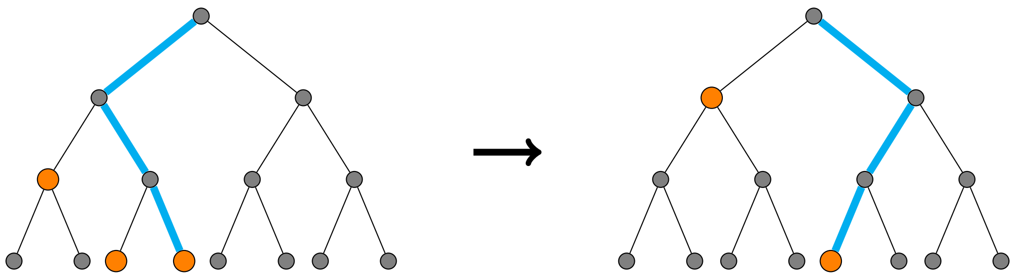

Figure 1: In this picture, it shows an example of a forest $\mathcal{T}_{\mathbf{f}}$ where the orange nodes are the marked ones, and the blue path is the root-to-leaf path ending at the current input batch F. Upon the arrival of a new input batch F, we need to update the root-to-leaf tree path and the marked sets accordingly.

Coloring $F_{l,r}$. To find colors for edges in $F_{l,r}$, we first need to find a proper palette for each vertex $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$. We will describe an algorithm FREQVEC(F) which finds a proper frequency vector $\mathbf{f}_v = (f_1, f_2, \ldots, f_h)$ for each such $v \in R$ and use the palette $\mathcal{C}^F$ associated with leaf node F in the forest $\mathcal{T}_{\mathbf{f}_v}$. To identify the proper frequency vector $\mathbf{f}_v$, we will figure out each coordinate $f_1, f_2, \ldots, f_h$ one by one inductively.

- **Basis.** Let N be the unique level-1 node containing F (recall that level-1 nodes are defined irrespective of frequency vectors). Find $f_1 \in \{\Delta^\epsilon, 2\Delta^\epsilon, \ldots, 2^{r+1}\}$ which is the smallest integer such that there exists a frequency vector $\mathbf{f}' = (f_1, f_2', f_3', \ldots)$, so that $M_{v,\mathbf{f}'}$ contains less than f_1 children of N. Note that such a vector must exist, because $f_1 = 2^{r+1}$ always satisfies this requirement.

  If $f_1 = \Delta^\epsilon$, return the 1-dimensional vector $\mathbf{f}_v = (f_1)$.

- **Induction.** In general, assume we have specified a prefix $f_1, f_2, \ldots, f_k$ for some $k \geq 1$. Note that all the forests $\mathcal{T}_{\mathbf{f}'}$ share the same topological structures from level 0 up to $k+1$. Let N be the unique level-$(k+1)$ node containing F conditioning on $f_1, f_2, \ldots, f_k$. Check all the frequency vectors $\mathbf{f}'$ that begin with the prefix $f_1, f_2, \ldots, f_k$, and find the smallest possible $f_{k+1} \in \{\Delta^\epsilon, 2\Delta^\epsilon, \ldots, f_k\}$ such that there exists a frequency vector $\mathbf{f}' = (f_1, f_2, \ldots, f_k, f_{k+1}, f_{k+2}', \ldots)$ under the condition that $M_{v,\mathbf{f}'}$ contains less than

f_{k+1} children of N. Note that such a vector must exist, because $f_{k+1} = f_k$ always satisfies this requirement.

If $f_{k+1} = \Delta^\epsilon$, then return the vector $\mathbf{f}_v = (f_1, f_2, \ldots, f_k, f_{k+1})$.

- **Choosing palette $\mathcal{C}_v$.** Once we have finished the above inductive process, we need to choose a palette $\mathcal{C}_v$ for v which contains $25 \cdot 2^{l+r+2}$ different colors. Basically, we will choose the palette associated with leaf node F in forest $\mathcal{T}_{\mathbf{f}_v}$ as $\mathcal{C}_v$, but must ensure that $\mathcal{C}_v$ has not been used previously.

  To make sure of this, let P denote the root-to-leaf tree path ending at leaf node F in $\mathcal{T}_{\mathbf{f}_v}$. Travel down the path from root to leaf and enumerate all the encountered tree nodes. For every such tree node N, if N already contains a sibling $W \in M_{v,\mathbf{f}_v}$ and $\mathcal{C}^W = \mathcal{C}^N$, then abort this procedure and assign $\mathcal{C}_v \leftarrow \emptyset$.

  In the end, if this procedure terminates without abortion, then assign $\mathcal{C}_v \leftarrow \mathcal{C}^F$; here $\mathcal{C}^F$ refers to the palette of size $25 \cdot 2^{l+r+2}$ associated with leaf node F in forest $\mathcal{T}_{\mathbf{f}_v}$.

In this way, we can select a frequency vector $\mathbf{f}_v$ for every vertex $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$. Let $\mathcal{C}_v$ be the palette assigned to leaf node F by forest $\mathcal{T}_{\mathbf{f}_v}$. Next, we are going to color all edges in $F_{l,r}$ using colors from $\mathcal{C}_v$ for edges incident on $v \in R$. Go over each vertex $u \in L$, and list its neighbors $v_1, v_2, \ldots, v_k, k < 2^{l+1}$ in $F_{l,r}$. For any index $1 \le i \le k$, if $\mathcal{C}_{v_i} \neq \emptyset$, then reserve a tentative color to edge (u, v_i), which is the κ_i-th color in palette $\mathcal{C}_{v_i}$ and

$$\kappa_i = 5 \cdot 2^{l+1} \cdot (\mathsf{cnt}(\mathcal{C}_{v_i}) + r_u) + i \mod 25 \cdot 2^{l+r+2}$$

Recall that $\mathsf{cnt}(\mathcal{C}_v)$ counts the number of times that palette $\mathcal{C}_v$ has appeared at previous leaf nodes under $\mathcal{T}_{\mathbf{f}}$. Notice that $k < 2^{l+1}$, these tentative colors are different around u.

On the side of $v \in R$, it checks all the tentative colors on all of its edges in $F_{l,r}$. If a tentative color is used more than once, then it keeps only one of them, and turns other tentative colors back to $\perp$. Finally, for each edge $e \in F_{l,r}$, assign e its own tentative color and print it in the output stream.

Postprocessing Marked Sets. After processing the input batch F, for every vertex $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$, add node F to $M_{v,\mathbf{f}_v}$; note that we mark F irrespective of whether $\mathcal{C}_v$ is $\emptyset$ or not as will be important for establishing Invariant 3.1(4). The whole algorithm is summarized in Algorithm 1.

3.1.3 Proof of Correctness

To begin with, let us first bound the total number of different colors that we could possibly use.

Lemma 3.4. *The total number of colors that the algorithm could use is $O((\log \Delta)^{O(1/\epsilon)} \Delta^{1+\epsilon} \cdot 2^l)$.*

Proof. TOPROVE 2 □

Let us next state some basic properties of any forest $\mathcal{T}_{\mathbf{f}}$.

Lemma 3.5. *Each palette is used by at most $2^{r+1}/\Delta^\epsilon$ different batches.*

Proof. TOPROVE 3 □

Corollary 3.1. *During the algorithm, the values of the counters $\mathsf{cnt}(\mathcal{C})$ never exceed $2^{r+1}/\Delta^\epsilon$ for any palette $\mathcal{C}$.*

To bound the total space, it is clear that the bottleneck is storing all the marked sets $M_{v,\mathbf{f}}$, since all the forest structures and color assignments only require space $O\left((\log \Delta)^{O(1/\epsilon)} \Delta\right)$.

Algorithm 1: ColorLowDeg(F)

```
   /* pre-processing marked sets */
1  for $v \in R$ and frequency vector $\mathbf{f}$ do
2      call UpdateMarkSet($F$) (as described in Section 3.1.2) to remove marked nodes
         in previous tree components in $\mathcal{T}_{\mathbf{f}}$,
3      and elevate the positions of all the marked nodes in $M_{v,\mathbf{f}}$ to their ancestors which
         are children of $P$;
   /* select frequency vector $\mathbf{f}_v$ and palette $\mathcal{C}_v$ */
4  for $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$ do
5      call FreqVec($F$) (as described in Section 3.1.2) to determine the frequency vector
         $\mathbf{f}_v = (f_1, f_2, \ldots)$ progressively;
6      find palette $\mathcal{C}_v$ while ensuring no conflicts with sibling nodes;
   /* assign colors for $F_{l,r}$ */
7  for $u \in L$ do
8      let $v_1, v_2, \ldots, v_k$ be all of $u$'s neighbors in $F_{l,r}$;
9      assign edge $(u, v_i)$ a tentative color which is the $\kappa_i$-th color in palette $\mathcal{C}_{v_i}$, where
         $\kappa_i = 5 \cdot 2^{l+1} \cdot (\mathsf{cnt}(\mathcal{C}_{v_i}) + r_u) + i \mod 25 \cdot 2^{l+r+2}$;
10 for $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$ do
11     discard tentative colors that appear more than once around $v$ in $F_{l,r}$;
12     output the unique tentative colors to the output stream;
   /* post-processing marked sets */
13 for $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$ do
14     add $F$ to $M_{v,\mathbf{f}_v}$;
```

Lemma 3.6. *If Invariant 3.1 is satisfied, then at any point during the execution of the algorithm, for any given frequency vector $\mathbf{f}$, the total size of marked sets $\sum_{v \in R} |M_{v,\mathbf{f}}|$ is bounded by $O(2^h n)$. Consequently, the total space usage is bounded by $O((\log \Delta)^{O(1/\epsilon)} n)$.*

Proof. TOPROVE 4 □

Now, our main focus will be on verifying all the properties in Invariant 3.1.

Lemma 3.7. *Invariant 3.1 is preserved by the algorithm throughout its execution.*

Proof. TOPROVE 5 □

Next, let us turn to the color assignment part. First, we need to verify that the algorithm never assigns the same color twice around the neighborhood of a single vertex.

Lemma 3.8. *In the output stream, the algorithm never prints the same color for two adjacent edges.*

Proof. TOPROVE 6 □

Finally, let us prove that the algorithm successfully colors a good fraction of all edges in the input stream.

Lemma 3.9. *The total number of colored edges in the output stream is at least δm in expectation, where $\delta = 2^{-O(1/\epsilon)}$.*

Proof. TOPROVE 7 □

3.2 Proof of Lemma 3.3

Without loss of generality, assume Δ is a power of 2.

3.2.1 Data Structures

At the beginning, for each vertex $u \in L$, draw a random number $s_u \in [\Delta/2^l]$ uniformly at random. For each vertex $v \in R$, draw a random number $t_v \in [\Delta/2^r]$ uniformly at random.

As in the preliminary steps, the input stream is divided into batches of size n (except for the last one). For each $u \in L$, let $\mathsf{cnt}(u)$ count the number of previous batches F where $\deg_F(u) \in [2^l, 2^{l+1})$, and symmetrically let $\mathsf{cnt}(v)$ count the number of previous batches F where $\deg_F(v) \in [2^r, 2^{r+1})$ for $v \in R$.

Additionally, we will use a palette matrix $\mathcal{C}$ of size $\frac{\Delta}{2^l} \times \frac{\Delta}{2^r}$, where each entry in $\mathcal{C}[i,j]$ corresponds to a distinct palette of size Δ_0, with $\Delta_0 \triangleq \left\lceil 4 \cdot \left(\frac{2^{l+r+1}}{\Delta} + 1\right)\right\rceil$. All the colors can be represented as integers in the range $\left[\frac{\Delta_0 \cdot \Delta^2}{2^{l+r}}\right]$ naturally.

3.2.2 Algorithm Description

Let us describe the coloring procedure upon the arrival of a new input batch F. For each vertex $u \in L$, propose a tentative row index $x_u = (s_u + \mathsf{cnt}(u)) \mod \frac{\Delta}{2^l}$. For each vertex $v \in R$, propose a tentative column index $y_v = (t_v + \mathsf{cnt}(v)) \mod \frac{\Delta}{2^r}$. For each edge $(u,v) \in F$, we will assign a color from the matrix $\mathcal{C}[x_u, y_v]$ in the following manner.

Let $E_{x,y}$ be the set of edges $(u,v) \in F_{l,r}$ where $(x_u, y_v) = (x,y)$, and let $G_{x,y}$ be the subgraph whose edge set is $E_{x,y}$. To color $G_{x,y}$ with only Δ_0 colors, we need to prune it so that its maximum degree does not exceed Δ_0, which is done in this way: for each edge $(u,v) \in E_{x,y}$, if $\max\{\deg_{E_{x,y}}(u), \deg_{E_{x,y}}(v)\} > \Delta_0$, mark it as $\perp$ (uncolored) and remove it from $G_{x,y}$. Finally, since $G_{x,y}$ is a bipartite graph, we can apply the Δ_0-edge coloring algorithm [COS01] to color $G_{x,y}$ using the palette $\mathcal{C}[x,y]$ which has size Δ_0.

Finally, for each vertex $u \in L$ such that $\deg_F(u) \in [2^l, 2^{l+1})$, increment the counter $\mathsf{cnt}(u)$ by 1; also, increment the counters for $v \in R$ in a symmetric way. The whole algorithm is summarized in Algorithm 2.

Algorithm 2: ColorHighDeg(F)

```
1 define $x_u \leftarrow s_u + \mathsf{cnt}(u) \mod \Delta/2^l, \forall u \in L$;
2 define $y_v \leftarrow t_v + \mathsf{cnt}(v) \mod \Delta/2^r, \forall v \in R$;
3 define $E_{x,y} = \{(u,v) \in F_{l,r} \mid (x_u, y_v) = (x,y)\}, \forall (x,y)$;
4 for every pair $(x,y)$ and edge $(u,v) \in E_{x,y}$ do
      /* prune $G_{x,y}$ to cap its maximum degree */
5     remove edge $(u,v)$ from $E_{x,y}$ if $\max\{\deg_{E_{x,y}}(u), \deg_{E_{x,y}}(v)\} > \Delta_0$;
6 for every pair $(x,y)$ do
7     use palette $\mathcal{C}[x,y]$ to color $E_{x,y}$;
8 increment counters $\mathsf{cnt}(*)$;
```

3.2.3 Proof of Correctness

Proper Coloring. To prove that the colored edges form a proper coloring, we need to show that for any two distinct edges $e_1 = (u, v_1)$ and $e_2 = (u, v_2)$ sharing a common vertex u, their colors are different. Since the algorithm is symmetric for L and R, we can assume $u \in L$. There are several cases below.

- If e_1 and e_2 use different matrix entries in $\mathcal{C}$, their colors are already distinct.
- Suppose both edges use palette $\mathcal{C}[x,y]$. Then, there are two sub-cases below.

- If e_1 and e_2 belong to different batches F_1 and F_2 with batch counters $\mathsf{cnt}^{(1)}(u)$ and $\mathsf{cnt}^{(2)}(u)$, we have $x \equiv s_u + \mathsf{cnt}^{(1)}(u) \equiv s_u + \mathsf{cnt}^{(2)}(u) \pmod{\Delta/2^l}$. Since $\deg_F(u) \in [2^l, 2^{l+1})$, cnt_u never exceeds $\Delta/2^l$. Thus, $\mathsf{cnt}^{(1)}(u) = \mathsf{cnt}^{(2)}(u)$, which leads to a contradiction.
- If e_1 and e_2 belong to the same batch, they belong to the same subgraph $G_{x,y}$. The correctness of the offline coloring algorithm guarantees they get distinct colors.

Space Usage. For each vertex u, we maintain its batch counter cnt_u and random shift s_u for $u \in L$ (or t_v for $v \in R$), requiring $O(n)$ space in total. During the batch coloring process, we also store the indices x_u and y_v, which require additional $O(n)$ space. Furthermore, each subgraph $G_{x,y}$ has at most n edges, so the offline coloring algorithm requires $O(n)$ space. These spaces are reused across different subgraph coloring processes and different batches. Therefore, the overall space complexity is $O(n)$.

Number of Colors. The total number of colors is given by

$$\frac{\Delta}{2^l} \cdot \frac{\Delta}{2^r} \cdot \Delta_0 = O\left(\Delta + \frac{\Delta^2}{2^{l+r}}\right).$$

Next, we show that at least half of the edges get colored in expectation.

Lemma 3.10. *During the algorithm, at least a $1/2$ fraction of the edges are colored in expectation.*

Proof. TOPROVE 8 □

4 Derandomization via Bipartite Expanders

In this section we will de-randomize Theorem 1.1 and prove Theorem 1.2. By the reductions from the preliminary section, it suffices to prove the following two statements.

Lemma 4.1. *Fix a parameter $\epsilon > 0$. Given a graph $G = (V, E)$ on n vertices with maximum vertex degree Δ, for any constant $\epsilon > 0$, and fix an integer pair (l, r), there is a deterministic W-streaming algorithm that outputs a coloring of all edges in $F_{l,r}$. The algorithm uses $O\left((\log \Delta)^{O(1/\epsilon)} \cdot (1/\epsilon)^{O(1/\epsilon^3)} \cdot \Delta^{1+2\epsilon} \cdot 2^l\right)$ colors and $O\left(n \cdot (\log n)^{O(1/\epsilon^4)}\right)$ space.*

Lemma 4.2. *Fix a parameter $\epsilon > 0$. Given a graph $G = (V, E)$ on n vertices with maximum vertex degree Δ, and fix an integer pair (l, r), there is a deterministic W-streaming algorithm that outputs a coloring of all edges in $F_{l,r}$. The algorithm uses $O\left(\Delta^{1+\epsilon} + \Delta^{2+\epsilon}/2^{l+r}\right)$ colors and $O\left(n \cdot (\log n)^{O(1/\epsilon^3)}\right)$ space.*

Proof. TOPROVE 9 □

4.1 Bipartite Expanders via Multiplicity Codes

We will use explicit constructions of unbalanced bipartite expanders from [KTS22] for de-randomization. For Lemma 4.1 we will only use bipartite expanders in a black-box manner, but for Lemma 4.2 we will also have to leverage some properties of the multiplicity code itself.

Definition 4.1 (bipartite expanders). *Given a bipartite graph $H = (A \cup B, I)$, H is a (K, D)-expander if for every $S \subseteq A$ of size K, the number of different neighbors of S in B is at least $K \cdot D$.*

Let q be a prime number and $\mathbb{F}_q$ be the corresponding finite field. Let a, b be two integers. Any vector in $\mathbb{F}_q^a$ can be interpreted as a uni-variate polynomial over $\mathbb{F}_q$, that is, as an element $f \in \mathbb{F}_q^{<a}[X]$. Define a mapping $\Gamma : \mathbb{F}_q^a \times \mathbb{F}_q \to \mathbb{F}_q^{b+2}$ as $\Gamma(f, x) = \big(x, f(x), f^{(1)}(x), \dots, f^{(b)}(x)\big)$, where $f^{(i)}$ is the i-th iterated derivative of f in $\mathbb{F}_q[X]$.

Lemma 4.3 (multiplicity codes are bipartite expanders [KTS22])**.** *Construct a bipartite graph $H = (A \cup B, I)$ where $A = \mathbb{F}_q^a, B = \mathbb{F}_q^{b+2}$, and for each vertex in A which is represented as a uni-variate polynomial $f \in \mathbb{F}_q^{<a}[X]$, connect vertex f to neighbors $\Gamma(f, x) \in B, \forall x \in \mathbb{F}_q$. Then, under the condition that $15 \leq b + 2 \leq a \leq q$, H is a (K, D)-expander for every $K > 0$ and $D = q - \frac{a(b+2)}{2} \cdot (qK)^{\frac{1}{b+2}}$.*

We will also need an upper bound on the degrees on the right-hand side of the bipartite expanders.

Lemma 4.4. *Let $H = (A \cup B, I)$ be the above bipartite graph by multiplicity codes. Then, for any $v \in B$, $\deg_H(v) \leq q^{a-b-1}$.*

Proof. TOPROVE 10 □

Online Perfect Matching in Bipartite Expanders. For the convenience of algorithm description, let us first extract a building block of our main algorithm here which is called *online perfect matching in bipartite expanders*. In this problem, we can set up a bipartite expander $H = (A \cup B, I)$ of our own, and then an adversary picks a sequence of vertices $u_1, u_2, \dots, u_K$ one by one. Every time a vertex u_i is revealed to the algorithm, we need to irrevocably match u_i to a vertex v_i not previously matched, and we wish to make K as large as possible.

Lemma 4.5. *For any integer parameters a, b, q such that $15 \leq b + 2 \leq a \leq q$, there is a deterministic construction of a bipartite graph $H = (A \cup B, I)$, $|A| = q^a, |B| = q^{b+2} \cdot \lceil (b + 2) \log_2 q \rceil$, such that the online perfect matching on H can be solved as long as the number of online vertices in A is at most $K \leq \frac{(2q-2)^{b+2}}{q \cdot a^{b+2}(b+2)^{b+2}}$. Plus, the algorithm only takes space $O(K)$.*

Proof. TOPROVE 11 □

4.2 Proof of Lemma 4.1

Let $\delta = \epsilon^2/10$ be a small constant. For convenience, let us assume $1/\delta$ is an integer. If $\Delta < \log^{20/\epsilon^2\delta} n$, then a $O(n \log^{200/\epsilon^4} n)$-space algorithm can store the whole input graph in memory and print a Δ-edge coloring in the output stream. For the rest, let us assume $\Delta \geq \log^{20/\epsilon^2\delta} n$.

4.2.1 Data Structures

We will reuse all the forest data structures defined previously in Section 3.1.1, so we will not describe them again here. However, the color package allocations on those forests will be different because previously we have used randomization for this part.

Deterministic Color Allocation on Forests. For any choice of the frequency vector $\mathbf{f} = (f_1, f_2, \dots, f_h)$, we will allocate color packages at each tree node of each forest $\mathcal{T}_\mathbf{f}$ in a deterministic manner. Take a prime number $q \in [\Delta^\delta, 2\Delta^\delta)$. By construction, each forest structure $\mathcal{T}_\mathbf{f}$ is a forest of $h + 1$ levels (from level-0 to level-h), and each tree is rooted at a level-h node. For each coordinate f_i of the frequency vector $\mathbf{f} = (f_1, f_2, \dots, f_h)$, define some parameters below:

$$\begin{aligned}
\lambda &= \lceil \log_2^{2+3/\delta} n \cdot (2 + 3/\delta)^{2+3/\delta} \rceil \\
b_0 &= \left\lceil \log_q(\lambda \cdot 2^{r+1}) \right\rceil \\
b_i &= \left\lceil \log_q(\lambda \cdot f_i) \right\rceil, \forall 1 \leq i \leq h \\
\lambda_i &= \lambda \cdot \lceil (b_i + 2) \cdot \log_2 q \rceil, \forall 1 \leq i \leq h
\end{aligned}$$

Here are some basic estimations of these parameters.

Lemma 4.6. *For any $0 \leq i \leq h$, we have $b_i \leq 3/\delta$.*

Proof. TOPROVE 12 □

Allocate an overall color package $\mathcal{C}^{\mathbf{f}}$ with $2^{l+1} \cdot q^{b_0+3} \cdot \prod_{i=1}^{h}(\lambda_i \cdot q^{b_i+2})$ new colors. Each color in this color package can be identified as a tuple $(c_0, c_1, \ldots, c_h)$ from the direct product space $[2^{l+1} \cdot q^{b_0+3}] \times [\lambda_1 \cdot q^{b_1+2}] \times \cdots \times [\lambda_h \cdot q^{b_h+2}]$. For any tuple $(c_1, c_2, \ldots, c_h)$, the collection of all colors $\{(*, c_1, c_2, \ldots, c_h)\}$ will be called a **palette**.

As we did in the randomized algorithm, we will specify a color package for each tree node. To do this, for any level $0 \leq i \leq h-1$, apply Lemma 4.5 which deterministically builds a bipartite graph $H_i^{\mathbf{f}} = (A_i^{\mathbf{f}} \cup B_i^{\mathbf{f}}, I_i^{\mathbf{f}})$ where $|A_i^{\mathbf{f}}| = q^{\lceil \log_q f_i \rceil}, |B_i^{\mathbf{f}}| = q^{b_{i+1}+2} \cdot \lceil (b_{i+1}+2) \cdot \log_2 q \rceil$.

Lemma 4.7. *Bipartite graph $H_i^{\mathbf{f}}$ admits an online perfect matching with f_{i+1} vertices in $A_i^{\mathbf{f}}$.*

Proof. TOPROVE 13 □

For any tree node N in $\mathcal{T}_{\mathbf{f}}$ on level-i for some $0 \leq i \leq h-1$, let P be the tree path which connects N and the root of $\mathcal{T}_{\mathbf{f}}$. List all the nodes of P as $N = N_i, N_{i+1}, \ldots, N_h$, and assume N_j is the k_j-th child of N_{j+1}. Then, define the color package at N to be:

$$\mathcal{C}^N = \{(*, *, \ldots, *, c_{i+1}, c_{i+2}, \ldots, c_h) \mid (k_j, c_{j+1}) \in I_j, \forall i \leq j < h\}$$

To justify this definition, since $1 \leq k_j \leq f_j \leq |A_j^{\mathbf{f}}|$ and $1 \leq c_{j+1} \leq \lambda_{j+1} \cdot q^{b_{j+1}+2} \leq |B_j^{\mathbf{f}}|$, we can encode k_j as a vertex in $A_j^{\mathbf{f}}$ and c_{j+1} as a vertex in $B_j^{\mathbf{f}}$.

Notice that, by definition, the same palette may appear at multiple leaf nodes (which represent input batches). For any leaf node N (or equivalently, a batch), let $\mathsf{cnt}(\mathcal{C}^N)$ count the total number of times that palette $\mathcal{C}^N$ was contained in the color packages of previous leaf nodes (batches). Note that these counters do not require extra space, because we can recompute them upon the arrival of any input batch.

Vertex-Wise Data Structures. As before, we will maintain some data structures for the vertices in V. For the vertices in L, we will not maintain the random shifts $\{r_u \in [3 \cdot 2^{r+1}] \mid u \in L\}$. Instead, we build a bipartite expander $H^{\mathbf{f}} = (A^{\mathbf{f}} \cup B^{\mathbf{f}}, I^{\mathbf{f}})$ using multiplicity codes according to Lemma 4.3 where $|A^{\mathbf{f}}| = q^{\lceil \log_q n \rceil}, |B^{\mathbf{f}}| = q^{b+2}, b = \lceil \log_q(\lambda \cdot 2^{r+1}) \rceil$; recall the definitions of q, λ from the previous paragraph.

For vertices $v \in R$, we will reuse the same data structures of marked nodes $M_{v,\mathbf{f}} \subseteq V(\mathcal{T}_{\mathbf{f}})$. In addition, for each marked node $N \in M_{v,\mathbf{f}}$ which is on level-i, we will store a length-h tuple

$$\overrightarrow{c_N} = (*, *, \ldots, *, c_{i+1}^N, c_{i+2}^N, \ldots, c_h^N)$$

to represent all the color palettes that were used at node N (the first i coordinates are $*$). Similar to Invariant 3.1, the marked sets will have the following requirements.

Invariant 4.1. *We will ensure the following properties with respect to the marked nodes $M_{v,\mathbf{f}}$ throughout the execution of the streaming algorithm.*

(1) All nodes in $M_{v,\mathbf{f}}$ are incomparable in forest $\mathcal{T}_{\mathbf{f}}$. Furthermore, suppose that the current input batch corresponds to a leaf F, and let P tree path from F to the tree root. Then, any node $N \in M_{v,\mathbf{f}}$ is a child of a node on the root-to-leaf path P.

(2) For any node $N \in \mathcal{T}_{\mathbf{f}}$ on level-k such that $M_{v,\mathbf{f}} \cap V(\mathcal{T}_{\mathbf{f}}(N)) \neq \emptyset$, let $F_1, F_2, \ldots, F_s \subseteq E$ be all the input batches which correspond to leaf nodes in subtree $\mathcal{T}_{\mathbf{f}}(N)$. Take the union of batches $U = F_1 \cup F_2 \cup \cdots \cup F_s$. Then, we have $\deg_U(v) \geq 2^{r-k} \cdot \prod_{i=1}^{k} f_i$.

(3) For any previous input batch F' before F such that:

- *F and F' are in the same connected component in $\mathcal{T}_{\mathbf{f}}$,*
- *$\deg_{F'}(v) \in [2^r, r^{r+1})$,*
- *v used some colors in $\mathcal{C}^{F'}$ during the algorithm,*

we guarantee that F' must be contained in some subtree $\mathcal{T}_{\mathbf{f}}(N)$ for some $N \in M_{v,\mathbf{f}}$. In addition, assume N is on level-i in $\mathcal{T}_{\mathbf{f}}$, then we require that all colors used by the edges in $F'_{l,r}$ incident on v share the suffix $\left(*, *, \ldots, *, c^N_{i+1}, c^N_{i+2}, \ldots c^N_h\right)$.

(4) For any pair of marked nodes $N_1, N_2 \in M_{v,\mathbf{f}}$, suppose their lowest ancestor is on level-k, then both tuples $\overrightarrow{c_{N_1}}, \overrightarrow{c_{N_2}}$ share the same suffix up to length $h - k$.

In addition, for all marked nodes $N_1, N_2, \ldots, N_i$ which shares the same parent W on level-k, suppose N_j is the k_j-th child of N. We will make sure that $i \leq f_k$, and all the pairs $\left(k_j, c^{N_j}_k\right), 1 \leq j \leq i$ form a perfect matching in the bipartite expander H_{k-1} which is chosen by the online perfect matching algorithm we described in Lemma 4.5.

4.2.2 Algorithm Description

Similar to the randomized algorithm in Section 3.1.2, the deterministic also consists of three steps for each input batch F: preprocessing marked sets, coloring $F_{l,r}$, and post-processing marked sets.

Preprocessing Marked Sets. Since the current input batch has changed due to the new arrival F, we have might violated Invariant 3.1(2) as the root-to-leaf tree path may have changed. Therefore, we first need to update all the marked sets as in the following procedure which is named DetUpdateMarkSet(F).

Go over every vertex $v \in R$ and every frequency vector $\mathbf{f}$. Consider the position of F in the forest $\mathcal{T}_{\mathbf{f}}$, and let P be the root-to-leaf path in $\mathcal{T}_{\mathbf{f}}$ ending at leaf F. First, remove all marked nodes $N \in M_{v,\mathbf{f}}$ which are not in the same tree as P; note that forest $\mathcal{T}_{\mathbf{f}}$ is actually a forest of trees, and this may happen when F is the first leaf in a new tree of $\mathcal{T}_{\mathbf{f}}$.

Next, go over every node W on lying on the tree path P. Assume W is on level-k. For any child node N of W, if (1) $V(\mathcal{T}_{\mathbf{f}}(N)) \not\ni F$ and (2) $V(\mathcal{T}_{\mathbf{f}}(N)) \cap M_{v,\mathbf{f}} \neq \emptyset$, then take an arbitrary node $U \in V(\mathcal{T}_{\mathbf{f}}(N)) \cap M_{v,\mathbf{f}}$, assign $\overrightarrow{c_W} = \left(*, *, \ldots, *, c^U_k, c^U_{k+1}, \ldots, c^U_h\right)$. After that, remove all nodes in $V(\mathcal{T}_{\mathbf{f}}(N)) \cap M_{v,\mathbf{f}}$ from $M_{v,\mathbf{f}}$ and add N to $M_{v,\mathbf{f}}$.

Coloring $F_{l,r}$. The procedure DetFreqVec(F) for selecting a frequency vector $\mathbf{f}_v$ for each $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$ is the same as Section 3.1.2 which is a deterministic subroutine. The different parts will be selection of the color palette $\mathcal{C}_v$ and color assignment.

- **Selecting $\mathcal{C}_v$.** Let W be the lowest ancestor of F such that $V(\mathcal{T}_{\mathbf{f}_v}(W)) \cap M_{v,\mathbf{f}_v} \neq \emptyset$. If such a node W does not exist, then $M_{v,\mathbf{f}_v}$ must be empty at the moment. In this case, set W to be the root of the tree containing F.

  In general, assume W is on level-k, and define

  $$F = W_0, W_1, \ldots, W_{k-1}, W_k = W$$

  to be the sub-path of P connecting W and F. Assume W_{k-1} is the s-th child of W.

  According to our selection of $\mathbf{f}_v$, W has less than f_k marked children. Assume these marked children of W are $N_1, N_2, \ldots, N_t, s < f_k$, and N_i is the s_i-th child of W. Under Invariant 4.1(4), all the pairs $\left(s_i, c^{N_i}_k\right)$ form a matching in bipartite expander $H^{\mathbf{f}_v}_{k-1}$ and was selected by the online perfect matching algorithm from Lemma 4.5. Since $t < f_k$, by Lemma 4.7, we can continue to apply Lemma 4.5 to find a value $c_k \notin \{c_k^{N_i} \mid 1 \leq i \leq t\}$ such that $(s, c_k) \in I_{k-1}^{\mathbf{f}_v} = E(H_{k-1}^{\mathbf{f}_v})$.

As for coordinates $c_{k+1}, c_{k+2}, \ldots, c_h$, assign $c_j = c_j^{N_1}, k < j \leq h$. To complete the coordinates of palette $\mathcal{C}_v$, we still need to specify $c_1, c_2, \ldots, c_{k-1}$. To do this, for each level $0 \leq i < k-1$, initialize an online perfect matching procedure on graph $H_i^{\mathbf{f}_v}$. Assume W_i is the t_i-th child of W_{i+1}, then view t_i as the first online vertex in $H_i^{\mathbf{f}_v}$ and match it using edge (t_i, c_i) in $H_i^{\mathbf{f}_v}$. In the end, set $\mathcal{C}_v = (c_1, c_2, \ldots, c_h)$ be the palette we will use.

- **Color assignment.** Next, we are going to color all edges in $F_{l,r}$ using colors from $\mathcal{C}_v$ for edges incident on $v \in R$. For any vertex $v \in R$, let $u_1, u_2, \ldots, u_k, k < 2^{r+1}$ be all the neighbors of v in $F_{l,r}$. Recall that $H^{\mathbf{f}_v} = (A^{\mathbf{f}_v} \cup B^{\mathbf{f}_v}, I^{\mathbf{f}_v})$ is a bipartite expander such that $|A^{\mathbf{f}_v}| \geq n$, we can interpret each vertex u_i as a vertex in $A^{\mathbf{f}_v}$. Then, since $k < 2^{r+1}$, plugging in $K = k$ in using Lemma 4.3, $H^{\mathbf{f}_v}$ is always a (D, k) expander where

$$\begin{aligned} D &\geq q - \frac{\log_q n \cdot (b_0 + 2)}{2} \cdot (qk)^{1/(b_0+2)} \\ &\geq q - 0.5 \cdot \lambda^{1/(b_0+2)} \cdot q \cdot \lambda^{1/(b_0+2)} = q/2 > 1 \end{aligned}$$

  Hence, we can find a perfect matching $\{(u_i, r_i) \mid 1 \leq i \leq k\}$ in $H^{\mathbf{f}_v}$, where $1 \leq r_i \leq q^{b_0+2}$; recall that $\lambda = \lceil \log^{2+3/\delta} n \cdot (2 + 3/\delta)^{2+3/\delta} \rceil$ and $b_0 = \lceil \log_q(\lambda \cdot 2^{r+1}) \rceil < 3/\delta$ by Lemma 4.6. Additionally, assume for each $1 \leq i \leq k$, edge (u_i, r_i) is the t_i-th edge in the adjacency list of u_i in graph $H^{\mathbf{f}_v}$; note that $1 \leq t_i \leq q$ by construction of multiplicity codes.

  Order all the edges in $F_{l,r}$ alphabetically. For each edge $(u_i, v) \in F_{l,r}$, assume it is the j-th edge around u_i. To encode the colors in $\mathcal{C}_v$, we interpret each color as a tuple $\in [q] \times [q^{b_0+2}] \times [2^{l+1}]$. Then, assign the κ_i-th color in $\mathcal{C}_v$ to (u_i, v) and print it in the output stream, where we set:

$$\kappa_i = (t_i, \mathsf{cnt}(\mathcal{C}_v) + r_i, j)$$

  As a minor issue, when $\mathsf{cnt}(\mathcal{C}_v) + r_i$ is larger than q^{b_0+2}, we take its modulo and replace it with $\mathsf{cnt}(\mathcal{C}_v) + r_i \mod q^{b_0+2}$.

Postprocessing Marked Sets. After processing the input batch F, for every vertex $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$, add node F to $M_{v,\mathbf{f}_v}$ with $\overrightarrow{c_F} = (c_1, c_2, \ldots, c_h)$. The whole algorithm is summarized as Algorithm 3.

4.2.3 Proof of Correctness

To begin with, let us first bound the total number of colors that we use throughout the algorithm.

Lemma 4.8. *The total number of colors used by the algorithm is at most*

$$O\left((\log \Delta)^{O(1/\epsilon)} \cdot (1/\epsilon)^{O(1/\epsilon^2)} \cdot \Delta^{1+2\epsilon} \cdot 2^l\right)$$

different colors.

Proof. TOPROVE 14 □

Lemma 4.9. *Each palette is used in at most 2^{r+1} different batches.*

Proof. TOPROVE 15 □

Corollary 4.1. *During the algorithm, the values of the counters $\mathsf{cnt}(\mathcal{C})$ never exceed 2^{r+1} for any palette $\mathcal{C}$.*

Algorithm 3: $\textsc{DetColorLowDeg}(F)$

```
   /* pre-processing marked sets */
1  for $v \in R$ and frequency vector $\mathbf{f}$ do
2      call $\textsc{DetUpdateMarkSet}(F)$ (as described in Section 4.2.2) to remove marked
         nodes in previous tree components in $\mathcal{T}_{\mathbf{f}}$,
3      and elevate the positions of all the marked nodes in $M_{v,\mathbf{f}}$ to their ancestors which
         are children of $P$,
4      and assign the tuples $\overrightarrow{c_*}$ properly for newly marked nodes;
   /* select frequency vector $\mathbf{f}_v$ and palette $\mathcal{C}_v$ */
5  for $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$ do
6      call $\textsc{DetFreqVec}(F)$ (as described in Section 4.2.2) to grow a vector
         $\mathbf{f}_q = (f_1, f_2, \ldots)$ progressively;
7      select palette $\mathcal{C}_v = (c_1, c_2, \ldots, c_h)$ using online perfect matching on bipartite
         expanders $H_0^{\mathbf{f}_v}, H_1^{\mathbf{f}_v}, \ldots, H_{h-1}^{\mathbf{f}_v}$;
   /* assign colors for $F_{l,r}$ */
8  for $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$ do
9      let $u_1, u_2, \ldots, u_k$ be all neighbors of $v$ in $F_{l,r}$;
10     find a perfect matching $\{(u_i, r_i) \mid 1 \leq i \leq k\}$ in $H^{\mathbf{f}_v}$;
11     assign the $\kappa_i$-th color in $\mathcal{C}_v$ to $(u_i, v)$, with $\kappa_i = (t_i, \mathsf{cnt}(\mathcal{C}_v) + r_i, j)$, assuming $(u_i, r_i)$
         is the $t_i$-th edge of $u_i$ in $H^{\mathbf{f}_v}$, and $(u_i, v)$ is the $j$-th edge around $u_i$ in $F_{l,r}$;
   /* post-processing marked sets */
12 for $v \in R$ such that $\deg_F(v) \in [2^r, 2^{r+1})$ do
13     add $F$ to $M_{v,\mathbf{f}_v}$ with $\overrightarrow{c_F} = (c_1, c_2, \ldots, c_h) = \mathcal{C}_v$;
```

As before, we can analyze the total space of the marked sets under the condition of Invariant 4.1(2). Since the proof would be the same, we omit it here.

Lemma 4.10. *If Invariant 4.1 is satisfied, then the total size $\sum_{v \in R} |M_{v,\mathbf{f}}|$ is bounded by $O(n/\epsilon)$ for any frequency vector $\mathbf{f}$.*

Now, let us verify that our algorithm preserves Invariant 4.1.

Lemma 4.11. *Invariant 4.1 is preserved by the algorithm throughout its execution.*

Proof. TOPROVE 16 □

Finally, let us verify the validity of the output.

Lemma 4.12. *In the output stream, the algorithm never prints the same color for two adjacent edges.*

Proof. TOPROVE 17 □

4.3 Proof of Lemma 4.2

4.3.1 Basic Notations

Define a constant parameter $\delta = \epsilon^2/10$. As before, we can assume Δ is a power of 2 without loss of generality. If $\Delta < \log^{10/\epsilon\delta} n$, then we will store the entire graph and output a Δ-edge coloring. For the rest, let us assume $\Delta > \log^{10/\epsilon\delta} n$.

Throughout the algorithm, each vertex $u \in L$ maintains a value $\mathsf{cnt}(u)$ that counts the number of input batches F where $\deg_F(u) \in [2^l, 2^{l+1})$; similarly, each vertex $v \in R$ maintains a value $\mathsf{cnt}(v)$ that counts the number of input batches F where $\deg_F(v) \in [2^r, 2^{r+1})$.

Take a prime number $q \in [\Delta^\delta, 2\Delta^\delta)$. Set parameters:

$$\lambda = \lceil \log_2^{2+3/\delta} n \cdot (2+3/\delta)^{2+3/\delta} \rceil$$

$$a_1 = \lceil \log_q(n\Delta/2^l) \rceil + 2, b_1 = \lceil \log_q(\lambda\Delta/2^l) \rceil$$

$$a_2 = \lceil \log_q(n\Delta/2^r) \rceil + 2, b_2 = \lceil \log_q(\lambda\Delta/2^r) \rceil$$

Apply Lemma 4.3 and construct two bipartite expanders based on multiplicity codes $H_1 = (A_1 \cup B_1, I_1), H_2 = (A_2 \cup B_2, I_2)$, where $A_i = \mathbb{F}_q^{a_i}, B_i = \mathbb{F}_q^{b_i+2}$.

Lemma 4.13. *For $i \in \{1, 2\}$, we have $b_i < 2/\delta$, and $\lambda \geq a_i^{b_i+2} \cdot (b_i+2)^{b_i+2}$.*

Proof. TOPROVE 18 □

Lemma 4.14. *If we have $O\left(\Delta^{1+\epsilon} + \Delta^{2+\epsilon}/2^{l+r}\right)$ colors, then we can encode each color as a tuple from $[q^{b_1+2}] \times [q^{b_2+2}] \times [\lceil 2^{l+r+2}/\Delta \rceil]$.*

Proof. TOPROVE 19 □

At any moment in time during the algorithm, for each $u \in L$, since $0 \leq \mathsf{cnt}(u) < \Delta/2^l$ and $|L| < n$, we can interpret $\mathsf{cnt}(u)$ as a polynomial $g_u(X) \in \mathbb{F}_q^{<\lceil \log_q \Delta/2^l \rceil}[X]$, and vertex u as a polynomial $f_u(X) \in \mathbb{F}_q^{<\lceil \log_q n \rceil}[X]$. Define polynomial $h_u(X) = g_u(X) + X^{\lceil \log_q \Delta/2^l \rceil} \cdot f_u(X)$.

Symmetrically, for each $v \in R$, we can interpret $\mathsf{cnt}(v)$ as a polynomial $g_v(X) \in \mathbb{F}_q^{\lceil \log_q \Delta/2^r \rceil}[X]$, and vertex v as a polynomial $f_v(X) \in \mathbb{F}_q^{<\lceil \log_q n \rceil}[X]$. Define polynomial $h_v(X) = g_v(X) + X^{\lceil \log_q \Delta/2^r \rceil} \cdot f_v(X)$. By definition, f_v does not change over time, and h_v, g_v are dependent on how many batches we have read so far.

4.3.2 Algorithm Description

Upon the arrival of an input batch F, let us discuss how to assign colors to edges in $F_{l,r}$. We are going to associate a pair $(s_e, t_e) \in [q^{b_1+2}] \times [q^{b_2+2}]$ for every edge $e \in F_{l,r}$ in the following way. Go over every vertex $u \in L$ and list all of its edges $(u, v_1), (u, v_2), \ldots, (u, v_k)$ in $F_{l,r}$. Since G is a simple graph, all the neighbors $v_1, v_2, \ldots, v_k$ are different.

Divide $\{v_1, v_2, \ldots, v_k\}$ into $\lceil 2^{l+r+2}/\Delta \rceil$ groups, each of size at most $\Delta/2^r$. For each of these group, say that it contains vertices $z_1, z_2, \ldots, z_K, K \leq \Delta/2^r$. Interpret each polynomial h_{z_i} as a vertex in A_2. According to Lemma 4.3, H_2 is a (D, K)-expander where:

$$D = q - \frac{a_2(b_2+2)}{2} \cdot (qK)^{1/(b_2+2)} \geq q - 0.5 \cdot \lambda^{1/(b_2+2)} \cdot q \cdot \lambda^{1/(b_2+2)} = q/2 > 1$$

Therefore, there exists a bipartite matching $\{(h_{z_i}, t_i) \mid 1 \leq i \leq K\}$ of H_2. Since $t_i \in \mathbb{F}_q^{b_2+2}$, we can interpret t_i as an integer $t_{(u,z_i)} \in [q^{b_2+2}]$.

Symmetrically, for each edge $e = (u, v)$ we can also define the value s_e for every edge $e \in F_{l,r}$. Next, for any integer pair $(s, t) \in [q^{b_1+2}] \times [q^{b_2+2}]$, define $F_{l,r}^{(s,t)} = \{e \in F_{l,r} \mid (s_e, t_e) = (s, t)\}$.

Lemma 4.15. *For any (s, t), the maximum degree of edge set $F_{l,r}^{(s,t)}$ is at most $\lceil 2^{l+r+2}/\Delta \rceil$.*

Proof. TOPROVE 20 □

By the above statement, we can use the set of colors $\{(s, t, c) \mid c \in [\lceil 2^{l+r+2}/\Delta \rceil]\}$ to color all edges in $F_{l,r}^{(s,t)}$. After the coloring is completed for all pairs (s, t), we print all the colors of $F_{l,r}$ in the output stream. The whole algorithm is summarized in Algorithm 4.

Algorithm 4: DetColorHighDeg(F)

```
1 compute polynomials $h_u(X), \forall u \in V$ according to $\mathsf{cnt}(u)$;
2 define $E_{x,y} = \{(u,v) \in F_{l,r} \mid (x_u, y_u) = (x,y)\}, \forall (x,y)$;
3 for $u \in L$ such that $\deg_F(u) \in [2^l, 2^{l+1})$ do
4     let $v_1, v_2, \ldots, v_k$ be all of $u$'s neighbor in $F_{l,r}$;
5     compute integers $t_{(u,v_i)}, 1 \le i \le k$ using bipartite matchings;
6 compute integers $s_e, \forall e \in F_{l,r}$ by a symmetric manner;
7 compute $F_{l,r}^{(s,t)} = \{e \in F_{l,r} \mid (s_e, t_e) = (s,t)\}$ for all $(s,t) \in [q^{b_1+2}] \times [q^{b_2+2}]$ and color
    each $F_{l,r}^{(s,t)}$ using $\lceil 2^{l+r+2}/\Delta \rceil$ colors;
8 increment counters $\mathsf{cnt}(*)$;
```

4.3.3 Proof of Correctness

It suffices to show that the algorithm always produces a valid edge coloring of G.

Lemma 4.16. *In the output stream, the algorithm never prints the same color for two adjacent edges.*

Proof. TOPROVE 21 □

4.3.4 Extension to Multi-graphs

So far we have only considered simple graphs. Let us briefly discuss how to extend our algorithm to multi-graphs. Basically, in multi-graphs it could be the case that $\Delta \gg n$, and so storing the forest structures and bipartite expanders would be too costly. Technically speaking, There are mainly three places where we needed the input graph to be simple.

(1) The size of the forest structure is at least Δ;

(2) Finding perfect matchings on Line 10 in Algorithm 3 and on Line 5 in Algorithm 4 requires explicitly storing some bipartite expanders of size larger than Δ;

(3) On Line 4 of Algorithm 4 we need all neighbors of u to be distinct.

Let us discuss how to bypass these three issues.

For issue (1), the main observation is that we can store the entire forest structures implicitly, since the color package allocations are defined by multiplicity codes in a closed-form; note that we could do this with the randomized algorithm, since we had to remember all the randomness in order to store the color package allocations.

For issue (2), instead of performing a standard perfect matching algorithm on the bipartite expander, we could apply the greedy matching algorithm on the bipartite expander from Lemma 4.5, which only takes space proportional to the matching size.

For issue (3), note that we only require that the subgraph (V, F) is simple, not necessarily the whole input graph G. Therefore, it suffices to use a reduction from a general input stream to input streams of batches which are simple subgraphs, which is presented below.

Multi- to Simple Reduction. For the above technical reason, we need the assumption that every input batch is a simple subgraph. We argue that this extra condition does not make the problem simpler.

Lemma 4.17. *Given an algorithm $\mathcal{A}$ for coloring graphs using $f(\Delta)$ colors and $g(n,\Delta)$ space under the condition that every input batch is a simple subgraph of G, there is an algorithm $\mathcal{B}$ using $O(f(\Delta)\log\Delta)$ colors and $O(g(n,\Delta)\log\Delta)$ space for coloring any graph streams. Furthermore, this reduction is deterministic.*

Proof. TOPROVE 22 □

Acknowledgment

Shiri Chehcik is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803118 UncertainENV). Tianyi Zhang is financially supported by the starting grant "A New Paradigm for Flow and Cut Algorithms" (no. TMSGI2_218022) of the Swiss National Science Foundation.

References

[ABB+24] Sepehr Assadi, Soheil Behnezhad, Sayan Bhattacharya, Martín Costa, Shay Solomon, and Tianyi Zhang. Vizing's theorem in near-linear time. *arXiv preprint arXiv:2410.05240*, 2024.

[Arj82] Eshrat Arjomandi. An efficient algorithm for colouring the edges of a graph with $\Delta + 1$ colours. *INFOR: Information Systems and Operational Research*, 20(2):82–101, 1982.

[Ass25] Sepehr Assadi. Faster Vizing and Near-Vizing Edge Coloring Algorithms. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025.

[ASZZ22] Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh. Simple streaming algorithms for edge coloring. In *30th Annual European Symposium on Algorithms (ESA 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2022.

[BBKO22] Alkida Balliu, Sebastian Brandt, Fabian Kuhn, and Dennis Olivetti. Distributed edge coloring in time polylogarithmic in Δ. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 15–25, 2022.

[BCC+24] Sayan Bhattacharya, Din Carmon, Martín Costa, Shay Solomon, and Tianyi Zhang. Faster $(\Delta + 1)$-Edge Coloring: Breaking the $m\sqrt{n}$ Time Barrier. In *65th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2024.

[BCHN18] Sayan Bhattacharya, Deeparnab Chakrabarty, Monika Henzinger, and Danupon Nanongkai. Dynamic Algorithms for Graph Coloring. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–20. SIAM, 2018.

[BCPS24] Sayan Bhattacharya, Martín Costa, Nadav Panski, and Shay Solomon. Nibbling at Long Cycles: Dynamic (and Static) Edge Coloring in Optimal Time. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2024.

[BCSZ25] Sayan Bhattacharya, Martín Costa, Shay Solomon, and Tianyi Zhang. Even Faster $(\Delta+1)$-Edge Coloring via Shorter Multi-Step Vizing Chains. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025. (to appear).

[BDH+19] 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)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[Ber22] Anton Bernshteyn. A fast distributed algorithm for $(\Delta+1)$-edge-coloring. *J. Comb. Theory, Ser. B*, 152:319–352, 2022.

[BGW21] Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of theACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2830–2842. SIAM, 2021.

[BM17] Leonid Barenboim and Tzalik Maimon. Fully-dynamic graph algorithms with sublinear time inspired by distributed computing. In *International Conference on Computational Science (ICCS)*, volume 108 of *Procedia Computer Science*, pages 89–98. Elsevier, 2017.

[BSVW24] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Online edge coloring is (nearly) as easy as offline. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 2024.

[BSVW25] Joakim Blikstad, Ola Svensson, Radu Vintan, and David Wajc. Deterministic Online Bipartite Edge Coloring. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025.

[CHL$^+$20] Yi-Jun Chang, Qizheng He, Wenzheng Li, Seth Pettie, and Jara Uitto. Distributed Edge Coloring and a Special Case of the Constructive Lovász Local Lemma. *ACM Trans. Algorithms*, 16(1):8:1–8:51, 2020.

[Chr23] Aleksander Bjørn Grodt Christiansen. The Power of Multi-step Vizing Chains. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1013–1026. ACM, 2023.

[Chr24] Aleksander B. G. Christiansen. Deterministic dynamic edge-colouring. *CoRR*, abs/2402.13139, 2024.

[CL21] Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the w-streaming model. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 181–183. SIAM, 2021.

[CMZ24] Shiri Chechik, Doron Mukhtar, and Tianyi Zhang. Streaming Edge Coloring with Subquadratic Palette Size. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*, volume 297 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:12, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[COS01] Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in o (e logd) time. *Combinatorica*, 21(1):5–12, 2001.

[CPW19] 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)*, pages 1–25. IEEE Computer Society, 2019.

[Dav23] Peter Davies. Improved distributed algorithms for the lovász local lemma and edge coloring. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4273–4295. SIAM, 2023.

[DFR09] Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. *ACM Transactions on Algorithms (TALG)*, 6(1):1–17, 2009.

[DGS25] Aditi Dudeja, Rashmika Goswami, and Michael Saks. Randomized Greedy Online Edge Coloring Succeeds for Dense and Randomly-Ordered Graphs. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2025.

[DHZ19] Ran Duan, Haoqing He, and Tianyi Zhang. Dynamic edge coloring with improved approximation. In *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019.

[EPS14] Michael Elkin, Seth Pettie, and Hsin-Hao Su. $(2\Delta - 1)$-Edge-Coloring is Much Easier than Maximal Matching in the Distributed Setting. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 355–370. SIAM, 2014.

[FGK17] Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. Deterministic distributed edge-coloring via hypergraph maximal matching. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 180–191. IEEE, 2017.

[GKMU18] Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. Deterministic distributed edge-coloring with fewer colors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 418–430, 2018.

[GNK+85] Harold N Gabow, Takao Nishizeki, Oded Kariv, Daneil Leven, and Osamu Terada. Algorithms for edge coloring. *Technical Rport*, 1985.

[GS24] Prantar Ghosh and Manuel Stoeckl. Low-memory algorithms for online edge coloring. In *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[GSS17] Christian Glazik, Jan Schiemann, and Anand Srivastav. Finding euler tours in one pass in the w-streaming model with o (n log (n)) ram. *arXiv preprint arXiv:1710.04091*, 2017.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from parvaresh–vardy codes. *Journal of the ACM (JACM)*, 56(4):1–34, 2009.

[KLS+22] Janardhan Kulkarni, Yang P. Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 104–116. ACM, 2022.

[KTS22] Itay Kalev and Amnon Ta-Shma. Unbalanced expanders from multiplicity codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2022)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022.

[PR01] Alessandro Panconesi and Romeo Rizzi. Some simple distributed algorithms for sparse networks. *Distributed computing*, 14(2):97–100, 2001.

[SB24] Mohammad Saneian and Soheil Behnezhad. Streaming edge coloring with asymptotically optimal colors. In *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 121:1–121:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[Sha49] Claude E Shannon. A theorem on coloring the lines of a network. *Journal of Mathematics and Physics*, 28(1-4):148–152, 1949.

[Sin19] Corwin Sinnamon. Fast and simple edge-coloring algorithms. *arXiv preprint arXiv:1907.03201*, 2019.

[SW21] 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)*, volume 198 of *LIPIcs*, pages 109:1–109:18, 2021.

[TSUZ01] Amnon Ta-Shma, Christopher Umans, and David Zuckerman. Loss-less condensers, unbalanced expanders, and extractors. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 143–152, 2001.

[Viz65] Vadim G Vizing. The chromatic class of a multigraph. *Cybernetics*, 1(3):32–41, 1965.