ON ADAPTIVITY AND RANDOMNESS FOR STREAMING ALGORITHMS

A Thesis
Submitted to the Faculty
in partial fullfillment of the requirements for the
degree of

Doctor of Philosophy

in

Computer Science

by Manuel Stoeckl

Guarini School of Graduate and Advanced Studies
Dartmouth College
Hanover, New Hampshire

April 2024

Abstract

A streaming algorithm has a limited amount of memory and reads a long sequence (data stream) of input elements, one by one, and computes an output depending on the input. Such algorithms may be used in an online fashion, producing a sequence of intermediate outputs corresponding to the prefixes of the data stream. Adversarially robust streaming algorithms are required to give correct outputs with a desired probability even when the data stream is adaptively generated by an adversary that can see all intermediate outputs of the algorithm. This thesis binds together research on a variety of problems related to the adversarial setting and other models for streaming algorithms.

- A toy problem in streaming called "Missing Item Finding" is studied in a variety of models, including: classical, adversarially robust, white-box adversarially robust, pseudo-deterministic, and deterministic streaming. Surprisingly, we find that for a wide range of problem parameters, adversarially robust algorithms for Missing Item Finding require access to a random oracle to work efficiently, requiring exponentially more space otherwise.

- We find lower and upper bounds for adversarially robust algorithms using semi-streaming space, which solve the task of maintaining an $O(\Delta^c)$-vertex-coloring of a graph edge insertion stream, for all constants $c > 1$. These give separations relative to classical and deterministic streaming. We also give a deterministic multi-pass algorithm for $(\Delta + 1)$-vertex-coloring.

- We obtain streaming algorithms for online edge coloring of graph edge insertion and vertex insertion streams that use sublinear space in the $O(\Delta)$-color regime; our algorithms can handle multi-graph streams, can be made deterministic, and give smooth space/color tradeoffs.

Acknowledgements

This work is based on, and includes text, equations, and figures from:

- Chapter 3: [Sto23] and [CS23]
- Chapter 4: [CGS22] and [ACGS23]
- Chapter 5: [GS23]
- Chapter 2: all of the above

My coauthors in the above are Amit Chakrabarti, Prantar Ghosh, and Sepehr Assadi.

These works were supported by:

- NSF awards CCF-1907738 and CCF-2006589.
- (S. Assadi): NSF CAREER Grant CCF-2047061, Google Research gift, Fulcrum award from Rutgers Research Council.
- (P. Ghosh) Grant 820931 to DIMACS from the Simons Foundation.

I would like to thank my advisor, Amit Chakrabarti, and my coauthors. I also wish to acknowledge those with whom I have discussed the research leading to this thesis: Peter Winkler, Deeparnab Chakrabarty, Hsien-Chih Chang, Sebastiaan Joosten, Maryam Neghabani, Hang Liao, Ankita Sarkar, and Jonathan Conroy.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Imagine that you and an opponent are playing a game, running for 2000 turns; in each turn, your opponent tells you a number between 1 and 10^6, and you try to reply with a number between 1 and 10^6 that was not said by your opponent, on this or on any previous turn. You win if you can do this on every turn, and lose if you ever reply with a number the opponent had said before. Whether or not you can reliably win this game depends on what resources you have available.[1] If you could write down all the numbers that were stated, it would be easy to pick a new number each turn. But if you cannot write anything down, how often you can win this game depends on how much about your opponent's sequence of numbers you can remember—and on what else you have available.

A simple strategy for this game is to say a random number between 1 and 10^6 each turn; on one hand, you don't need to remember anything to do this; on the other, you are only guaranteed to win the game a bit less than 1/8th of the time. You can win much more often, however, if you have with you a piece of paper listing a few thousand randomly chosen numbers between 1 and 10^6, that were chosen in advance, and which your opponent does not know. Then you just need to say *the first number in the list that the opponent has not said*; doing this requires just remembering what the last number you said is, plus a few numbers from the remainder of the list that your opponent managed to guess. As Chapter 3 will show, this strategy is close being optimal; and having a random list of numbers is often *necessary*, if you do not want to keep track of a large amount of information. Furthermore, if you want to win the game with certainty, there *is* a strategy in which you don't need to remember exactly which numbers your opponent said, but it still requires an impractical amount of memory.

[1]And of course, on what resources the opponent has available; in any case, to be certain of victory you should use a strategy that works against someone with perfect memory and infinite time, who already knows what your strategy is.

Streaming perspective. Memory-limited strategies for the two player game we described (which we call MissingItemFinding) can be studied through the framework of streaming algorithms. A "streaming algorithm" is an algorithm that reads a sequence of values in order, operating on one value at a time, and computes some value associated to the sequence. For many problems, streaming algorithms exist which require much less computer memory than would be needed to store the entire sequence (the "data stream"). Among many other uses, such algorithms have historically been useful to compactly estimate properties[2] of distributions from a large sequence of samples. The study of streaming algorithms has been intertwined with the study of "sketches", summaries of sequences of data which satisfy an additional mergeability property [Cor23]. The focus of research on streaming algorithms has changed several times over the past few decades: for example, the constraints of tape drives have led to work on "multi-pass" streaming algorithms that read a sequence of values multiple times; a wave of interest in big data has motivated work on estimating graph properties, including in streaming; and the use of streaming algorithms in interactive environments has motivated research in making more "robust" streaming algorithms.

A powerful tool for streaming algorithms is randomization. Letting streaming algorithms make random decisions can significantly reduce the space required by the algorithm; however, this randomization often comes at the cost of having an algorithm fail with some small probability. For example, there is a randomized streaming algorithm using $O(\log m)$ space that with $\leq 1/\operatorname{poly}(m)$ error probability detects whether a sequence of m letters is a palindrome; but any deterministic algorithm must store the first $\lfloor m/2 \rfloor$ letters.

We will devise and prove lower bounds for streaming algorithms in a number of models that fall between plain "randomized" and "deterministic" streaming.

In this dissertation, we describe three main projects: we consider a toy problem under a variety of streaming model variants; study graph coloring in the adversarially robust and multi-pass deterministic streaming models, and find streaming algorithms for online edge coloring. In more detail:

Missing Item Finding. A recent paper [BJWY20] introduced the notion of an "adversarially robust" streaming algorithm. In the usual "static" setting, a streaming algorithm is considered to solve a problem with error $\leq \delta$ if for all possible input streams, the algorithm produces a correct output for the problem at the end of the stream, with probability $\geq 1 - \delta$ over the randomness of the algorithm. In the "adversarial setting", an algorithm solves a problem with error $\leq \delta$ if it produces correct output at each point in the stream, with total error probability $\leq \delta$, when the stream is produced by an adversary that chooses the next values based on the history of the algorithm's outputs. An algorithm is "adversarially robust" if it solves a problem in the adversarial

[2]For example, to estimate distribution quantiles, find the most common values, or calculate the moments of the vector giving the frequencies of the elements in the stream.

setting; it is “classic” if it is randomized and solves a problem in the static setting. (More formal definitions are given in Section 2.2.)

The adversarial setting is useful to model worst-case performance in scenarios where streaming algorithms periodically report outputs, and the outputs may influence future inputs to the algorithm. For example, if one connects a streaming algorithm to estimate which parts of a machine need cleaning to a control that can clean them, it is possible that any initial biases of the streaming algorithm (to, say, be insensitive to sensors at a few random locations), will be amplified over time (moving dirt to locations the streaming algorithm cannot detect). There have been a number of papers exploring problems in this model which we will detail in Chapter 2.

However, many aspects of adversarially robust streaming algorithms are not yet well understood. For example, how large of a separation in the space complexity, for a given problem, between randomized algorithms in the static and adversarial settings could there be? [KMNS21] gave a rather complicated example for which algorithms in the adversarial setting use exponentially more space than algorithms in the static setting, but we find that there is a much simpler one.

In the Missing Item Finding (MIF) problem (see Problem 3.1.1), parameterized by integers $1 \leq \ell < n$, one is given a stream considering of ℓ integers, $a_1, \ldots, a_\ell$, which are not-necessarily-distinct elements of $\{1, \ldots, n\}$. The goal is, after receiving each integer, to output an integer in $\{1, \ldots, n\}$ which was not part of the stream so far.

We find that:

- The space complexity of randomized algorithms in the static setting with error $\leq 1/\operatorname{poly}(n)$ is $O(\operatorname{polylog} n)$;
- The space complexity of randomized algorithms in the adversarial setting with error $\leq 1/\operatorname{poly}(n)$ is between $\Omega(1 + \ell^2/n)$ and $O((1 + \ell^2/n) \operatorname{polylog}(n))$;
- The space complexity of deterministic algorithms in the static setting[3] is $\Omega(\ell/\log(n/\ell))$.

In particular, for $\ell = n/2$, we obtain an exponential separation between the space complexity of randomized algorithms in the classic and adversarial settings.

Our claim that randomized algorithms in the adversarial setting use $O((1 + \ell^2/n) \operatorname{polylog}(n))$ space has a caveat: the only algorithm we have found that does this assumes access to a random oracle (needs to be able to read from random string of length $O(\ell \log n)$, at any time). Note that $\ell \log n$ is exponentially larger than $O((1 + \ell^2/n) \operatorname{polylog}(n))$ when $\ell = \sqrt{n}$. Having a random oracle is not an unreasonable assumption under standard cryptographic assumptions (the existence of one-way functions and hence of cryptographic pseudo-random generators). We investigate whether or not this is necessary, and find that it is: when $\ell = \sqrt{n}$, streaming algorithms for MIF in the

[3]As the outputs of deterministic algorithms must always be correct and are perfectly predictable, there is no different between static and adversarial settings for them.

adversarial setting, that do not have a random oracle, and can only use randomness in the usual way (by flipping a fair random coin or hardware random generator) require $n^{\Omega(1)}$ bits of space.

We also evaluate the space complexity of Missing Item Finding in a number of other models, including:

- randomized algorithms in the white-box adversarial [ABJ$^+$22] setting,
- randomized algorithms which are pseudo-deterministic [GGMW20],
- randomized algorithms in the adversarial setting, with the additional constraint that they can only make random choices before starting to receive the stream.

Our full results here are described in Chapter 3.

Graph vertex coloring. A graph coloring is an assignment of colors to the vertices of a graph so that no two adjacent vertices receive the same color. It is NP-hard to compute a graph coloring for a general graph using a minimum number of colors. If a graph has maximum degree Δ, then a greedy algorithm will efficiently find a coloring of it using at most $\Delta + 1$ colors. In fact, there is a greedy algorithm for the harder task of $(\deg +1)$-list coloring, wherein each vertex v has an associated set of colors L_v of size $\deg(v) + 1$, and one must assign to v a color from L_v. There are also polynomial time algorithms to find a coloring of a graph using only Δ colors, assuming there is one.

One way to frame the graph coloring problem for streaming algorithms is as follows. Say as input one is given a "graph edge insertion stream" – a sequence of edges in a graph over a fixed and known set of vertices, with the promise that the maximum degree of any vertex will be $\leq \Delta$. The objective is to output a coloring of the graph which is formed by the edges in the stream; where colors are taken from a finite set. In the static setting, using "semi-streaming" space (that is, $O(n \operatorname{polylog}(n))$ bits of space, where n is the number of vertices), a line of work leading up to [ACK19] proved that there exists a randomized streaming algorithm using $\Delta + 1$ colors.[4]

However, the algorithm of [ACK19] does not work in the adversarial setting: it will break when faced with an adversary that periodically evaluates the algorithm to get a coloring of the graph formed by the stream so far, and submits new edges between vertices that the algorithm gave the same color to. We sought to determine the space complexity of adversarially robust algorithms for graph coloring, and found:

- A space lower bound for any number of colors: for any constant $c \geq 1$, when $\Delta = \Omega(\log n)$ and $\Delta = O(n^{1/c})$, coloring with $O(\Delta^c)$ colors requires $\Omega(n\Delta^{1-c})$ space

[4]Later work by [AKM22] found a way to use only Δ colors, if the graph encoded by the stream has a Δ-coloring.

- An algorithm which for any $c \in [1, 2.5]$ uses $O(\Delta^c)$ colors and $O(n\Delta^{(c-1)/1.5} \operatorname{polylog}(n))$ space, with the caveat of requiring a random oracle.
- An algorithm which for any $c \in [1, 3]$ uses $O(\Delta^c)$ colors and $O(n\Delta^{(c-1)/2} \operatorname{polylog}(n))$ space, and does not need a random oracle.

For deterministic graph coloring algorithms, [ACS22] proved a lower bound on the number of colors that would be needed in the worst case, for any given amount of space. With light modifications to the paper (see Corollary 4.7.1), one finds that any algorithm using $\leq n/2$ colors when $\Delta = \Omega((\log n)^2)$ needs $\Omega(n\Delta/\operatorname{polylog}(n))$ bits of space. Implementing a naive greedy algorithm in the streaming setting guarantees a $(\Delta + 1)$ coloring, using only $O(n\Delta)$ bits of space, so their result shows that deterministic algorithms cannot save very much space without using far too many colors. In conjunction with our results, this implies a three-way separation between the number of colors used by the best randomized classic, adversarially robust, and deterministic algorithms using semi-streaming space.

[ACS22] also obtain a multi-pass deterministic algorithm in semi-streaming space for $O(\Delta)$ coloring using $O(\log \Delta)$ passes, leaving open whether a $(\Delta+1)$ coloring algorithm using few passes exists. We find that there is in fact a multi-pass, deterministic, semi-streaming $(\Delta + 1)$-coloring algorithm using $O(\log \Delta \log \log \Delta)$ passes over the input, and that this generalizes to a specific streaming form of the degree+1 list coloring problem.

Our full results here are described in Chapter 4.

Graph edge coloring. A somewhat less common problem than graph *vertex* coloring is graph *edge* coloring. Here, instead of assigning colors to the vertices of a graph, one assigns colors to the edges of a graph, so that no two edges incident on the same vertex have the same color. By Vizing's theorem [Viz65] on simple graphs of maximum degree Δ, this can be done using $\Delta + 1$ colors; while on multigraphs of maximum degree Δ, $3\Delta/2$ colors suffice. As a vertex of degree Δ will necessarily have Δ differently colored edges next to it, the optimum number of colors is never much lower than these upper bounds.

One active direction of research is to find "online" algorithms for edge coloring, using as few colors as possible. If the edges of a graph are provided in some order, an online edge coloring algorithm will process and assign a color to each edge *before* seeing any of the following edges; colors are never changed once assigned. [BMN92] find that, when the maximum degree of the graph is small ($\Delta = O(\sqrt{\log n})$) compared to n, even randomized online algorithms cannot do better than the simple greedy algorithm here, but leave open the best number of colors for larger values of Δ. While there have been major improvements over the last few years, an asymptotically tight bound is still open for this and a few variations of the problem.

In a fusion of the online and streaming models, we consider the problem of "streaming online edge coloring": finding algorithms that use $o(n\Delta)$ space to compute an online edge coloring of a stream of edges encoding a graph of maximum degree Δ. This builds off some work by [BDH+19, CL21, ASZZ22] in a "W-streaming" model for edge coloring that, compared to streaming online edge coloring, allows the algorithm to delay the reporting of edge colors.

We obtain a number of results, finding new algorithms for streaming edge coloring that significantly reduce the amount of space needed, at the cost of increasing the number of colors used by large constant factors or more. We consider both the "edge-arrival" model (where the algorithm receives and picks colors for a single edge at a time) and the "vertex-arrival" model (where the algorithm receives and picks colors for *all* the edges between a vertex and earlier-arriving vertices, in one batch). We find randomized algorithms and, while they use more colors and need exponential preprocessing time, deterministic algorithms. Among our results are:

- A general mechanism to trade space used for the number of colors, which can be applied to algorithms which edge-color multigraphs;
- A deterministic algorithm to edge-color multigraphs presented as vertex-arrival streams, using $O(\Delta)$ colors and $O(n \operatorname{polylog}(n, \Delta))$ space;
- A deterministic algorithm to edge-color multigraphs presented as vertex-arrival streams, using $O(\Delta(\log \Delta)^2)$ colors and $O(n \operatorname{polylog}(n, \Delta))$ space.

We also prove that deterministic edge-coloring algorithms for both vertex- and edge-arrival streams, which use $\leq (2-\varepsilon)\Delta$ colors, for some $\varepsilon > 0$ and $\Delta/n \ll 1$, must use $\Omega(\varepsilon^3 n)$ space.

Our full results here are described in Chapter 5.

Conclusion. All three problems: MissingItemFinding, graph coloring, and edge coloring, require one to output values from a set that shrinks as the stream progresses. While there will be common themes and techniques between these chapters, the details of the problems differ enough that we find no simple message for all three cases. That being said, if we suddenly had to design a streaming algorithm to color CW-complexes or maintain a safe spot in a collapsing art gallery:

- For an algorithm, we could pick outputs from a random sequence of values, and skip over those which don't work.
- For lower bounds, we would first consider a counting argument: what is an upper bound on the fraction of inputs that can be compatible with a given state of the algorithm?

See the ends of individual chapters for specific observations and open problems.

Chapter 2

Definitions

2.1 Basic definitions and notation

Notation. The following notation applies throughout this thesis. Individual chapters and sections may introduce more.

The set $[n] := \{1, 2, \ldots, n\}$, and $\binom{A}{k}$ is the set of k-sized subsets of A. $\log(x)$ is the base-2 logarithm, while $\ln(x)$ is the base-e logarithm. We define:

$$\mathbb{1}_B := \begin{cases} 1 & \text{if } B \text{ holds} \\ 0 & \text{if it does not} \end{cases}.$$

The notation $\vec{1}$ gives an all-1s vector. X^* is the set of lists of elements of X, including the empty list. $\mathbb{F}_q$ is the finite field with q elements, for q a prime or power of a prime, and $\mathbb{Z}_t$ is the ring of integers $\bmod t$. For a set B, $\textsc{sort}(B)$ is the vector containing the elements of B in ascending order. For a vector v, and set A, $A \subseteq v$ means that every element of A is an entry of v. If $\mathcal{D}$ is a distribution, then $A \sim \mathcal{D}$ means that A is randomly sampled according to the distribution $\mathcal{D}$. If S is a set, then $A \in_R S$ means that A is chosen uniformly at random from S. $\mathbb{E}[X]$ is the expected value of random variable X. For a set T, $\triangle[T]$ is the space of distributions over T.

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 $b_1, \ldots, b_k \in B$,

$$\Pr_{h \in_R \mathcal{H}} \left[h(a_1) = b_1 \wedge \cdots \wedge h(a_k) = b_k\right] = 1/|B|^k .$$

The family is 2-*universal* if, for all distinct $a_1, a_2 \in A$,

$$\Pr_{h \in_R \mathcal{H}} \left[h(a_1) = h(a_2)\right] \leq 1/|B| .$$

One-Way Communication Complexity. In Chapters 3 and 4, we shall consider a special kind of two-player communication game: one where all input belongs to the speaking player Alice, whose goal is to induce Bob to produce a suitable output.[1] Such a game, g, is given by a relation $g \in \mathcal{X} \times \mathcal{Z}$, where $\mathcal{X}$ is the input domain and $\mathcal{Z}$ is the output domain. In a protocol Π for g, Alice and Bob share a random string R. Alice is given $x \in \mathcal{X}$ and sends Bob a message $\text{msg}(x, R)$. Bob uses this to compute an output $z = \text{out}(\text{msg}(x, R))$. We say that Π solves g to error δ if $\forall\, x \in \mathcal{X} :\ \Pr_R[(x, z) \in g] \geq 1 - \delta$. The communication cost of Π is $\text{cost}(\Pi) := \max_{x,R} \text{length}(\text{msg}(x, R))$. The (one-way, randomized, public-coin) δ-error communication complexity of g is $\text{R}^{\to}_{\delta}(g) := \min\{\text{cost}(\Pi) : \Pi \text{ solves } g \text{ to error } \delta\}$.

If Π does not depend on R, it is deterministic. Minimizing over zero-error deterministic protocols gives us the one-way deterministic communication complexity of g, denoted $\text{D}^{\to}(g)$.

2.2 Streaming algorithms and models

A streaming algorithm is one which processes a long data stream $x = (a_1, \ldots, a_m)$ of input items, and computes a function of the data stream, using significantly less space than would be needed to store the stream (or, if the stream is of updates to some object, significantly less space than is needed to store the object.) While usually one would like an algorithm to operate efficiently – perhaps in polylogarithmic time per item – the space usage of the algorithm is typically the initial target of research.

2.2.1 Types of randomness

In many cases, one can prove that a deterministic streaming algorithm can not do much better than just storing its input. If one is willing to accept a chance of error in return, one can obtain algorithms requiring significantly less space, using random decisions to discard information that is unlikely to be useful. In this subsection, we classify a few common ways in which a streaming algorithm might use randomness. While the "stronger" types of randomness may provide better performance guarantees, in exchange they require more programming and operational effort to use.[2]

We view streaming algorithms as generalizations of finite state machines. An algorithm $\mathcal{A}$ has a finite set of states Σ, a finite input set $\mathcal{I}$, and a finite output set $\mathcal{O}$. The algorithm also has a

[1] Strictly speaking, this is an encoding-decoding problem.

[2] For example, to make random decisions a computer ultimately needs some source of randomness: but many micro-controllers may not have one. It is often much easier to verify that a deterministic algorithm is correct than to check that a randomized algorithm is correct with the desired probability. Randomized algorithms which can fail may require careful balancing between cost and reliability: an error level appropriate for a few runs will not be when an algorithm is run trillions of times. Finally, a deterministic algorithm is helpful when observability is limited: one can be certain that an unexplained failure in a system was not caused by the algorithm, and instead evaluate other rare failure modes.

transition function $\tau : \Sigma \times \mathcal{I} \times \mathcal{R} \to \Sigma$ indicating the state to switch to after receiving an input; how the third parameter (in $\mathcal{R}$) is used depends on the type of randomness. There are four cases:

Deterministic. The initial state of the algorithm is fixed, and τ is deterministic (does not depend on the third parameter). Each state has a unique output in $\mathcal{O}$ associated with it.

Random seed. The initial state of the algorithm is drawn randomly from a distribution $\mathcal{D}$ over Σ, and τ is deterministic. Each state has a unique output in $\mathcal{O}$ associated with it.

Random tape. The initial state of the algorithm is drawn randomly from a distribution $\mathcal{D}$ over Σ.[3] $\mathcal{R}$ is a sample space; when the algorithm receives an input $e \in \mathcal{I}$, and is at state $\sigma \in \Sigma$, it chooses a random $\rho \in \mathcal{R}$ independent of all previous choices and moves to state $\tau(e, \sigma, \rho)$. Each state has a unique output value in $\mathcal{O}$ associated to it.[4]

Random oracle. The initial state of the algorithm is fixed. $\mathcal{R}$ is a sample space. At the start of the algorithm, a specific value $R \in \mathcal{R}$ is drawn, and stays the same over the course of the algorithm. When the algorithm is at state σ and receives input e, its next state is $\tau(e, \sigma, R)$. A random oracle algorithm can be interpreted as choosing a random deterministic algorithm, indexed by R, from some list. The output of the algorithm is a function of the current state σ and R.

We briefly comment on the hierarchy of these models. Every z-bit (2^z-state) deterministic algorithm can be implemented in any of the random models using z bits of space; the same holds for any z-bit random seed algorithm. However, with m the maximum stream length, every z-bit random tape algorithm only has a corresponding $\leq (z + \log(m))$-bit random oracle algorithm, because for a random oracle algorithm to emulate a random seed algorithm it must have a way to get "fresh" randomness on each turn. An alternative, which lets one express z-bit random tape algorithms using a z-bit random oracle, is to assume the random oracle algorithm has access to a clock or knows the number of inputs made so far for free; both are reasonable assumptions in practice.

As a consequence of Newman's theorem [New91], any random oracle or random tape algorithm in the static setting with error δ can be emulated using a random seed algorithm with only ε increase in error and an additional $O(\log m + \log\log|\mathcal{I}| + \log\frac{1}{\varepsilon\delta})$ bits of space.[5]

[3] Requiring that this model use a fixed initial state could make implementations of algorithms use one additional "INIT" state.

[4] Alternatively, we could associate a *distribution* of outputs to each state, or a function mapping (input, state) pairs to outputs. As these formulations are slightly more complicated to prove things with, and only affect the space usage of MissingItemFinding algorithms by an additive $O(\log n + \log\frac{1}{\delta})$ amount, we stick with the one state = one output convention.

[5] The resulting emulated algorithm is non-constructive/not polynomial time.

2.2.2 Setting and performance requirements

Definition 2.2.1. We generally require that algorithms be "tracking" [BJWY20]; i.e., that they present an output after each input item and that this *entire sequence* of outputs be correct.[6]

Streaming algorithms are also classified by the kind of correctness guarantee they provide: here are a few possible meanings of the statement "algorithm $\mathcal{A}$ is δ-error" (we assume that $\mathcal{A}$ handles streams of length ℓ with elements in $\mathcal{I}$ and has outputs in $\mathcal{O}$):

Static setting. For all input streams $x \in \mathcal{I}^\ell$, running $\mathcal{A}$ against x will produce incorrect output with probability $\leq \delta$.

Adversarial[7]setting. For all adaptive adversaries α (i.e., functions[8] $\alpha\colon \mathcal{O}^\star \to \mathcal{I}$; programs that choose the next input to $\mathcal{A}$ based on the full history of outputs of $\mathcal{A}$), running $\mathcal{A}$ against α will produce incorrect output with probability $\leq \delta$.

White-box adversarial setting. For all "white box adaptive adversaries" $\mathtt{X}$, running the algorithm $\mathcal{A}$ against $\mathtt{X}$ will produce incorrect output with probability $\leq \delta$. A white-box adaptive adversary is one which chooses the next input as a function of the current *state* of the algorithm.[9] Such adversaries are not omniscient: we assume that for random tape algorithms, they cannot predict any future random decisions, and for random oracle algorithms, they cannot see the contents of the random oracle.[10] When setting a white-box adversary against a random seed algorithm, the random seed algorithm might as well be deterministic, as the white box algorithm can perfectly predict the way it will respond to any input.

Pseudo-deterministic setting. There exists a canonical output function $f\colon \mathcal{I}^\ell \to \mathcal{O}^\ell$ so that, for each $x \in \mathcal{I}^\ell$, $\mathcal{A}(x)$ fails to output $f(x)$ with probability $\leq \delta$.

Algorithms for the static setting are called "classic" streaming algorithms; ones for the adversarial setting are called "adversarially robust" streaming algorithms. All pseudo-deterministic algorithms are adversarially robust, and all adversarially robust algorithms are also classic.

We also consider "zero-error" variants of some of the streaming setting. Here, we require that the algorithm *always* produce correct output, but measure its space cost differently. The algorithm

[6]We do not consider algorithms with a "one-shot" guarantee, to only be correct at the end of the stream, as for MissingItemFinding, graph coloring, and most other problems the difference in space complexity is generally small.

[7]Notation for algorithms has many conflicts: some use "adversarial" to refer to the requirement that a classic algorithm work on every possible input stream, not just randomly ordered streams; others may use it to signify that a fraction of the inputs in an otherwise random stream may be corrupted by an adversary. It may help to think of the "adversarial setting" as the "adaptive setting", instead.

[8]It suffices to consider deterministic adversaries; because any randomized adversary can be implemented by randomly choosing a deterministic algorithm from some distribution; then apply the minimax theorem.

[9]This can model the worst-case scenario of real-world adversaries that use timing attacks or other side channels to determine the algorithm state.

[10]Other papers using the white-box adversarial setting use "random oracle" in the cryptographic sense. See Section 2.2.3.

encodes its state as a string in $\{0,1\}^\star$ using a variable-length, prefix-free encoding, and the cost of the algorithm on a given input is the *expected* peak number of bits that this encoding takes. (See Eq. 2.1.) The two associated settings are:

Zero-error static setting. The cost of the algorithm is maximum value of its cost (expected peak length of the state) on any input stream $x \in \mathcal{I}^\ell$. Formally, say $S_x \in \mathbb{N}^{\ell+1}$ is the random variable giving the history of the space used when an instance of the algorithm is run on the input stream x. The $(i+1)$st entry of S_x is the length of the encoding of the state of the instance after the ith element was received. The cost of the algorithm is:

$$\max_{x \in \mathcal{I}^\ell} \mathbb{E}[\max_{i \in [\ell+1]} (S_x)_i] \,. \tag{2.1}$$

This model is useful in scenarios where many instances of the algorithm are being used in parallel. Then the total amount of memory used will be very unlikely to exceed the expected value by much.[11] One need only consider and handle a single bad event (the entire system running out of memory), instead of carefully picking error levels for individual instances to balance space usage and reliability.

Zero-error adversarial setting. The cost of the algorithm is maximum value of its cost (expected peak state length) when faced with any adaptive adversary $\mathtt{X}$. This cost can be larger than the cost in the zero-error static setting, because an adversary might use the outputs of the algorithm to pick next inputs which will make the algorithm use more space.

It is important to distinguish the *zero-error* setting from the following definition:

Definition 2.2.2. We say that a streaming algorithm is **zero-mistake** with error δ if, for all inputs, it always produces either a correct output or the symbol $\perp$, and produces $\perp$ with probability $\leq \delta$. The space cost of the algorithm is defined to be the maximum space used at any time, over all possible inputs.

As we will explain later, given any zero-error algorithm, one can construct a zero-mistake algorithm for any error level δ. The reverse does not work, and there are many problems with exponential separations between the zero-error and zero-mistake models; MissingItemFinding in Chapter 3 is but one example.

However, for computational and communication complexity, the "zero-error" and "zero-mistake" definitions are equivalent up to constants. A zero-mistake communication protocol can

[11] Thus, the space complexity in the zero-error static setting gives an upper bound on the amortized space complexity.

be converted to a zero-error communication protocol by repeating it until it succeeds [DHP+22].[12] The same can be done for general (Turing-machine) randomized algorithms; repeating a zero-mistake algorithm produces a zero-error/Las-Vegas algorithm. However, for streaming algorithms, it is *not* possible to repeat the stream, so this trick does not work.

In general, if there exists a zero-error algorithm $\mathcal{A}$ using (in the worst case over all inputs) expected peak S bits of space, it can be converted to a δ-error, zero-mistake algorithm which always uses $O(S \log \frac{1}{\delta})$ bits of space. The construction is straightforward: run $\lceil \log \frac{1}{\delta} \rceil$ independent copies of $\mathcal{A}$ on the input, and whenever one of the copies uses more than $2S$ space, stop running it. Outputs are provided using an arbitrary copy of $\mathcal{A}$ that is still running. Each copy independently uses too much space with (by Markov's inequality) $\leq \frac{1}{2}$ probability, so with $\geq 1 - \delta$ probability at least one of the copies will keep running until the end of the stream. (This method works because the copies of $\mathcal{A}$ can *only* stop by running out of space; they never give incorrect outputs.) Using this reduction, lower bounds in the static and adversarial settings can be translated to lower bounds in the zero-error-static and -adversarial settings.

These distinctions will prove important when discussing the MIF problem in Chapter 3, where we will use both zero-error and zero-mistake notions. In fact, most of the streaming graph coloring algorithms in Chapter 4 and streaming edge coloring algorithms in Chapter 5 will provide also a zero-mistake guarantee.[13]

2.2.3 Related work

The following brief survey of work related to the above definitions is by no means comprehensive.

Streaming algorithms. Streaming algorithms are often useful wherever long sequences of data are processed. One of the more notable early[14] works is [AMS99], which describes an algorithm for estimating the frequency moments of cash-register-type data streams. (A cash-register-type stream is a sequence of elements in $[n]$; one has a vector $y \in \mathbb{Z}^{[n]}$ counting the number of times each element arrives; the goal for kth moment estimation is to compute $\sum_{i \in [n]} |y_i|^k$ after the updates have been applied. In contrast, a turnstile stream contains updates which can both increase and decrease the entries of y.) It also proves lower bounds using a reduction from a problem in multi-player one-way communication. Since then, a lot of work has been published on the topic: an older survey is given by [Mut05]. Recent trends in streaming algorithms include the study of graph streaming [McG14]

[12] For one-way communication, the message is also not repeatable, so one could similarly define a zero-error/zero-mistake distinction. The idea is not new–[MWY13] consider one-way protocols parameterized by abort (returning $\perp$) and mistake probabilities.

[13] With some effort, many could also be converted to zero-error algorithms; see Theorem 3.4.3 for how this works for MissingItemFinding.

[14] Technically, [Kah65]'s algorithm for floating point summation qualifies as a streaming algorithm, and has been used as such.

and clustering [ZA21].

Adaptive adversaries and streaming. An influential early work [HW13] considered adaptive adversaries for *linear* sketches for the task of estimating the ℓ_2 norm of turnstile streams; they found that any linear sketch could be broken with a polynomial number of inputs. This is significant because many streaming algorithms use linear sketches; and in fact, [LNW14, KP20] show that under certain conditions, the most efficient streaming algorithms for turnstile streams use linear sketches. The adversarial setting was formally introduced by [BJWY20], who provided general methods (like "sketch-switching") for designing adversarially robust algorithms given classic streaming algorithms, especially in cases where the problem is to approximate a real-valued quantity. For some tasks, like F_0 estimation, they obtain slightly lower upper bounds in the random oracle model, although later work ([WZ22]) removed this assumption. In particular, for $(1 \pm \varepsilon)$-approximating a real-valued function, [BJWY20] gave two generic frameworks that can "robustify" a standard streaming algorithm, blowing up the space cost by roughly the *flip number* $\lambda_{\varepsilon,m}$, defined as the maximum number of times the function value can change by a factor of $1 \pm \varepsilon$ over the course of an m-length stream; this is a lower bound on the number of times the algorithm must change its value (and thereby potentially leak new information that an adversary could exploit). For *insertion-only streams* and monotone functions, $\lambda_{\varepsilon,m}$ is roughly $O(\varepsilon^{-1} \log m)$, so this overhead is very small. Subsequent works [HKM+20, WZ21, ACSS21] have used how to use differential privacy techniques and "difference estimators" to improve this multiplicative overhead; the current best factor is $O\left(\sqrt{\varepsilon \lambda_{\varepsilon,m}}\right)$ [ACSS21].

Most of these papers focus on providing algorithms and general techniques, but there has been some work on proving adversarially robust lower bounds. [KMNS21] describe a problem to approximate a certain real-valued function (derived from an adaptive data analysis problem) that requires exponentially more space in the adversarial setting than in the static setting.

There have been strengthenings—and weakenings—of the adversarial setting since its introduction. [ABJ+22] introduced the white-box adversarial setting, and rule out efficient white-box adversarially robust algorithms for tasks like F_p moment estimation, while finding algorithms for heavy-hitters-type problems. They also show how to reduce white-box adversarially robust algorithms from deterministic 2-party communication protocols, where lower bounds may be easier to prove. On the other hand, [SSS23] consider a modification of the adversarial setting in which the adaptivity of the adversary is limited; they also introduce a model variant where the algorithm can occasionally access an oracle to get advice about the history of the stream.

[CKL+24] consider a clustering problem for "adaptive-order" streams: where the elements of the stream are fixed, but the adversary adaptively chooses the order in which they are presented.

A few other tasks have been studied in the adversarial setting. [BSS22] consider the space needed to maintain dynamic spanners of graphs. [WZZ23, PR23] consider the Experts Problem. [CNSS23, CLN$^+$22] design specific adversaries for the CountSketch algorithm for heavy hitters, and describe a way to make it adversarially robust. [PR22] consider how the HyperLogLog sketch behaves against different types of adversaries, and suggest ways to reinforce the sketch.

Adaptive adversaries and sublinear data structures. A streaming algorithm is just a data structure for a specific task, using space sublinear in the input size. One common randomized, sublinear-space data structure is the Bloom filter, used to check (with false positives, but no false negatives) whether an element is in a set. [NY19] and others [CPS19, FPUV22, NO22] consider adaptive attacks on bloom filters and other probabilistic membership/count query data structures, in some cases providing more robust (but still sublinear space) data structures.

Adversaries and machine learning. What is a machine learning model that learns from data presented online, but a streaming algorithm that produces a compact summary of its input? [HJN$^+$11] gave a survey on the topic of adversaries for online machine learning algorithms, classifying the many possibly objectives of such adversaries – among others, to learn private training information, trick the algorithm to produce incorrect or biased output, or discover on which inputs the model will make mistakes. While there have been many specific attacks and defenses constructed [CAD$^+$18], we are not aware of any strong guarantees, as exist for typical streaming tasks.

Pseudo-determinism. While the notion of pseudo-deterministic algorithms appears to have been introduced (with different terminology) by [GG11], it was only applied to streaming algorithms by [GGMW20], who found lower bounds for a few problems. Later work [BKKS23, GGS23] finds lower bounds for pseudo-deterministic streaming algorithms for approximately counting the number of elements received; the latter finds they require $\Omega(\log m)$ space, where m is the stream length; in contrast, in the static setting, Morris's counter algorithm[15] uses only $O(\log \log m)$ space.

Random oracles. Assuming access to a random oracle is a reasonable temporary measure when designing streaming algorithms in the static setting. As noted at the beginning of Section 4.1, [Ind06] designed L_p-estimation algorithms using random linear sketch matrices, without regard to the amount of randomness used, and then described a way to apply Nisan's PRG [Nis90] to partially derandomize these algorithms and obtain efficient (random seed) streaming algorithms. In general, the use of PRGs for linear sketches has some space overhead, which later work (see [JW23] as a recent example) has been working to eliminate.

[15]Morris's is a "random tape" algorithm; "random seed" algorithms for counting aren't better than deterministic ones.

It is important to distinguish the "random oracle" type of streaming algorithm from the "random oracle model" in cryptography [BR93], in which one assumes that *all* agents have access to the random oracle. Such a public random oracle can be used as stronger version of a cryptographic hash function, with the understanding that later work can use more specific constructions. This is similar to the way in which random oracles can be a temporary step when designing streaming algorithms. [ABJ+22], when defining white box adversaries, also assume they can see the same random oracle as the algorithm; and, for one task, obtain a more efficient algorithm against a computationally bounded white-box adversary, when both have access to a random oracle, than when neither do. Tight lower bounds are known in neither case. [FW23] also designs a few streaming algorithms for this computationally bounded white-box adversarial setting with shared random oracle.

The power of different types of access to randomness has been studied in computational complexity. [Nis93] finds that logspace Turing machines with access to a multiple-access random tape can (with zero error) accept languages that logspace Turing machines with a read-once random tape accept with bounded two-sided error.[16]

2.3 Common lemmas

Lemma 2.3.1 (Multiplicative Azuma's inequality)**.** *Let $X_1, \ldots, X_t$ be $[0,1]$ random variables, and $\alpha \geq 0$. If, for all $i \in [t]$, $\mathbb{E}[X_i \mid X_1, \ldots, X_{i-1}] \leq p_i$, then*

$$\Pr\left[\sum_{i=1}^{t} X_i \geq (1+\alpha)\sum_{i=1}^{t} p_i\right] \leq \exp\left(-((1+\alpha)\ln(1+\alpha)-\alpha)\sum_{i=1}^{t} p_i\right) \leq \exp\left(-\frac{\alpha^2}{2+\alpha}\sum_{i=1}^{t} p_i\right).$$

On the other hand, if for all i, $\mathbb{E}[X_i \mid X_1, \ldots, X_{i-1}] \geq p_i$, then

$$\Pr\left[\sum_{i=1}^{t} X_i \leq (1-\alpha)\sum_{i=1}^{t} p_i\right] \leq \exp\left(-((1-\alpha)\ln(1-\alpha)+\alpha)\sum_{i=1}^{t} p_i\right) \leq \exp\left(-\frac{\alpha^2}{2}\sum_{i=1}^{t} p_i\right).$$

(The usual form of Azuma's inequality uses a martingale presentation and gives an additive-type bound.)

Proof of Lemma 2.3.1. We only prove this elementary lemma because it's hard to find a source for. We essentially repeat the proof of the Chernoff bound, with slight modifications to account for the dependence of X_i on its predecessors.

First, the $\geq$ direction. Choose, with foresight, $z = \ln(1+\alpha)$.

$$\Pr\left[\sum_{i=1}^{t} X_i \geq (1+\alpha)\sum_{i=1}^{t} p_i\right]$$

[16] There is no power difference between read-once and multiple access random tapes for *time* complexity classes, since an algorithm can just copy every bit of the read-once tape that it uses into scratch memory.

$$\begin{aligned}
&= \Pr\left[\exp(z\sum_{i=1}^{t} X_i) \geq \exp(z(1+\alpha)\sum_{i=1}^{t} p_i)\right] \\
&\leq \frac{\mathbb{E}\exp(z\sum_{i=1}^{t} X_i)}{\exp(z(1+\alpha)\sum_{i=1}^{t} p_i)} \\
&\leq \frac{\mathbb{E}[e^{zX_1}\mathbb{E}[e^{zX_2}\ldots\mathbb{E}[e^{zX_t}|X_1,\ldots,X_{t-1}]\ldots|X_1]]}{\exp(z(1+\alpha)\sum_{i=1}^{t} p_i)}\,.
\end{aligned}$$

The innermost term $\mathbb{E}[e^{zX_t}|X_1,\ldots,X_{t-1}]$ is, by the convexity of e^z, $\leq p_t e^z + (1-p_t) \leq e^{p_t(e^z-1)}$; after applying this upper bound, we can factor it out and then bound the $\mathbb{E}[e^{zX_{t-1}}|\ldots]$ term, and so on. Continuing the chain of inequalities gives:

$$\leq \frac{\exp\left((e^z-1)\sum_{i=1}^{t} p_i\right)}{\exp\left(z(1+\alpha)\sum_{i=1}^{t} p_i\right)} = \exp\left(-((1+\alpha)\ln(1+\alpha)-\alpha)\sum_{i=1}^{t} p_i\right)\,.$$

For the other direction, set $z = \ln(1-\alpha)$, which is < 0. This time, $\mathbb{E}[e^{zX_t}|X_1,\ldots,X_{t-1}] \leq p_t e^z + (1-p_t)$ because p_t is a *lower bound* for $\mathbb{E}[X_t|X_1,\ldots,X_{t-1}]$, and z is negative. We again have, $p_t e^z + (1-p_t) \leq e^{p_t(e^z-1)}$, so:

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{t} X_i \leq (1-\alpha)\sum_{i=1}^{t} p_i\right] &= \Pr\left[\exp(z\sum_{i=1}^{t} X_i) \geq \exp(z(1-\alpha)\sum_{i=1}^{t} p_i)\right] \\
&\leq \ldots \leq \frac{\exp\left((e^z-1)\sum_{i=1}^{t} p_i\right)}{\exp\left(z(1-\alpha)\sum_{i=1}^{t} p_i\right)} \\
&= \exp\left(-((1-\alpha)\ln(1-\alpha)+\alpha)\sum_{i=1}^{t} p_i\right)\,. \qquad \square
\end{aligned}$$

The following definition proves helpful when trying to prove concentration bounds derived from randomly chosen subsets in $\binom{[n]}{k}$.

Definition 2.3.2 (Negative association)**.** We say that a collection of real-valued random variables $X_1,\ldots,X_n$ is negatively associated (NA), if for all pairs of disjoint subsets A_1, A_2 of $[n]$, and nondecreasing[17] functions $f_1 : \mathbb{R}^{A_1} \to \mathbb{R}$, $f_2 : \mathbb{R}^{A_1} \to \mathbb{R}$,

$$\mathbb{E}[f_1((X_i)_{i\in A_1}) \cdot f_2((X_i)_{i\in A_1})] \leq \mathbb{E}[f_1((X_i)_{i\in A_1})] \cdot \mathbb{E}[f_2((X_i)_{i\in A_1})]\,. \tag{2.2}$$

This notion was introduced by [JP83]. In particular, if the random variable S is chosen uniformly at random from $\binom{[n]}{k}$, and for each $i \in [n]$, $X_i = \mathbb{1}_{i\in S}$, then the random variables $X_1,\ldots,X_n$ are negatively associated. A few common transformations described in [JP83] preserve this property:

- Subsets of NA families are also NA.

[17] Replacing both functions f_1, f_2 with $-f_1$ and $-f_2$ in Eq. 2.2 does not change the truth value, so we can also assert negative association by considering all nonincreasing functions.

- The union of independent sets of NA random variables is NA.
- Applying nondecreasing functions to disjoint subsets of NA variables produces an NA family. (For example: if $\{X_1, X_2, X_3, X_4\}$ are NA, then so are $\{X_1 + X_2, \max(X_3, 0) \cdot \max(X_4, 0)\}$.)

Lemma 2.3.3 (Chernoff bound with negative association, from [JP83])**.** *The standard multiplicative Chernoff bounds work with negatively associated random variables. Specifically, if $X_1, \ldots, X_n$ are negatively associated $[0,1]$ random variables, $\delta \geq 0$, and $\mu = \mathbb{E} \sum_{i=1}^n X_i$, then:*

$$\Pr[\sum_{i=1}^n X_i \geq (1+\delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu \leq \exp\left(-\frac{\delta^2}{2+\delta}\mu\right)$$
$$\Pr[\sum_{i=1}^n X_i \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^\mu \leq \exp\left(-\frac{\delta^2}{2}\mu\right) .$$

Lemma 2.3.4 (Error amplification by majority vote)**.** *Let $\varepsilon \leq \delta \leq 1/3$. Say X is a random variable, and v a value with $\Pr[X = v] \geq 1 - \delta$. If $X_1, \ldots, X_p$ are independent copies of X, then the most common value in $(X_1, \ldots, X_p)$ will be v with probability $\geq 1 - \varepsilon$, for $\varepsilon = (2\delta)^{p/30}$.*

(This standard lemma can be used to trade error for space for algorithms which only have a single valid output.)

Proof of Lemma 2.3.4. For each $i \in [p]$, let Y_i be the random indicator variable for the event that $X_i \neq v$. Let $\alpha = \frac{1}{2\delta} - 1$.The probability that v is not the most common element can be bounded by the probability that it is the not the majority element; by a Chernoff bound, this is:

$$\begin{aligned}\Pr[\sum_{i\in[p]} Y_i \geq \frac{1}{2}p] = \Pr[\sum_{i\in[p]} Y_i \geq (1+\alpha)\delta p] &\leq \exp\left(-((1+\alpha)\ln(1+\alpha) - \alpha)\delta p\right)\\ &\leq \exp\left(-0.073((1+\alpha)\ln(1+\alpha))\delta p\right) \qquad \text{since } \alpha \geq 1/6\\ &\leq \exp\left(-\frac{0.073}{2\delta}\ln\frac{1}{2\delta}\delta p\right) = (2\delta)^{\frac{0.073}{2}p} \leq (2\delta)^{p/30} .\end{aligned}$$

□

Chapter 3

Streaming algorithms for Missing Item Finding

3.1 Introduction

The extent to which a streaming algorithm is vulnerable to input made by adaptive adversaries depends critically on the use of randomness by the algorithm. An adversary that somehow manages to determine all the past and future random choices made by an algorithm may be able to counteract the benefits of that randomness, and identify a specific continuation of the input stream on which the instance fails. Algorithms that are robust to adversaries often prevent the adversary from learning any of their important random decisions, and ensure that the decisions which are revealed do not affect the future performance of the algorithm. For example, [BJWY20] mentions a sketch-switching method in which a robust algorithm maintains multiple independent copies of a non-robust algorithm; it emits output derived from one non-robust instance until it reaches the point where an adversary might make the instance fail, at which point the algorithm switches to another instance, none of whose random choices have been revealed to the adversary yet.

Recent research has introduced models with requirements stronger than adversarial robustness. In the white-box streaming model [ABJ$^+$22], algorithms must avoid errors even when the adversary can see the current state of the algorithm (i.e, including past random decisions), but not future random decisions. In the pseudo-deterministic model [GGMW20], streaming algorithms should with high probability always give the same output for a given input; such algorithms are automatically robust against adversaries, because (assuming the algorithm has not failed) the outputs of the algorithm reveal nothing about any random decisions made by the algorithm.

In order to better understand the differences between all these models, we study a streaming problem known as Missing Item Finding (MIF). This problem is perhaps the simplest search problem for data streams where the space of possible answers shrinks as the stream progresses.

Problem 3.1.1 (MissingItemFinding (mif))**.** Let $\ell < n$ be positive integer parameters. The input to the $\textsc{mif}(n, \ell)$ problem[1] is a data stream $a_1, \ldots, a_\ell$ of length ℓ, where the elements are integers in the set $[n]$; duplicates are permitted. The goal is, after every stream prefix $a_1, \ldots, a_i$, to output an integer $x \in [n]$ not seen before (so that $x \notin \{a_1, \ldots, a_i\}$.)

This problem has proven itself interesting because it has significantly different space complexities in a variety of streaming models; we obtain significantly different space complexities for randomized algorithms in the static setting, random oracle algorithms in the adversarial setting, and deterministic algorithms. Surprisingly, we find that when $\ell \in [n^{\Omega(1)}, O(\sqrt{n})]$, random oracle adversarially robust algorithms require exponentially less space than random tape robust algorithms; and when $\ell = 2^{O(\sqrt{\log n})}$, adversarially robust random tape algorithms use exponentially less space than random seed algorithms. Thus, in the adversarial setting – unlike the static setting – one can obtain significant space reductions if one has access to a random oracle (in practice, a cryptographic random number generator) or a random tape (in practice, a hardware random number generator).

3.1.1 Results

Definitions for all the models that will be mentioned are given in Sections 2.2.1 and 2.2.2.

Our results are summarized in Table 3.1. For a view organized by setting/performance requirement and by the type of randomness used, see Table 3.2. The space complexities in the adversarial setting are plotted for the four different types of randomness in Figure 3.1. We will briefly present our theorems, grouped logically, not by proof order, and defer detailed explanations of their techniques to the sections in which they are proven.

Randomized algorithms in the static setting. We first consider MissingItemFinding in the static setting. We obtain two nontrivial algorithms: one which is very efficient when $\ell \ll n$, and one which guarantees low space usage even when $\ell = n - 1$. To simplify comparisons with algorithms in the adversarial setting, we present our results using tracking, not one-shot error (recall Definition 2.2.1).

Theorem 3.2.1. *In the static setting, there is a random oracle streaming algorithm, Algorithm 3.2.1, which solves $\textsc{mif}(n, \ell)$ with tracking error $\leq \delta$, and uses $t \leq \min(\ell, \frac{\log(1/\delta)}{\log(n/\ell)})$ bits of space. (The total number of oracle random bits used is $O((t + 1)\log\frac{en}{t+1})$: accounting for these explicitly gives a random-seed algorithm using $O(\min(\ell, \log(n) + \log(1/\delta)\frac{\log(n)}{\log(n/\ell)}))$ space.)*

[1] For this problem, we use ℓ for the stream length, instead of m, because m and n are too easily confused in speech and in handwriting.

[2] The table entry for the white-box adversarial, random oracle case assumes that only the algorithm has access to the random oracle, not the adversary.

Setting	Type	Bound	Reference
Static	Deterministic	$O(\sqrt{\ell \log \ell} + \frac{\ell \log \ell}{\log n})$	Theorem 3.5.2
		$O(\frac{\ell \log\log(4n/\ell)}{\log(2n/\ell)} + \sqrt{\ell \log \ell})$	Theorem 3.5.3 †
		$\Omega(\frac{\ell}{\log(2n/\ell)} + \sqrt{\ell})$	Theorem 3.5.1
Static	Random seed	$O\left((\log n)^2 \log \frac{\log n}{\delta}\right)$	Theorem 3.2.2 †
Static	Random tape	$\geq \log(\ell + 1)$	Lemma 3.1.2
Static	Random oracle	$O(\frac{\log(1/\delta)}{\log(n/\ell)})$	Theorem 3.2.1
Static	Random oracle	$\geq \Omega\Big(\frac{\log(1/\delta)}{\log n \log(2n/\ell)} + \sqrt{\frac{\log(1/\delta)}{\log n}})\Big)$	Theorem 3.9.1
Static	Random oracle	$\geq \Omega\left(\max\left(0, \frac{\ell}{n} \min\left(\ell, \frac{\log \frac{2}{\delta}}{\log \frac{2n}{\ell}}\right) - 50\right)\right)$	Theorem 3.9.3
Adversarial	Random oracle	$O((1 + \frac{\ell^2}{n} + \log \frac{1}{\delta}) \log \ell)$	Theorem 3.3.6
		$\Omega(\log(1-\delta) + \frac{\ell^2}{n})$	Theorem 3.3.5
		$\geq \log(\ell + 1)$	Lemma 3.1.3
Zero error adversarial	Random oracle	$O((1 + \frac{\ell^2}{n}) \log n)$	Theorem 3.4.2
Zero error static	Random oracle	$\Omega(1 + \frac{\ell^2}{n})$	Theorem 3.4.1
Pseudo-deterministic	Random oracle	$\Omega(\frac{\ell \log 1/(2\delta)}{(\log \frac{2n}{\ell})^2 \log n} + \left(\ell \log \frac{1}{2\delta}\right)^{1/4})$	Theorem 3.6.8
Adversarial	Random seed	$O((\frac{\ell^2}{n} + \sqrt{\frac{\ell}{\log n}} + \ell^{1/3} + \log \frac{1}{\delta}) \log \ell)$	Theorem 3.7.6
		$\Omega(\sqrt{\frac{\ell}{(\log n)^3}} + \ell^{1/5})$ if $\delta \leq \frac{1}{6}$	Corollary 3.7.3
Adversarial	Random tape	$O(\ell^{\frac{\log \ell}{\log n}} (\log \ell)^2 \log \frac{1}{\delta})$	Theorem 3.10.6
		$\Omega(\frac{\log \ell}{\log n} \ell^{\frac{15}{32} \frac{\log \ell}{\log n})})$ if $\delta \leq \frac{\ell}{2^{13} n}$	Theorem 3.11.7
White-box adversarial	Random tape	$\Omega(\frac{\ell}{\log(2n/\ell)} + \sqrt{\ell})$ if $\delta = O(\min(1, \frac{\ell^2}{n}))$	Theorem 3.8.1

Table 3.1: All our results for the space complexity of MissingItemFinding(n, ℓ), parameterized by tracking error level $\delta \in [2^{-\ell}, 1/3]$, where applicable. Some of the theorems provide tighter and more verbose bounds. † = algorithm is not polynomial time

Table 3.2: A more structured view of the space bounds from Section 3.1.1. The rows indicate the setting/performance requirement for the algorithm; the column its implementation type. This table shows how the space complexity increases as the requirements grow and use of randomness are restricted. Bounds are evaluated at tracking error level $\delta = O(1/\operatorname{poly}(n))$, *and* with $\ell = n^{\Omega(1)}$.

	Random Oracle	Random Tape	Random Seed	Deterministic
Static	$\Omega(\frac{\ell}{n}\log n)$ $O((\log n)^3)$	$\Omega(\log \ell)$		
Adversarial	$\Omega(\frac{\ell^2}{n} + \log \ell)$ $O((\frac{\ell^2}{n}+1)\log n)$	$\Omega(\ell^{\frac{15}{32}\frac{\log \ell}{\log n}})$ $O(\ell^{\frac{\log \ell}{\log n}}(\log n)^2)$	$\Omega(\frac{\ell^2}{n} + \sqrt{\frac{\ell}{(\log n)^3}})$ $O((\frac{\ell^2}{n}+\sqrt{\ell})(\log n)^2)$	$\Omega(\frac{\ell}{\log(n/\ell)})$
Zero error static	$\Omega(\frac{\ell^2}{n} + \log \ell)$ $O((\frac{\ell^2}{n} + \log \ell)\log \ell)$			
Zero error adversarial	$\Omega(\frac{\ell^2}{n} + \log \ell)$ $O((\frac{\ell^2}{n}+1)\log \ell)$	$\Omega(\frac{1}{\log n}\ell^{\frac{15}{32}\frac{\log \ell}{\log n}})$ $O(\ell^{\frac{\log \ell}{\log n}}(\log \ell)^2)$	$\Omega(\frac{\ell^2}{n} + \sqrt{\frac{\ell}{(\log n)^3}})$ $O((\frac{\ell^2}{n}+\sqrt{\ell})\log \ell)$	
Pseudo-deterministic	$\Omega(\frac{\ell}{(\log n)^3} + \ell^{1/4})$			$O(\frac{\ell \log\log(n/\ell)}{\log(n/\ell)})$
White-box adversarial	See Adv./R. Oracle[2]	$\Omega(\frac{\ell}{\log(n/\ell)})$	See Det.	

Theorem 3.2.2. *Algorithm 3.2.2 is a random seed streaming algorithm solving* $\textsc{mif}(n, n-1)$ *with tracking error* $\leq \delta$ *in the static setting. It uses* $O((\log n)^2 \log \frac{\log n}{\delta})$ *bits of space.*

At small error levels ($\delta = 1/\operatorname{poly}(n)$), the algorithm of Theorem 3.2.2 uses $O((\log n)^3)$ bits of space. This becomes more efficient than the random-seed version of Theorem 3.2.1's algorithm when $\ell \geq n - O(n/\log n)$; and more efficient than the random oracle version when $\ell \geq O(n/(\log n)^2)$.

Unfortunately, we do not have tight lower bounds in either case. For random-tape (and hence also random-seed) algorithms, there is an $\geq \log(\ell+1)$ space lower bound for $\textsc{mif}(n,\ell)$, from Lemma 3.1.2. For cases where δ is very small, we obtain, by a reduction to the $\textsc{mif}$ lower bound for deterministic algorithms:

Theorem 3.9.1. *For any* $\delta \leq 1/(2n)$*, the space complexity for a random oracle algorithm solving* $\textsc{mif}(n,\ell)$ *with error* $\leq \delta$ *is*

$$\geq \Omega\left(\sqrt{\min\left(\ell, \frac{\log(1/\delta)}{\log n}\right)} + \min\left(\ell, \frac{\log(1/\delta)}{\log n}\right)\frac{1}{1+\log(n/\ell)}\right).$$

On the other hand, a different approach (which essentially reduces the problem to two-player communication) gives:

Theorem 3.9.3. *For any $\delta \leq 1/2$ and $\ell \geq 4$, the space complexity of a random oracle algorithm solving* MIF(n, ℓ) *with error $\leq \delta$ (even if just evaluated at the end) is*

$$\geq \Omega\left(\max\left(0, \frac{\ell}{n}\min\left(\ell, \frac{\log\frac{2}{\delta}}{\log\frac{2n}{\ell}}\right) - 50\right)\right) .$$

The two results are incomparable; each is better for a certain set of parameters. The second, when $\ell = \Theta(n)$ and $\delta = \Omega(1)$, has the form $\Omega(\log(1/\delta))$, where the first only yields an $\Omega(\frac{\log(1/\delta)}{\log n})$ lower bound. On the other hand, the first theorem is much more useful when $\ell \ll n$, as for parameters like $\ell = \sqrt{n}$, $\delta = 1/\operatorname{poly}(n)$ the first theorem implies $\Omega(\sqrt{\log n})$ space is needed, while the second gives a lower bound of zero.

We note that there are no nontrivial *random oracle* lower bounds for constant values of δ which hold when $\ell \ll n$ – because the random oracle algorithm which just outputs a uniform random number in $[n]$ (determined from the oracle), will use zero bits of state and have worst-case error level $\delta = \frac{\ell}{n}$ in the static setting.

Deterministic algorithms. We obtain almost matching upper and lower bounds for the space complexity of deterministic streaming algorithms for MissingItemFinding. (As deterministic algorithms have the same space complexity in all settings mentioned in this thesis, it is not necessary to mention the setting.) Surprisingly, one can do somewhat better than the simple algorithm which uses ℓ bits of state (which is described later in Algorithm 3.1.1).

Theorem 3.5.1. *Every deterministic streaming algorithm for* MIF(n, ℓ) *requires $\Omega(\sqrt{\ell} + \frac{\ell}{1+\log(n/\ell)})$ bits of space.*

Theorem 3.5.2. *Algorithm 3.5.1 is a deterministic algorithm that solves* MIF(n, ℓ) *using $O(\sqrt{\ell}\log\ell + \frac{\ell\log\ell}{\log n})$ bits of space.*

Theorem 3.5.3. *There is a deterministic algorithm (Algorithm 3.5.2) for* MIF(n, ℓ) *using space:*

$$O\left(\frac{\ell\log\log\frac{4n}{\ell}}{\log\frac{4n}{\ell}} + \sqrt{\ell\log\ell}\right) .$$

Of the two deterministic algorithms for MIF(n, ℓ), Algorithm 3.5.1 processes updates in polynomial time, while the slightly more space-efficient Algorithm 3.5.2 relies on a non-explicitly constructed set family. In both cases, the algorithms use within an $O(\log\ell)$ multiplicative factor of the space lower bound for all values of ℓ, n. The terms similar to $\frac{\ell}{\log n}$ dominate as long as $\ell = \Omega((\log n)^2)$; and when $\ell = O((\log n)^2)$, the algorithm can no longer effectively use all the possible output values, and the deterministic space complexity falls between $\Omega(\sqrt{\ell})$ and $O(\sqrt{\ell\log\ell})$.

Adversarial setting. In the adversarial setting, we find separations between all four types of randomness. As the formulas for the random tape case are somewhat complicated, we plot the asymptotic behavior in the $\ell \in [n^{\Omega(1)}, n]$ regime as Figure 3.1.

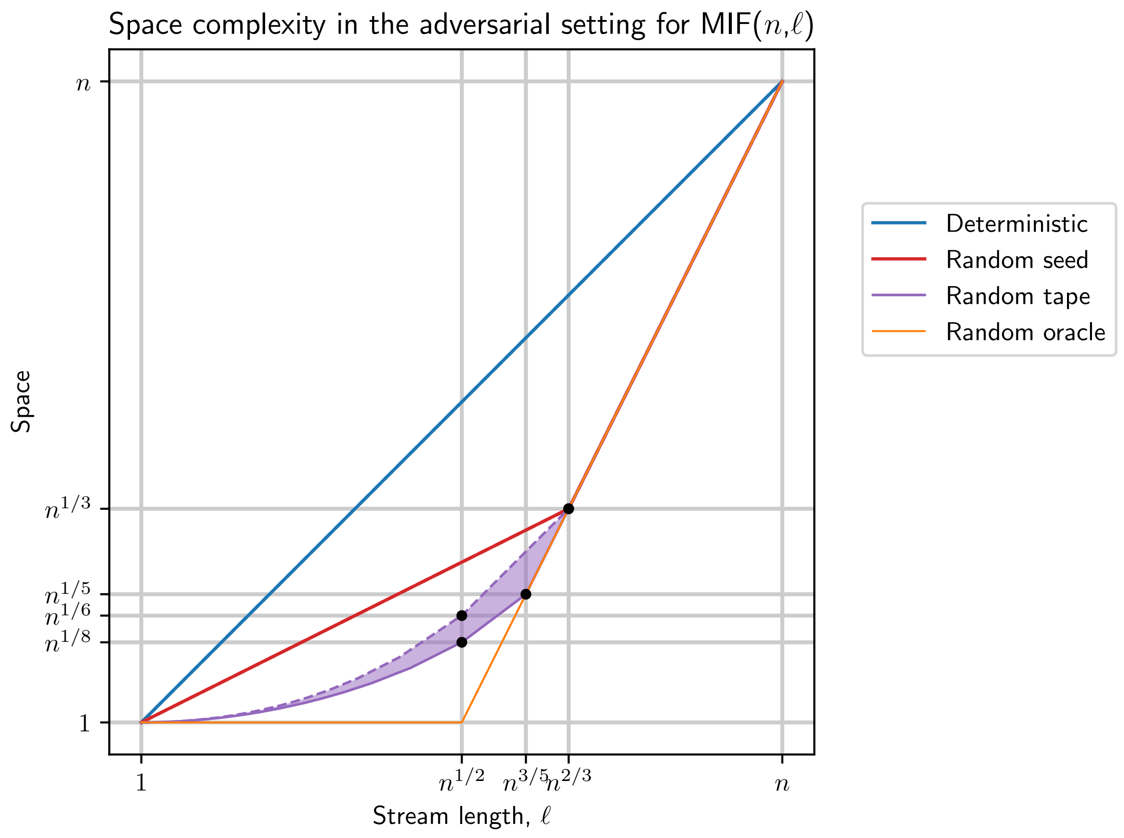


Figure 3.1: A log-log scale plot of the space complexity of algorithms in the adversarial setting with four different types of access to randomness. The upper and lower bounds for the random tape case differ asymptotically; the possible space complexities are shown in the shaded region. The random seed, tape, and oracle space complexities overlap for $\ell \geq n^{2/3}$. The plot is evaluated at $n = 2^{10000}$ and $\delta = 1/n^2$.

First, we find almost matching upper and lower bounds for algorithms using a random oracle.

Theorem 3.3.5. *Any algorithm which solves* $\text{MIF}(n, \ell)$ *against adaptive adversaries with error* δ *requires* $\geq \log(\binom{n}{\lceil \ell/2 \rceil} / \binom{n - \lceil \ell/2 \rceil}{\lfloor \ell/2 \rfloor + 1}) + \log(1 - \delta)$ *bits of space; or less precisely,* $\Omega(\ell^2/n + \log(1 - \delta))$.

Theorem 3.3.6. *Algorithm 3.3.1 solves* $\text{MIF}(n, \ell)$ *against adaptive adversaries, with error* δ*, and can be implemented using* $O(\min(\ell, \left(1 + \frac{\ell^2}{n} + \ln \frac{1}{\delta}\right) \cdot \log \ell))$ *bits of space. (It requires oracle access to* $(\ell + 1) \log n$ *random bits.)*

A key point for random oracle algorithms is when $\ell = \sqrt{n}$; below this threshold, only $O(\log \frac{1}{\delta} \log \ell)$ bits of space are needed.

While we have not fully resolved the random tape space complexity, our upper and lower bounds for it are both piecewise-polynomial curves in ℓ:

Theorem 3.10.6. *There is a family of adversarially robust random tape algorithms, where for* $\textsc{mif}(n, \ell)$ *the corresponding algorithm has* $\le \delta$ *error and uses*

$$O\left(\left\lceil \frac{(4\ell)^{\frac{2}{d-1}}}{(n/4)^{\frac{2}{d(d-1)}}} \right\rceil (\log \ell)^2 + \min(\ell, \log \tfrac{1}{\delta}) \log \ell \right)$$

bits of space, where $d = \max\left(2, \min\left(\lceil \log \ell \rceil, \left\lfloor 2\frac{\log(n/4)}{\log(16\ell)} \right\rfloor\right)\right)$. *At* $\delta = 1/\operatorname{poly}(n)$ *this space bound is* $O\left(\ell^{\log_n \ell}(\log \ell)^2 + \log \ell \log n\right)$.

Theorem 3.11.7. *Random tape* δ*-error adversarially robust algorithms for* $\textsc{mif}(n, \ell)$ *require*

$$\Omega\left(\max_{k \in \mathbb{N}} \frac{1}{k} \left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}}\right) = \Omega\left(\frac{\log \ell}{\log n} \ell^{\frac{15}{32} \log_n \ell}\right)$$

bits of space, for $\delta \le \frac{\ell}{2^{13} n}$.

For the algorithm of Theorem 3.10.6, the point where the space complexity stops depending on n is when $\ell = 2^{\Theta(\sqrt{\log n})}$. For ℓ below this point, $O((\log \ell)^2 + \log 1/\delta \log \ell)$ space is used (and by Lemma 3.1.2, $\Omega(\log \ell)$ space is needed.) Also note that for $\ell \ge n^{2/3}$, the space complexity matches the random oracle case (up to $\operatorname{polylog}(n)$ factors).

Random seed algorithms require a bit more space. Here we again have relatively tight bounds when $\ell = n^{\Omega(1)}$, although the exact exponent for ℓ when $\ell = O(\log n)$ is still open.

Corollary 3.7.3. *Adversarially robust random seed algorithms for* $\textsc{mif}(n, \ell)$ *with error* $\le \frac{1}{6}$ *require* $\Omega\big(\sqrt{\ell/(\log n)^3} + \ell^{1/5}\big)$ *bits of space.*

Theorem 3.7.6. *Algorithm 3.7.1 is a random seed algorithm that solves* $\textsc{mif}(n, \ell)$ *in the adversarial setting, with error* $\le \delta$*, and can be implemented using* $O\left(\left(\frac{\ell^2}{n} + \sqrt{\frac{\ell}{\log n}} + \ell^{1/3} + \log \frac{1}{\delta}\right) \log \ell\right)$ *bits of space.*

Random seed algorithms always require $\ell^{\Omega(1)}$ space; when $\ell = 2^{\Theta(\sqrt{\log n})}$, random tape algorithms use exponentially less space.

Zero-error variations. We briefly consider the zero-error static and adversarial settings. We find that zero-error algorithms in the static setting require (in expectation) about as much space as random oracle algorithms in the adversarial setting; the same lower bound applies, of course, to random seed and random tape algorithms.

Theorem 3.4.1. *All algorithms solving* MIF(n, ℓ) *with zero error on any stream require* $\Omega(\ell^2/n)$ *bits of space, in expectation over the randomness of the algorithm.*

Almost matching this theorem, we find that it is possible to convert any algorithm for MIF(n, ℓ) that does not "guess blindly" to a zero-error algorithm using some additional expected space.

Theorem 3.4.3. *Let* $\mathcal{B}$ *be a zero-mistake (Definition 2.2.2) algorithm solving* MIF(n, ℓ) *with* $\leq 1/\ell$ *error in the static (or adversarial) settings, implemented as a random tape, seed, or oracle algorithm, with worst-case space usage* S*. Then there is a zero-error static (or adversarial) algorithm with the same randomness type solving* MIF(n, ℓ) *with zero error that uses at most* $S + O((1 + \ell^2/n)\log \ell)$ *bits of space in expectation, on any input stream (or adversary).*

In particular, by modifying the algorithm of Theorem 3.2.1 to abort instead of guessing when the variable x is the all-ones vector, and applying this Theorem 3.4.3, we obtain a zero-error random-seed algorithm in the static setting which uses $O(\min(\ell, \log \ell \frac{\log n}{\log(n/\ell)})) + O((\log \ell + \ell^2/n)\log \ell) = O((\log \ell + \ell^2/n)\log \ell)$ bits of space in expectation.

As noted in Section 2.2, a zero-error algorithm with expected space S can be converted to a zero-mistake randomized streaming algorithm with error δ and space $O(S \log \frac{1}{\delta})$. Consequently, the lower bounds in the adversarial setting all translate to lower bounds in the zero-error adversarial setting. In particular, the bound of Corollary 3.7.3, as it holds for $\delta = 1/6$, also applies for zero error algorithms (albeit weaker by a constant factor). As the random tape lower bound of Theorem 3.11.7 requires $\delta = O(\ell/n)$, the corresponding lower bound for zero-error random-tape algorithms in the adversarial setting is lower by a factor of $O(\log \frac{n}{\ell}) = O(\log n)$.

As our random seed, random tape, and random oracle algorithms for the adversarial setting are all zero-mistake, we can combine them with Theorem 3.4.3 to obtain corresponding zero-error algorithms for the adversarial setting. For random oracle algorithms in particular, a simple variation on the random oracle algorithm of Theorem 3.3.6 can also be used instead of grafting on code with Theorem 3.4.3:

Theorem 3.4.2. *There is an algorithm solving* MIF(n, ℓ) *with zero error against adaptive adversaries, which uses* $O((1 + \ell^2/n)\log \ell)$ *bits of space, in expectation over the randomness of the algorithm.*

White-box adversarial. In the white-box adversarial setting, we consider only random tape algorithms.[3] We find:

[3]As noted in Section 2.2, against a white-box adversary, a random seed is not a secret, and one can do no better than deterministic algorithms; while a random oracle, assuming the white-box adversary cannot see it, can be used to conceal the algorithm state entirely, thereby reducing the white-box adversary to a regular adaptive adversary.

Theorem 3.8.1. *Random tape algorithms for* MIF(n, ℓ) *in the white-box adversarial setting with error* $\delta \le \min\left(\frac{1}{10}, \frac{\ell^2}{400n}\right)$ *require space*

$$\Omega\left(\frac{\ell}{1 + \log\frac{n}{\ell}} + \sqrt{\ell}\right) .$$

This gives a strong lower bound for the $\delta = O(\min(1, \ell^2/n))$ regime, matching up to constants the current best lower bound for deterministic algorithms. As the simple $O(\log n)$-bit algorithm that produces a random output every step has error $\Theta(\min(1, \ell^2/n))$, it is not possible to extend this lower bound to values of δ which are larger by more than a constant factor.

Pseudo-deterministic. A lower bound on the space complexity of pseudo-deterministic algorithms for MissingItemFinding is given by the following theorem. As δ approaches 0, the lower bound approaches (within a constant factor) the deterministic lower bound.

Theorem 3.6.8. *Pseudo-deterministic* δ*-error random oracle algorithms for* MIF(n, ℓ) *require*

$$\Omega\left(\min\left(\frac{\ell}{\log\frac{2n}{\ell}} + \sqrt{\ell}, \frac{\ell \log\frac{1}{2\delta}}{(\log\frac{2n}{\ell})^2 \log n} + \left(\ell \log\frac{1}{2\delta}\right)^{1/4}\right)\right)$$

bits of space when $\delta \le \frac{1}{3}$*. In particular, when* $\delta = 1/\operatorname{poly}(n)$ *and* $\ell = \Omega(\log n)$*, this is:*

$$\Omega\left(\frac{\ell}{(\log\frac{2n}{\ell})^2} + (\ell \log n)^{1/4}\right) .$$

We do not know of any pseudo-deterministic algorithms for MIF which are more efficient than the deterministic algorithm of Theorem 3.5.3.

3.1.2 Related work

The literature related to the streaming models studied in this chapter has been surveyed in Section 2.2.3. In what follows we consider the context of the MissingItemFinding problem.

Variations. The MissingItemFinding problem appears to have been first studied by [Tar07]. While they primarily consider the problem of finding a duplicate element in a stream of $m > n$ elements chosen from $[n]$, most of their results also apply to MIF$(n, n-1)$. For example, their multi-pass duplicate finding algorithms can easily be translated to multiple pass algorithms to find a missing element. Their main results also hold: they find an deterministic streaming algorithm for MIF$(n, n-1)$ using $O(\log n)$ bits of space must make $\Omega(\log n / \log\log n)$ passes over the stream, and claim that a single-pass deterministic algorithm for MIF$(n, n-1)$ requires at least $2^n - 1$ states.[4]

[4]As Algorithm 3.1.1 uses exactly 2^{n-1} states for MIF$(n, n-1)$, the value $2^n - 1$ may be a typo.

A variation on the MissingItemFinding problem, that forbids repeated elements in the input stream, was briefly studied in the first chapter of [Mut05], which mentions that for any $k \geq 1$, on a stream encoding a subset of $[n]$ of size $n-k$, it is possible to recover the remaining k elements with a sketch of size $O(k \log n)$.

If we were to extend the MissingItemFinding problem to turnstile streams, then we would end up with something opposite to the "support-finding" streaming problem. In the support-finding problem, the algorithm is given a turnstile stream of updates to a vector $x \in \mathbb{Z}^{[n]}$; on querying the algorithm, it must return any index $i \in [n]$ where $x_i \neq 0$. [KNP+17] find that this problem – and the harder L_0 sampling problem, where one must find a uniformly random element of the support of x – have a space lower bound of $\Omega\left(\min\left(n, \log\frac{1}{\delta}(\log\frac{n}{\log(1/\delta)})^2\right)\right)$. This is close to [JST11]'s L_0 sampling algorithm which uses $O(\log\frac{1}{\delta}(\log n)^2)$ bits of space.

Similar games. The paper [MN22b] studies a two player game that is similar to MissingItemFinding. Here there are two players, a "Dealer" and a "Guesser": for each of n turns, the players simultaneously do the following: the Dealer chooses a number from $[n]$ that it has not picked so far, and the Guesser guesses a number in $[n]$. The goal of the Guesser is to maximize expected score, the number of times their number matches the Dealer's choice; the Dealer tries to minimize the score. The paper proves upper and lower bounds on the expected score, for a number of scenarios. Notably, a Guesser that is limited to remember only m bits of information can do much better against a static Dealer (that chooses a hard ordering of numbers at the start of the game) than against an adaptive Dealer (that may choose the next number depending on the guesses made by the Guesser). For example, $m = O((\log n)^2)$ suffices for an expected score of $\Omega(\log n)$ against a static Dealer, but there exists an adaptive Dealer which limits any Guesser's expected score to $(1+o(1)) \ln m + O(\log\log n)$. The objectives of the Guesser and Dealer are similar to those of the algorithm and adversary in MissingItemFinding: the Guesser tries to avoid, if possible, guessing any value that the Dealer has revealed before; while the Dealer tries to ensure the Guesser chooses that the Dealer had already sent before. However, unlike MissingItemFinding, the Dealer-Guesser game requires that numbers dealt never be repeated and that all numbers be used, which makes it much easier to identify a number that will be dealt in the future.

In the Mirror Game of [GS18], there are two players, Alice and Bob who alternately declare numbers from the set $[2n]$. Unlike mif, players must continue until all numbers are used. The players lose if they declare a number that has been declared before. Since Alice goes first, even if Bob can only remember $O(\log n)$ bits about the history of the game, Bob still has a simple strategy that will not lose. On the other hand, [GS18] prove that in order for Alice to guarantee a draw against Bob, Alice requires $\Omega(n)$ bits of memory. If a low probability of error is acceptable, [Fei19] provide a randomized strategy for Alice with $O((\log n)^3)$ bits of memory that draws with high probability – but this requires oracle access to a large number of random bits, or cryptographic assumptions. [Fei19] and [MN22b] ask whether there is a strategy using $O(\operatorname{polylog} n)$ bits of memory and of randomness. [MN22a] considers the scenario in which Alice is "open-book" (equivalently, that Bob is a white-box adversary and can see her state), and find that Alice needs $\Omega(n)$ bits of state to tie-or-win.

Data structures. The problem of constructing an adversarially resilient Bloom filter is addressed by [NY19]. Here one seeks an "approximate set membership" data structure, which is initialized on a set S of size n, and thereafter answers queries of the form "is $x \in S$" with false positive error probability ε. An implementation of this structure is adversarially resilient if the false positive probability of the last element in the sequence is still $\leq \varepsilon$ when the adversary chooses the sets S, and adaptively chooses the sequence of t elements to query. In addition to lower and upper bound results conditional on the existence of one-way functions, [NY19] find a construction for an adversarially resilient bloom filter using $O(n \log 1/\varepsilon + t)$ bits of memory.

Streaming. The MissingItemFinding problem has connections to graph streaming problems. Just as the L_0-sampling problem has been used by streaming algorithms that find a structure in a graph, behaviors like those of the MissingItemFinding problem appear in algorithms that look for a structure which is not in a graph. Specifically, the graph coloring problem is equivalent to finding a small collection of cliques which cover all vertices but do not include any edge in the graph. [ACK19] proved that general randomized streaming algorithms can $\Delta+1$ color a graph in $\widetilde{O}(n)$ space, where n is the number of vertices, and [ACS22] proved that deterministic streaming algorithms using $\widetilde{O}(n)$ space must use $\exp(\Delta^{\Omega(1)})$ colors. Their lower bound is noteworthy in particular because they independently discovered essentially the same core argument as the deterministic lower bound for MissingItemFinding, Algorithm 3.5.2. We suspect that many graph coloring problems will behave similarly to MissingItemFinding, and discuss this further in Section 3.12.

Independent work. [Mag24] independently proved a few results for mif, that are largely superseded by this chapter. They prove an $\Omega(\ell/\log n)$ lower bound for pseudo-deterministic algorithms for $\textsc{mif}(n, \ell)$ which have the *zero-mistake* property (recall Definition 2.2.2), of either producing a correct value or the $\perp$ symbol, and get a corresponding $\Omega(\sqrt{\ell/\operatorname{polylog}(n)})$ lower bound for robust zero-mistake random-seed algorithms using Lemma 3.7.1. They also describe an $O(\operatorname{polylog}(n))$-space random seed algorithm for the static setting, and give upper and lower bounds for solving $\textsc{mif}(n, \ell)$ for "overfull" inputs in which $\ell \geq n$, but with the additional promise that there is at least one missing item.

3.1.3 Warm-up

A simple algorithm. While in most cases there are more efficient alternatives, this algorithm for MIF(n, ℓ) is particularly simple, and uses only ℓ bits of space.

Algorithm 3.1.1 A simple deterministic streaming algorithm for MIF(n, ℓ)

```
   Initialization:
1: $x \leftarrow \{0, \ldots, 0\}$, a vector in $\{0,1\}^{[\ell]}$

   Update($e \in [n]$):
2: if $e \leq \ell$ then
3:     $x_e \leftarrow 1$

   Query:
4: if $\exists j \in [\ell] : x_j = 0$ then
5:     output: $j$
6: else
7:     output: $\ell + 1$
```

Basic observations.

Lemma 3.1.2. *Random tape and random seed streaming algorithms for* MIF(n, ℓ)*, which have a nonzero chance of success on all inputs, require* $\geq \log(\ell + 1)$ *bits of space.*

Proof of Lemma 3.1.2. Each state of a random tape or random seed streaming algorithm has a single associated output value. If $z < \log(\ell+1)$, then the streaming algorithm has at most ℓ states. Let H be the set of outputs of these states. When sent a stream containing each element of H, the algorithm will fail because every possible output has been used. $\square$

Lemma 3.1.3. *Random oracle streaming algorithms in the adversarial setting for* MIF(n, ℓ)*, with a nonzero chance of success on all inputs, require* $\geq \log(\ell + 1)$ *bits of space.*

Proof of Lemma 3.1.3. Given a random oracle streaming algorithm $\mathcal{A}$, run it against the "echo" adversary, which repeatedly checks the output of the algorithm and sends that output back as the next input. By the definition of a random oracle algorithm in Section 2.2.1, each state of $\mathcal{A}$ corresponds to a single output. Run against the "echo" adversary, to be correct $\mathcal{A}$ must produce $\ell+1$ distinct outputs: one to start, and one new output after every input. Thus, $\mathcal{A}$ must use $\geq \ell+1$ different states. $\square$

3.2 Classical randomized algorithms

In this section we give two algorithms for MissingItemFinding in the static setting. Lower bounds in the static setting are somewhat complicated (and far from being tight), so we defer their presentation until Section 3.9.

3.2.1 A sampling algorithm

We give a simple algorithm that chooses a small random subset of $[n]$, tracks which of the elements in that set have been included ("covered") in the stream so far, and outputs an arbitrary element from the set that was not covered. This algorithm scales badly as ℓ/n approaches 1, as the subset must be made larger to compensate for the increased fraction of covered elements,

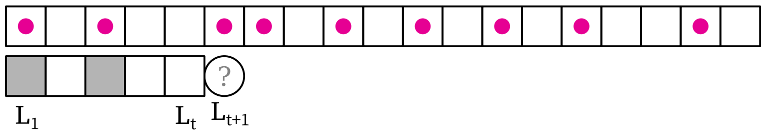


Figure 3.2: This diagram shows the behavior of Algorithm 3.2.1 on an example input. The top row of squares corresponds to the set $[n]$, ordered so that the leftmost squares corresponds to the elements L_1, L_2, …, L_{t+1} from Algorithm 3.2.1. In the top row, cells contain a pink dot if the corresponding element has already been seen in the stream. In the bottom row, each of the cells is shaded dark if the corresponding entry in the vector x is equal to 1 – except for L_{t+1}, whose state Algorithm 3.2.1 does not track.

Theorem 3.2.1. *In the static setting, there is a random oracle streaming algorithm, Algorithm 3.2.1, which solves* MIF(n,ℓ) *with tracking error* $\le \delta$*, and uses* $t \le \min(\ell, \frac{\log(1/\delta)}{\log(n/\ell)})$ *bits of space. (The total number of oracle random bits used is* $O((t+1)\log\frac{en}{t+1})$*: accounting for these explicitly gives a random-seed algorithm using* $O(\min(\ell, \log(n) + \log(1/\delta)\frac{\log(n)}{\log(n/\ell)}))$ *space.)*

Proof of Theorem 3.2.1. First, we observe that Algorithm 3.2.1 gives an incorrect output only when the input stream $\sigma = (e_1, \ldots, e_\ell)$ contains every element of L. Otherwise, either the first t elements of L are in σ, and L_{t+1} isn't – in which case Line 8 returns L_{t+1} – or there is some $j \in [t]$ where L_j has not been seen in the stream so far, in which case Line 6 correctly returns L_j. Given a fixed input stream $\sigma \in [n]^\ell$, the probability that Algorithm 3.2.1 fails is:

$$\Pr[L \subseteq \sigma] = \binom{|\sigma|}{t+1} \Big/ \binom{n}{t+1} \le \binom{\ell}{t+1} \Big/ \binom{n}{t+1} = \frac{\ell(\ell-1)\cdots(\ell-t)}{n(n-1)\cdots(n-t)} \le \left(\frac{\ell}{n}\right)^{t+1} .$$

Thus $\Pr[L \subseteq \sigma]$ is $\le \delta$ when $t = \lfloor \log(1/\delta)/\log(n/\ell) \rfloor$, and is equal to 0 when $t = \ell$, because no set of size ℓ can contain a set of size $\ell + 1$.

Algorithm 3.2.1 A streaming algorithm for $\textsc{mif}(n, \ell)$ with error rate $\leq \delta$ on any input stream

```
   Let $t = \min(\ell, \lfloor \log(1/\delta)/\log(n/\ell) \rfloor)$

   Initialization:
1: Let $L = \{L_1, \ldots, L_{t+1}\}$ be a random subset of $t+1$ distinct elements in $[n]$, chosen uniformly
   at random, with $L_1 < L_2 < \ldots < L_{t+1}$.  ▷ This can be stored explicitly using $O(\log \binom{n}{t+1})$ bits,
   or computed on demand as a function of an oracle random string.
2: $x \leftarrow (0, \ldots, 0)$, a vector in $\{0,1\}^t$

   Update($e \in [n]$):
3: if $\exists j \in [t] : L_j = e$ then
4:     $x_j \leftarrow 1$

   Query:
5: if $\exists j \in [t] : x_j = 0$ then
6:     output: $L_j$
7: else
8:     output: $L_{t+1}$
```

The space used of this algorithm is just t, for random oracle algorithms. For random seed algorithms, where we explicitly store the list L, we need space:[5]

$$t + \left\lceil \log \binom{n}{t+1} \right\rceil \leq (t+1)(1 + \log n) \leq (t+1) \log 2n$$
$$\leq (\frac{\log \frac{1}{\delta}}{\log \frac{n}{\ell}} + 1) \log 2n = \log 2n + \frac{\log 2n}{\log \frac{n}{\ell}} \log \frac{1}{\delta} .$$

□

3.2.2 Using sparse recovery

The performance of Algorithm 3.2.1 grows worse as ℓ approaches n; for $\ell = n-1$, $\log(n/\ell) \approx \frac{\ln 2}{n-1}$, in which case the space usage of Algorithm 3.2.1 is $\Theta(n)$. We will now describe an algorithm that is guaranteed to be efficient even for $\ell = n-1$. Let $\mathcal{I}$ be the set of integers included in the stream. The set of integers in $[n]$ which occur with frequency ≥ 2 in the stream has size $\leq \ell - |\mathcal{I}|$, while the set of integers with frequency zero has size $= n - |\mathcal{I}|$. Thus, if we pick a random integer in $[n]$ which was in the stream with a frequency other than 1, it will have frequency zero with probability $\geq \frac{n-|\mathcal{I}|}{n-|\mathcal{I}|+(\ell-|\mathcal{I}|)} \geq \frac{1}{2}$. Using a variant of an L_0 sampling sketch, like that in [JST11], picking a uniformly random element with frequency other than 1 can be done efficiently.

First, we note that while the linear sketch in [JST11] only reveals a *coordinate* i of the input

[5]When $\ell/\delta \ll n$, it is possible to improve the asymptotic space usage slightly by pretending that $n = \lceil \ell/\delta \rceil$; then the algorithm uses $O(\log \frac{\ell}{\delta})$ space.

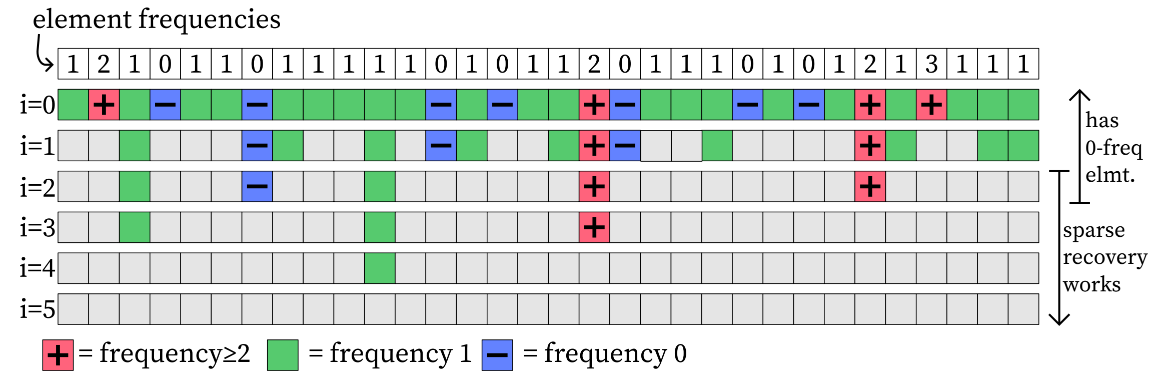


Figure 3.3: Because $\geq \frac{1}{2}$ of the elements in $[n]$ with frequency $\neq 1$ in the stream have frequency 0, if we pick random subsets of $[n]$ of progressively smaller sizes, one of them will likely both: contain few elements of frequency $\neq 1$; and contain at least one element of frequency 0. In the above diagram, each box on the horizontal axis corresponds to a different integer in [32]. If in the diagram shown, Algorithm 3.2.2 could only sparse-recover vectors of support ≤ 4, then it would report the zero-frequency element (number 7 from the left) at filtering level $i = 2$.

vector x where x_i is nonzero, the underlying sparse recovery sketch also provides the value of x_i itself. Thus, with only light modifications to [JST11], we obtain the following theorem. A diagram which may help explain the algorithm is given in Figure 3.3.

Theorem 3.2.2. *Algorithm 3.2.2 is a random seed streaming algorithm solving* $\textsc{mif}(n, n-1)$ *with tracking error* $\leq \delta$ *in the static setting. It uses* $O((\log n)^2 \log \frac{\log n}{\delta})$ *bits of space.*

First, we give a sparse recovery lemma:

Lemma 3.2.3. *For* $s < n$*, there is a deterministic linear sketch* $A \in \mathbb{Z}^{d \times n}$ *to exactly recover a vector* x *in* $\mathbb{Z}^n$ *satisfying* $\|x\|_1 \leq n$ *and* $\|x\|_0 \leq s$*. The matrix* A *has dimension* $d = O(\frac{s \log \frac{2n}{s}}{\log n})$ *and entries in* $\{0, \ldots, n\}$*.*

Even when applied to dense vectors $y \in \mathbb{Z}^n$ satisfying $\|y\|_1 \leq n$, the entries of Ay will be in $[-n^3, n^3]$, so Ay can be encoded using $O(s \log \frac{2n}{s})$ bits.

This is not a practical sketch for sparse recovery of integer vectors. However, [IR08] provide one with $\{0, 1\}$ matrix entries which, when applied to integer vectors, would have slightly larger dimension $d = O(s \log \frac{2n}{s})$, and which has a polynomial time decoder.

Proof of Lemma 3.2.3. Let $K \subseteq \mathbb{Z}^n$ be the set of vectors with ℓ_1-norm $\leq n$, and ℓ_0 norm $\leq s$. Let $K_2 = K - K := \{x - y : x, y \in K\}$. A matrix $A \in \mathbb{Z}^{d \times n}$ can be used to exactly recover all vectors in K if for all $y \in K_2 \setminus \{\vec{0}\}$, $Ay \neq \vec{0}$.

We prove a valid A exists by the probabilistic method. Say the entries of A are independently randomly chosen in $\{0, \ldots, n\}$. For any nonzero $y \in K_2$, if we fix all entries of A but those on a column corresponding to a nonzero coordinate i of y, we obtain:

$$\Pr[Ay = \vec{0}] \le \prod_{j=1}^{d} \Pr[A_{j,i} y_i = -\sum_{k \ne i} A_{j,k} y_k] \le (1/(n+1))^d \,.$$

Counting the number of nonzero entries, the signs, and the magnitudes of entries of vectors in K_2:

$$|K_2| \le \binom{n}{2s} 2^{2s} \binom{2n+2s-1}{2s} < 2^{4s\log(en\sqrt{2}/s)} \,.$$

With $d = \left\lceil \frac{4s\log(en\sqrt{2}/s)}{\log n} \right\rceil$, by a union bound the probability that A fails to recover all vectors in K is < 1, so in particular there must exist at least one valid value for A. □

Algorithm 3.2.2 A random-seed streaming algorithm for $\textsc{mif}(n, n-1)$ with $\le \delta$ error

```
    Let $s = \left\lceil 48 \ln \frac{5 \log n}{\delta} \right\rceil$
    Define: $\textsc{lsb} : \{0,1\}^{\lceil \log n \rceil} \to \{0, \ldots, \lceil \log n \rceil\}$ to give the 0-base-index of the first coordinate of
    its input which is 1, or $\lceil \log n \rceil$ if there are none.

    Initialization:
 1: Let $h : [n] \to \{0,1\}^{\lceil \log n \rceil}$ be an $s$-wise independent hash function
 2: for $i = 0, \ldots, \lceil \log n \rceil$ do
 3:     Let $D_i = \{j : \textsc{lsb}(h(j)) \ge i\}$                ▷ This ensures $D_0 = [n] \supseteq D_1 \ldots \supseteq D_{\lceil \log n \rceil}$
 4:     Let $\mathcal{A}_i$ be a linear sketch to recover integer vectors in $\mathbb{Z}^n$ with $\ell_1$ norm $\le n$ and $\ell_0$ norm
    $\le s$, using e.g. Lemma 3.2.3
 5:     Initialize $\mathcal{A}_i$ on the vector which is $-1$ on $D_i$ and 0 elsewhere

    Update($e \in [n]$):
 6: for $i = 0, \ldots, \textsc{lsb}(h(e))$ do
 7:     Update $\mathcal{A}_i$ to increase the frequency of $e$ by 1

    Query:
 8: for $i = \lceil \log n \rceil, \ldots, 0$ do
 9:     Let $y_i \in \mathbb{Z}^n$ be the recovered vector from $\mathcal{A}_i$
10:     if $\exists j \in D_i : y_i = -1$ then
11:         return: $j$
12: abort
```

We will use the following lemma from [SSS95], which gives concentration bounds for sums of random variables with limited independence; this concentration behavior makes it possible to use a random hash function to construct the chain of subsets $D_0 \supseteq D_1 \supseteq \ldots \supseteq D_{\lceil \log n \rceil}$ that we use to sample elements.

Lemma 3.2.4 (Concentration under limited independence, [SSS95, Theorem 5, part III])**.** *Let X be a sum of k-wise independent random variables, each of which is confined to $[0,1]$, and $\mu = E[X]$.*

If $\delta \geq 1$ *and* $k \geq \lceil \delta\mu \rceil$, *then* $\Pr[|X-\mu| \geq \delta\mu] \leq e^{-\delta\mu/3}$.

Proof of Theorem 3.2.2. For each $t = 1, \ldots, n-1$, let $x^{(t)} \in \mathbb{Z}^{[n]}$ be the frequency vector of the stream after t elements have arrived, recording the number of times each integer was present. Let $y^{(t)} = x^{(t)} - \vec{1}$, the vector that Algorithm 3.2.2 applies sketches to. Since $\sum_{i=1}^{n} x_i^{(t)} \leq t \leq n-1$, y has at least one coordinate which is -1. Also note that the set $N^{(t)} := \{i \in [n] : y_i^{(t)} = -1\}$ is larger than the set $P^{(t)} = \{i \in [n] : y_i^{(t)} \geq 1\}$. Since $x^{(t)}$ only ever increases with time, we have $N^{(1)} \supseteq N^{(2)} \ldots \supseteq N^{(n-1)}$. On the other hand, $P^{(1)} \subseteq \ldots \subseteq P^{(n-1)}$.

Let $i_\star = \max(0, \left\lceil \log(\frac{8|P^{(n-1)}|}{s}) \right\rceil)$. We will bound the probability that $P^{(n-1)}$ contributes more than $s/2$ nonzero coordinates to the region $D_{i_\star}$; in other words, that $|D_{i_\star} \cap P^{(n-1)}| \leq s/2$. As $D_{i_\star+1} \subseteq D_{i_\star}$, this event of course implies that for all $i \geq i_\star$, $|D_i \cap P^{(n-1)}| \leq s/2$. For each $j \in P^{(n-1)}$, let $X_j = \mathbb{1}_{j \in D_{i_\star}}$. Since h is drawn from an s-wise independent hash family, the $\{X_j\}_{j \in P^{(n-1)}}$ are s-wise independent. By linearity of expectation,

$$\mathbb{E}[D_{i_\star} \cap P^{(n-1)}] \sum_{j \in P^{(n-1)}} X_j = 2^{-i_\star}|P^{(n-1)}| \leq \frac{s}{8|P^{(n-1)}|}|P^{(n-1)}| \leq \frac{s}{8}\,.$$

Applying Lemma 3.2.4,

$$\Pr[|D_{i_\star} \cap P^{(n-1)}| > s/2] \leq \Pr[\sum_{j \in P^{(n-1)}} X_j \geq 2\mathbb{E}[\sum_{j \in P^{(n-1)}} X_j] \leq \exp(-\frac{1}{3}\frac{s}{8}) \leq \exp(-s/24)\,. \tag{3.1}$$

Next, we will bound the probability of the event that there exists some time t at which there is no value $j \geq i_\star$ for which $0 < |N^{(t)} \cap D_j| \leq s/2$. We first address a special case: if $|N^{(t)}| \leq s/8$, then $|P^{(n-1)}| \leq |P^{(t)}| < |N^{(t)}|$, so $i_\star = 0$. Then setting $j = 0$ will always work, because $0 < |N^{(t)}| = |N^{(t)} \cap D_0| \leq s/8 < s/2$. To handle the general case, we will bound, for each level $j \geq i_\star$, the probability that $0 < |N^{(t)} \cap D_j| \leq s/2$ fails to hold for any t for which $\frac{s2^j}{16} \leq |N^{(t)}| \leq \frac{s2^j}{8}$. Each time $t \in [n-1]$ with $|N^{(t)}| > s/8$ will be covered by the level $j_t = \left\lceil \log \frac{8|N^{(t)}|}{s} \right\rceil$; note $j_t \geq i_\star$.

For a given $j \geq i_\star$, define $N_{j\uparrow}$ to be the largest set in the sequence $N^{(1)}, \ldots, N^{(n-1)}$ which has size $\leq \frac{s2^j}{8}$, and define $N_{j\downarrow}$ to be the smallest set which has size $\geq \frac{s2^j}{16}$. Then for each t for which $\frac{s2^j}{16} \leq |N^{(t)}| \leq \frac{s2^j}{8}$, we have $N_{j\downarrow} \subseteq N^{(t)} \subseteq N_{j\uparrow}$. Thus if $|N_{j\uparrow} \cap D_j| \leq s/2$, and $|N_{j\downarrow} \cap D_j| > 0$, we will also have $0 < |N^{(t)} \cap D_j| \leq s/2$.

Since $\mathbb{E}|N_{j_\uparrow} \cap D_j| \leq \frac{s2^j}{8} \cdot 2^{-j} = s/8$, applying Lemma 3.2.4 gives

$$\Pr[|N_{i_\uparrow} \cap D_j| \geq s/2] \leq \exp(-\frac{1}{3}\frac{s}{8}) = \exp(-s/24)\,, \tag{3.2}$$

and since $\mathbb{E}|N_{i_\downarrow} \cap D_j| \geq \frac{s2^j}{16} \cdot 2^{-j} = s/16$, applying Lemma 3.2.4 gives

$$\Pr[|N_{i_\downarrow} \cap D_j| \leq 0] \leq \exp(-\frac{1}{3}\frac{s}{16}) = \exp(-s/48)\,. \tag{3.3}$$

Taking a union bound over the probability that $|D_{i_\star} \cap P^{(n-1)}|$ is too large (Eq. 3.1) or that for any $j \geq i_\star$, that $|N_{j_\uparrow} \cap D_j| \geq s/2$ (Eq. 3.2) or $|N_{j_\downarrow} \cap D_j| \leq 0$ (Eq. 3.3), we find that the probability of any of these events occurring is

$$\exp(-s/24) + \lceil \log n \rceil (\exp(-s/24) + \exp(-s/48)) \leq 5 \log n \exp(-s/48) \leq \delta \,.$$

If none of these bad events occur, then Algorithm 3.2.2 will always succeed, because the *for* loop on Lines 8 to 11 will, before it reaches some i where $|D_i \cap (P^{(t)} \cup N^{(t)})| \geq s$ and the sparse recovery sketch might fail, reach a value i where $D_i \cap N^{(t)} \neq \emptyset$. Then the algorithm successfully reports an element which is not in the stream. Thus, the probability that the algorithm fails at any point in the stream is $\leq \delta$.

Algorithm 3.2.2 uses $O(\log n)$ linear sketches, each of whose data can be encoded using $O(s \log n)$ bits of space. The hash function h can be drawn from the s-wise-independent hash family [WC81] of degree $\leq s$-polynomials over the finite field $\mathbb{F}_{2^{\lceil \log n \rceil}}$ and can be encoded using $O(s \log n)$ space. In total, the algorithm uses $O((\log n)^2 \log \frac{\log n}{\delta})$ space. □

3.3 Random oracle space complexity, adversarial setting

3.3.1 Introducing AVOID

Most of the lower bounds for MissingItemFinding use, as a key step, a reduction from the following one-way communication problem.

Definition 3.3.1. Let $\text{AVOID}(t, a, b)$ denote the following one-way communication game.

- Alice is given $S \subseteq [t]$ with $|S| = a$;
- Bob must produce $T \subseteq [t]$ with $|T| = b$ for which T is disjoint from S.

Strictly speaking, this is not so much a communication problem as an encoding-decoding problem; Alice must encode a set T disjoint from the given set S.[6] The lower bound will follow by a counting argument.

Lemma 3.3.2. *The public-coin δ-error communication complexity of* $\text{AVOID}(t, a, b)$ *is bounded thus:*

$$\mathrm{R}^{\to}_{\delta}(\text{AVOID}(t,a,b)) \geq \log(1-\delta) + \log\left(\binom{t}{a} \Big/ \binom{t-b}{a}\right) \tag{3.4}$$

$$> \log(1-\delta) + ab/(t \ln 2)\,. \tag{3.5}$$

[6]On the other hand, an almost equivalent formulation of $\text{AVOID}(t, a, b)$ more closely matches the one-way communication framework. In this variation, Bob is given $T' \subseteq [t]$ of size $b-1$, and must output some $i \in [t] \setminus S \setminus T'$. The deterministic (multi-round) communication complexity of this is $O(\log n)$, using the protocol from [KN97, Solution to exercise 5.10][7].

[7]The protocol is correct, but the solution makes an off-by-one error calculating the number of bits used.

Proof of Lemma 3.3.2. Let Π be a δ-error protocol for $\textsc{avoid}(t,a,b)$ and let $d = \mathrm{cost}(\Pi)$ (see Section 2.1 for the definition). Since, for each input $S \in \binom{[t]}{a}$, the error probability of Π on that input is at most δ, there must exist a fixing of the random coins of Π so that the resulting deterministic protocol Π' is correct on all inputs in a set

$$\mathcal{C} \subseteq \binom{[t]}{a}, \quad \text{with } |\mathcal{C}| \geq (1-\delta)\binom{t}{a}.$$

The protocol Π' is equivalent to a function $\phi\colon \mathcal{C} \to \binom{[t]}{b}$ where

- the range size $|\operatorname{image}(\phi)| \leq 2^d$, because $\mathrm{cost}(\Pi) \leq d$, and
- for each $S \in \mathcal{C}$, the value $T := \phi(S)$ is a correct output for Bob, i.e., $S \cap T = \emptyset$.

For any fixed $T \in \binom{[t]}{b}$, the set of all $S \in \binom{[t]}{a}$ for which S is disjoint from T is precisely the set $\binom{[t]\setminus T}{a}$. The cardinality of this set is exactly $\binom{t-b}{a}$. Thus, for any subset $\mathcal{D}$ of $\binom{[t]}{b}$, it holds that $\left|\mathcal{C} \cap \phi^{-1}(\mathcal{D})\right| \leq \binom{t-b}{a}|\mathcal{D}|$. Consequently,

$$(1-\delta)\binom{t}{a} \leq |\mathcal{C}| = |\phi^{-1}(\operatorname{image}(\phi))| \leq \binom{t-b}{a}|\operatorname{image}(\phi)| \leq \binom{t-b}{a}2^d,$$

which, on rearrangement, gives Eq. 3.4.

To obtain Eq. 3.5, we note that

$$\begin{aligned}\binom{t}{a}\Big/\binom{t-b}{a} &= \frac{t!a!(t-a-b)!}{(t-a)!a!(t-b)!} = \frac{t\cdot(t-1)\cdots(t-a+1)}{(t-b)\cdot(t-b-1)\cdots(t-a-b+1)} \\ &\geq \left(\frac{t}{t-b}\right)^a = \left(\frac{1}{1-b/t}\right)^a > e^{ab/t}, \end{aligned} \tag{3.6}$$

which implies

$$\log(1-\delta) + \log\left(\binom{t}{a}\Big/\binom{t-b}{a}\right) > \log(1-\delta) + ab/(t\ln 2). \qquad \square$$

The lower bound in Lemma 4.3.1 is close to being optimal. We describe a randomized protocol whose space requirement is larger by only a function of δ:

Lemma 3.3.3. *For any $t \in \mathbb{N}$, $0 < a+b \leq t$, and $\delta \in (0,1)$, the randomized complexity of solving $\textsc{avoid}(t,a,b)$ with $< \delta$ error is bounded thus:*

$$\mathrm{R}^{\rightarrow}_{\delta}(\textsc{avoid}(t,a,b)) \leq \log\left(\binom{t}{a}\Big/\binom{t-b}{a}\right) + \ln\frac{1}{\delta} + 1. \tag{3.7}$$

Proof of Lemma 3.3.3. Let $z := \left\lceil \left(\binom{t}{a}/\binom{t-b}{a}\right)\ln\frac{1}{\delta}\right\rceil$. Using public randomness, choose an ordered collection $\mathcal{R}$ of z subsets of $[t]$ of size b. Alice's protocol is, given a set $S \in \binom{[t]}{a}$, to send the index j of the first set T in $\mathcal{R}$ which is disjoint from S. If there is no such set, Alice sends an arbitrary value (or aborts). Bob in turn returns the jth element of $\mathcal{R}$.

For any $S \in \binom{[t]}{a}$, define $\mathcal{O}_S$ to be the set of subsets in $\binom{[t]}{b}$ which are disjoint from S; observe that $|\mathcal{O}_S| = \binom{t-a}{b}$. Then the protocol will succeed if $\mathcal{R}$ overlaps with $\mathcal{O}_S$. The failure probability is:

$$\begin{aligned}
\Pr\left[\mathcal{R} \cap \mathcal{O}_S = \varnothing\right] &= \Pr\left[\mathcal{R} \in \binom{\binom{[t]}{b} \setminus \mathcal{O}_S}{z}\right] \\
&= \left(\binom{\binom{t}{b} - \binom{t-a}{b}}{z} \Big/ \binom{\binom{t}{b}}{z}\right) \\
&< \exp\left(-z\binom{t-a}{b} \Big/ \binom{t}{b}\right) && \triangleleft \text{ by Eq. 3.6} \\
&= \exp\left(-z\binom{t-b}{a} \Big/ \binom{t}{a}\right) \le \exp(-\ln\frac{1}{\delta}) = \delta\,.
\end{aligned}$$

The number of bits needed to encode an integer in $[z]$ is:

$$\begin{aligned}
\lceil \log z \rceil = \left\lceil \log \left\lceil (\binom{t}{a} / \binom{t-b}{a}) \ln\frac{1}{\delta} \right\rceil \right\rceil &= \left\lceil \log\,(\binom{t}{a} / \binom{t-b}{a}) \ln\frac{1}{\delta} \right\rceil \\
&\le \log\left(\binom{t}{a} / \binom{t-b}{a}\right) + \log\ln\frac{1}{\delta} + 1\,. \qquad \square
\end{aligned}$$

If we apply Lemma 3.3.3 with $\delta = \binom{t}{a}$, we obtain a protocol with $< 1/\binom{t}{a}$ probability of failure; by a union bound, it has < 1 probability of failing on any input. By the probabilistic method, it follows that there exists a deterministic algorithm for AVOID(t, a, b) using

$$\le \log\left(\binom{t}{a} \Big/ \binom{t-b}{a}\right) + \log\ln\binom{t}{a} + 1$$

bits of space. An often more convenient upper bound on the binomial is:

$$\begin{aligned}
\binom{t}{a} \Big/ \binom{t-b}{a} = \frac{t!(t-a-b)!}{(t-a)!(t-b)!} &= \frac{t\cdots(t-b+1)}{(t-a)\cdots(t-a-b+1)} \\
&= \prod_{i=0}^{b}\left(1 + \frac{a}{t-a-i}\right) \\
&\le \left(1 + \frac{a}{t-a-b+1}\right)^b \le \exp\left(\frac{ab}{t-a-b+1}\right)\,.
\end{aligned}$$

More formally:

Lemma 3.3.4. *For any $t \in \mathbb{N}$, $0 < a + b \le t$, the deterministic communication complexity of solving* AVOID(t, a, b) *is:*

$$\le \log\left(\binom{t}{a} \Big/ \binom{t-b}{a}\right) + \log\ln\binom{t}{a} + 1 \le \frac{ab}{(t-a-b+1)\ln 2} + \log t + 1\,. \tag{3.8}$$

Equivalently, for z where $\lceil \log z \rceil$ is bounded by Eq. 3.8, there exists a collection $R_1, \ldots, R_z$ of *subsets of $[t]$ of size b each so that, for any $S \in \binom{[t]}{a}$, there exists an $i \in [z]$ with $R_i \cap S = \emptyset$.*

We do not know if this result for deterministic algorithms is optimal, or whether one can outperform the current randomized construction and avoid the additional $\log \ln \binom{t}{a}$ factor.[8]

If one takes the complement of each of the subsets $R_1, \ldots, R_z$ from Lemma 3.3.4, one obtains a $(t, t-b, a)$-covering design, using the notation of [GPK95].[9]. In general, a (v, k, t)-covering design is a collection of subsets of $[v]$ of size k, so that every set of size t is contained by at least one of the subsets. The minimum size $C(v, k, t)$ of course has a lower bound of $\geq \binom{v}{t}/\binom{k}{t}$ (by the same counting argument as for Lemma 3.3.2), although [GPK95] observe that there are slightly stronger lower bounds; the best known general upper bound for $C(v, k, t)$ also uses the probabilistic method to get $C(v, k, t) \leq (1 + \ln \binom{k}{t})\binom{v}{t}/\binom{k}{t}$. A covering design in which each set of size t is included in *exactly one* set of size k would be ideal; these are known as Steiner systems. However, our algorithm for Algorithm 3.5.2 in Section 3.5 would not be able to use them: it uses the regime $t \approx v/2$, $k \approx (1-\varepsilon)v$, where any two subsets in the covering design will contain a set of size t in their intersection; thus such a Steiner system can not exist for $\textsc{avoid}(v, v/2, \varepsilon v)$ protocols.

3.3.2 Lower bound: reduction from AVOID

This lower bound uses a trick that is common to many of the $\textsc{mif}$ lower bounds: if an algorithm for $\textsc{mif}$ is adversarially robust with tracking error δ, one can (with $\geq 1-\delta$ probability) recover not just one, but any number $t \leq \ell$ distinct new elements by repeatedly querying the algorithm for an output, and passing that output back to the algorithm as its next input.

Theorem 3.3.5. *Any algorithm which solves $\textsc{mif}(n, \ell)$ against adaptive adversaries with error δ requires $\geq \log(\binom{n}{\lceil \ell/2 \rceil}/\binom{n-\lceil \ell/2 \rceil}{\lfloor \ell/2 \rfloor + 1}) + \log(1-\delta)$ bits of space; or less precisely, $\Omega(\ell^2/n + \log(1-\delta))$.*

Proof of Theorem 3.3.5. We prove this by reducing the communication task $\textsc{avoid}(n, \lceil \ell/2 \rceil, \lfloor \ell/2 \rfloor + 1)$ to $\textsc{mif}(n, \ell)$.

Say Alice is given the set $A \subseteq [n]$ of size $\lceil \ell/2 \rceil$. Alice instantiates an instance $\mathcal{X}$ of the given algorithm for $\textsc{mif}(n, \ell)$, and runs it on the partial stream of length $\lceil \ell/2 \rceil$ containing the elements of A in some arbitrary order. Alice then sends the state of $\mathcal{X}$ to Bob; since this is a public coin protocol, all randomness can be shared for free. Bob then runs the following adversary against $\mathcal{X}$; it queries $\mathcal{X}$ for an element b_0, and then sends that element back to $\mathcal{X}$ to be its next input, repeating this process $\lfloor \ell/2 \rfloor + 1$ times to recover elements $b_0, b_1, \ldots, b_{\lfloor \ell/2 \rfloor}$. The instance will fail to give correct answers to this adversary with total probability $\leq \delta$. If it succeeds, then by the definition of the Missing Item Finding problem, $b_0 \notin A$, $b_1 \notin \{b_0\} \cup A$, $b_2 \notin \{b_0, b_1\} \cup A$, and so on;

[8]There *is* a simple lower bound of $\geq \log(a+1)$, which exactly matches the upper bound if $(a+1)b \leq t$. This suggests that it may be possible to shave the $\log \log t$ part of $\log \ln \binom{t}{a} = O(\log a + \log \log t)$.

[9]The book [CD07] uses the name t-$(v, k, 1)$-covering for (v, k, t)-covering designs

thus Bob can report $B := \{b_0, \ldots, b_{\lfloor \ell/2 \rfloor+1}\}$ as a set of $\lfloor \ell/2 \rfloor + 1$ elements which are disjoint from A.

This AVOID protocol implementation uses the same number of bits of communication as $\mathcal{X}$ does of space. By Lemma 3.3.2, it follows $\mathcal{X}$ needs space:

$$\begin{aligned} &\geq \log\left(\binom{n}{\lceil \ell/2 \rceil} \Big/ \binom{n - \lfloor \ell/2 \rfloor - 1}{\lceil \ell/2 \rceil}\right) + \log(1-\delta) \\ &\geq \frac{\lceil \ell/2 \rceil (\lfloor \ell/2 \rfloor + 1)}{n \ln 2} + \log(1-\delta) \geq \frac{\ell^2}{4n \ln 2} + \log(1-\delta)\,. \end{aligned} \qquad \square$$

3.3.3 Upper bound: the hidden list algorithm

In this section we describe a random oracle algorithm for MIF(n, ℓ) in the adversarial setting, which uses its oracle-type access to random bits to keep track of a random list L of outputs that it might give, but which the adversary cannot easily learn. The design is simple: Algorithm 3.3.1 uses oracle randomness to pick a list L of $\ell + 1$ distinct elements uniformly at random, and then on being queried outputs the first element of L which is still available. Because the list L is chosen uniformly at random, the adversary has no significant advantage in predicting the available elements of the list. There is one exception: the adversary always knows what the available element with the lowest index is, since this is what the algorithm outputs output. As a result, the adversary at each point only has a choice between an "echo" strategy of returning the algorithm's last output, and a "guess" strategy of picking some new random element in $[n]$.[10] Algorithm 3.3.1 only needs space to record which elements of L were already in the input; to handle the echo strategy, it keeps a counter which marks a prefix of L that has been used; and to handle the guess strategy, it stores a set J of correctly guessed elements in L (which one can prove will have size $O(1 + \ell^2/n)$ with high probability).

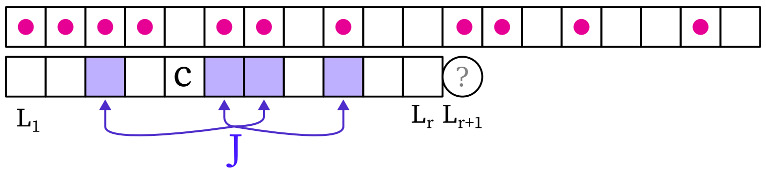


Figure 3.4: This diagram shows the behavior of Algorithm 3.3.1 on an example input. The top row of squares corresponds to the set $[n]$, ordered so that the leftmost squares corresponds to the elements L_1, L_2, $\ldots$, $L_{\ell+1}$ from Algorithm 3.3.1. In the top row, cells contain a pink dot if the corresponding element has already been seen in the stream. In the bottom row, the letter C indicates the cell corresponding to L_c. Cells that are shaded dark blue indicate the values contained in J. The third cell from the left is included in J because, at the time the element L_3 was added by the adversary, c was less than or equal to 2.

[10]Against this algorithm, an adversary has nothing to gain by repeating an input, so we can assume all inputs are distinct.

Theorem 3.3.6. *Algorithm 3.3.1 solves* MIF(n, ℓ) *against adaptive adversaries, with error* δ*, and can be implemented using* $O(\min(\ell, \left(1 + \frac{\ell^2}{n} + \ln\frac{1}{\delta}\right) \cdot \log \ell))$ *bits of space. (It requires oracle access to* $(\ell + 1)\log n$ *random bits.)*

Algorithm 3.3.1 Adversarially robust, random oracle algorithm for MIF(n, ℓ) with error $\le \delta$

```
    Let $t = \min(\ell, \left\lceil 3\frac{\ell^2}{n} + \ln\frac{1}{\delta} \right\rceil)$

    Initialization:
 1: Let $L = \{L_1, \dots, L_{\ell+1}\}$ be a fixed sequence of elements in $[n]^{\ell+1}$ without repetitions, chosen
    uniformly at random.   ▷ With oracle access to $O(\ell \log n)$ random bits, the value of $L$ can be
    computed on demand, instead of stored.
 2: $c \leftarrow 1$, an integer in the range $\{1, \dots, \ell + 1\}$
 3: $J \leftarrow \emptyset$, a subset of $\{L_1, \dots, L_\ell\}$ of size $\le t$

    Update($e \in [n]$):
 4: while $e = L_c$ or $L_c \in J$ do
 5:     $c \leftarrow c + 1$
 6: if $e \in \{L_{c+1}, \dots, L_\ell\}$ then
 7:     $J \leftarrow J \cup \{e\}$
 8: if $|J| > t$ then
 9:     abort

    Query:
10: output: $L_c$
```

Proof of Theorem 3.3.6. First, we observe that the only way that Algorithm 3.3.1 can fail is if it aborts. At any point in the stream, the set J includes the intersection of the earlier elements from the stream, with the list $\{L_{c+1}, \dots, L_\ell\}$ of possible future outputs. The while loop ensures that the element L_c emitted will neither be equal to the current element nor collide with any past stream elements (those in J). It is not possible for c to go out of bounds, because each element in the stream can lead to an increase in c of at most one; either immediately when the element arrives, if $e = L_c$; or delayed slightly, if $e \in \{L_{c+1}, \dots, L_\ell\}$. Since the stream contains ℓ elements, c will increase by at most ℓ, to a value of $\ell + 1$. Note that if c has reached the value $\ell + 1$, then the entire stream was a permutation of $\{L_1, \dots, L_\ell\}$, making $L_{\ell+1}$ is a safe output.

This algorithm needs $\log(\ell + 1)$ bits to store c, but the main space usage is in storing J. We will show that $|J| \le t$ with probability $\ge 1 - \delta$, in which case J can be stored as either a bit vector of length ℓ, or a list of $\le t$ indices in $[\ell]$, using $O(\min(\ell, t \log \ell))$ bits of space.

We observe that after $i - 1$ elements have been received (and up to i distinct elements emitted), the probability that the ith element chosen by the adversary will be newly stored in J will be $\le 2\frac{\ell}{n}$, no matter what the earlier elements were or what the adversary picks. If $\ell \ge n/2$, this is immediate. Otherwise, write E_{i-1} for the set containing the first $i-1$ elements of the stream, e_i for the ith element, and let c_i be the value of the variable c as of Line 6. Let X_i denote the indicator random variable for the event that e_i was not in J before, but has been added now.

Because the adversary has only been given outputs deriving from $L_{\le c_i} := (L_1, \ldots, L_{c_i})$, if we condition on the random variable $L_{\le c_i}$, then the element e_i and set E_{i-1} are independent of $L_{>c_i} := \{L_{c_i+1}, \ldots, L_\ell\}$. Given E_{i-1}, the values $X_1, \ldots, X_{i-1}$ determine whether or not each element of E_{i-1} is in $L_{>c_i}$. Then, conditioning on $L_{\le c_i}, e_i, E_{i-1}$, and $X_1, \ldots, X_{i-1}$, we have that $L_{>c_i} \setminus E_{i-1}$ is a set of size $\ell - c_i - |L_{>c_i} \cap E_{i-1}|$ chosen uniformly at random from $[n] \setminus L_{\le c_i} \setminus E_{i-1}$. Thus, if $e_i \notin L_{\le c_i} \cup E_{i-1}$, the probability that $X_i = 1$ is precisely the probability that e_i is contained in $L_{>c_i} \setminus E_{i-1}$, so:

$$\begin{aligned}\Pr\left[X_i = 1 \mid (X_j)_{j=1}^{i-1}, e_i, E_{i-1}, L_{\le c_i}, \{e_i \notin L_{\le c_i} \cup E_{i-1}\}\right] &= \frac{\ell - c_i - |L_{>c_i} \cap E_{i-1}|}{n - c_i - |E_{i-1} \setminus L_{\le c_i}|} \\ &\le \frac{\ell - c_i}{n - c_i - \ell} \le \frac{\ell}{n-\ell} \le \frac{2\ell}{n}.\end{aligned}$$

On the other hand, the event $e_i \in L_{\le c_i} \cup E_{i-1}$, implies $X_i = 0$ always. Together, these imply $\Pr[X_i = 1 \mid (X_j)_{j=1}^{i-1}] \le 2\ell/n$.

Then applying the (modified, see Lemma 2.3.1) Azuma's inequality bound, we find that with $z := \max\{1, \frac{3n}{2\ell^2} \ln \frac{1}{\delta}\}$:

$$\begin{aligned}\Pr[\sum_{i\in[\ell]} X_i \le \frac{2\ell^2}{n}(1+z)] &\le e^{-\frac{z}{2+z} z \frac{2\ell^2}{n}} \\ &\le e^{-z\frac{2\ell^2}{3n}} && \text{since } z \ge 1 \\ &\le e^{-\ln\frac{1}{\delta}} = \delta\,. && \text{since } z \ge \frac{3n}{2\ell^2}\ln\frac{1}{\delta}\end{aligned}$$

This implies that the probability that $|J|$ exceeds $2\ell^2/n + 3\ln(1/\delta)$ will be $\le \delta$. Consequently, our bound for the total space usage of the algorithm is:

$$\begin{aligned}&O(\log \ell) + O(\min(\ell, \left(\frac{\ell^2}{n} + \ln\frac{1}{\delta}\right)\log \ell)) \\ &= O(\min(\ell, \left(1 + \frac{\ell^2}{n} + \ln\frac{1}{\delta}\right) \cdot \log \ell))\,.\end{aligned}$$ □

While it is possible to reduce the space usage of Algorithm 3.3.1 by removing all elements from the set J that are less or equal than c, this only changes the constant factor.

3.4 Zero-error model variant

As in the adversarial setting (see Section 3.3.2), in the zero-error static setting one can iteratively extract from an algorithm $\ell/2$ distinct elements that were not in the original input.[11] We use this property to reduce AVOID to MIF, and obtain:

Theorem 3.4.1. *All algorithms solving* MIF(n,ℓ) *with zero error on any stream require* $\Omega(\ell^2/n)$ *bits of space, in expectation over the randomness of the algorithm.*

Proof of Theorem 3.4.1. First, we prove that if there is a zero-error algorithm Φ for MIF(n,ℓ) using exactly s bits, in expectation, then there is a communication protocol for AVOID$(n,\lceil\ell/2\rceil,\lfloor\ell/2\rfloor+1)$ using prefix-encoded messages with an expected length of s bits. The construction is the same as for Theorem 3.3.5. Alice, on being given a set $A\subseteq[n]$ of size $\lceil\ell/2\rceil$, initializes an instance X of Φ, and runs it on an input stream α of length $\lceil\ell/2\rceil$ containing each element of A in some arbitrary order. Any random bits used by X are shared publicly with Bob. Alice sends the encoding of X's state to Bob, who queries X to find an element $b_0\notin\alpha$, updates X with b_0, queries it to find $b_1\notin\alpha\cup\{b_0\}$, and so on until Bob has recovered $B=\{b_0,\ldots,b_{\lfloor\ell/2\rfloor}\}$. Because the algorithm is guaranteed to never fail on any input stream, it must in particular succeed on Bob's adaptively chosen continuation of α. This ensures that $B\cap A=\emptyset$ holds with probability 1.

Next, we prove that any zero error randomized communication protocol Π for AVOID(t,a,b) requires $\geq ab/(t\ln 2)$ bits *in expectation.* Following the argument of Lemma 3.3.2, we observe that there must exist a fixing of the public randomness of Π for which the expected number of bits used when inputs A are drawn uniformly at random from $\binom{[t]}{a}$, is at least as large as when Π is run unmodified. Let Υ be the deterministic protocol with this property, and let M be the set of all messages sent by Υ. Each message $m\in M$ has a length $|m|$, probability (over the random choice of A) p_m of being sent, and makes Bob output the set B_m. For all $m\in M$, we have:

$$p_m=\Pr[m\text{ is sent}]\leq\Pr[B_m\text{ is a correct output}]=\Pr[A\cap B_m=\emptyset]\leq\binom{t-a}{b}\Big/\binom{t}{a}\leq 2^{-\frac{ab}{t\ln 2}}\,.$$

Let $\Upsilon(A)\in M$ be the message sent by Υ for a given value of A. Then the entropy

$$H(\Upsilon(A))=\sum_{m\in M}p_m\log\frac{1}{p_m}=\mathbb{E}_{A\in\binom{[t]}{a}}\log\frac{1}{p_{\Upsilon(A)}}\geq\mathbb{E}_{A\in\binom{[t]}{a}}\frac{ab}{t\ln 2}=\frac{ab}{t\ln 2}\,.$$

[11]This trick works in the *zero-error static* setting, not just the zero-error adversarial setting, because the zero-error models ensure the algorithm output will always be correct. In the zero-error static setting, it is *space usage* that is only measured against fixed streams, not *correctness*; and the protocol for AVOID that our proof describes will prepare the algorithm state using a fixed stream, not an adversary, so the expected message length will be bounded by the zero-error static cost of the algorithm. If we had adaptively prepared the algorithm state for the message, this would not have worked.

By the source coding theorem [Sha48, Sec. 9],

$$\mathbb{E}_{A \in \binom{[t]}{a}} \mathbb{E}|\Upsilon(A)| \geq H(\Upsilon(A)) \geq \frac{ab}{t \ln 2}\,.$$

Applying the above lower bound to the task $\textsc{avoid}(n, \lceil \ell/2 \rceil, \lfloor \ell/2 + 1 \rfloor)$, we conclude that Φ requires $\geq \frac{\ell^2}{4n \ln 2}$ bits of space in expectation.

□

Theorem 3.4.2. *There is an algorithm solving* $\textsc{mif}(n, \ell)$ *with zero error against adaptive adversaries, which uses* $O((1 + \ell^2/n) \log \ell)$ *bits of space, in expectation over the randomness of the algorithm.*

Proof of Theorem 3.4.2. We use a slight variation of Algorithm 3.3.1, in which internal parameter t is instead set to ℓ. This ensures that the algorithm will never abort; the proof of Theorem 3.3.6 has established that Algorithm 3.3.1 will then always give a correct output for the $\textsc{mif}(n, \ell)$ task.

The counter c can be encoded in binary using at most $\lceil \log(\ell + 1) \rceil$ bits. We encode the set J by concatenating the binary value of $|J|$, followed by the binary values of the indices $i_1, \ldots, i_{|J|}$ in $[\ell]$ for which L_{i_k} is equal to the kth smallest element of J. (As both the encodings of c and J are prefix codes, so too is the encoding of the algorithm's state formed by concatenating them.) The total space S used by the algorithm (excluding random bits) is then:

$$S = \lceil \log(\ell + 1) \rceil + \lceil \log \ell \rceil (1 + |J|)\,.$$

As in the proof of Theorem 3.3.6, let X_i be the indicator random variable for the event that the ith element that the adversary chooses for the stream is stored in J; we showed that for all $i \in [\ell]$, $\Pr[X_i = 1 \mid X_{i-1}, \ldots, X_1] \leq \frac{\ell - 1}{n}$, which implies $\mathbb{E}[X_i] \leq \frac{\ell-1}{n}$. By linearity of expectation,

$$\begin{aligned} \mathbb{E}S &= \lceil \log(\ell + 1) \rceil + \lceil \log \ell \rceil \left(1 + \mathbb{E} \sum_{i \in [\ell]} X_i \right) \\ &\leq \lceil \log(\ell + 1) \rceil + \lceil \log \ell \rceil \left(1 + \frac{\ell(\ell - 1)}{n} \right) = O((1 + \frac{\ell^2}{n}) \log \ell)\,. \end{aligned}$$

□

Say one has a randomized streaming algorithm $\mathcal{A}$ in the static or adversarial settings, with the property that it either produces a correct output or aborts. Most of the algorithms presented in this chapter can be modified to do this. To convert $\mathcal{A}$ to a zero-error algorithm, one must only cover for the cases where $\mathcal{A}$ aborts, using some fallback method to ensure that a correct output is always made. Since the risk of aborting can often be driven very low with suitable parameter choices, the expected cost of this fallback is dominated by the expected cost in the event that the fallback does not need to be used. Using a lot of space in the event that $\mathcal{A}$ aborts is acceptable, as this has little impact on the expected space cost.

Theorem 3.4.3. *Let $\mathcal{B}$ be a zero-mistake (Definition 2.2.2) algorithm solving* MIF(n,ℓ) *with $\leq 1/\ell$ error in the static (or adversarial) settings, implemented as a random tape, seed, or oracle algorithm, with worst-case space usage S. Then there is a zero-error static (or adversarial) algorithm with the same randomness type solving* MIF(n,ℓ) *with zero error that uses at most $S + O((1+\ell^2/n)\log \ell)$ bits of space in expectation, on any input stream (or adversary).*

Algorithm 3.4.1 Extending a non-guessing algorithm $\mathcal{B}$ for MIF(n,ℓ) to have zero error

```
    Require: $\ell \leq n/32$, $\ell \geq 2$; otherwise use Algorithm 3.1.1
    Let $t = \max(\lceil 32\ell^2/n \rceil, \lceil 10 \log \ell \rceil)$
    Let $w = \min(\ell, \lfloor n/8\ell \rfloor)$

    Initialization:
 1: $B \leftarrow$ instance of $\mathcal{B}$ parameterized for MIF$(n,\ell)$
 2: For $i \in [8\ell]$, define $H_i := \{(i-1)w+1, \ldots, iw\}$
 3: $F \in_R \binom{[8\ell]}{t}$ be a random subset                 ▷ uses $O(t \log \ell)$ space
 4: $x \leftarrow (1,\ldots,1)$ be a vector in $\{0,1\}^F$
 5: ▷ $Y$ is stored using variable-length encoding, using $O(1+|Y|\log(tw))$ bits in total
 6: $Y \leftarrow \emptyset$ a subset of $F \times [w]$

    Update($e \in [n]$):
 7: if $B$ has not yet failed then
 8:     $B$.Update($e$)
 9: if $\exists j \in F$ for which $e \in H_j$ then
10:     if $\sum_{i \in F} x_i > \frac{t}{2}$ then
11:         $x_j \leftarrow 0$
12:     else
13:         Let $k = e - (j-1)w$
14:         $Y \leftarrow Y \cup \{(j,k)\}$

    Query:
15: if $B$ has not yet failed then
16:     return $B$.Query()
17: else
18:     Let $j,k \in F \times [w]$ be indices where $x_j = 1$ and $(j,k) \notin Y$.
19:     return $(j-1)w + k$
```

Proof of Theorem 3.4.3. We describe a new algorithm which uses $\mathcal{B}$ as a subroutine in Algorithm 3.4.1. The additional code requires only a random seed (not random tape or oracle). We assume (at the cost of using ≤ 1 additional bit) that $\mathcal{B}$ tracks whether or not it has failed, and only fails during its UPDATE step, not when its output is queried. The worst-case (over possible streams/adversaries) probability that $\mathcal{B}$ fails is $\leq 1/\ell$.

If the instance B of $\mathcal{B}$ does not fail, then Algorithm 3.4.1 will return correct output. If B does fail, the output of the algorithm will be determined by the additional logic (Lines 9 to 14 for the update, and Lines 18 to 19 for querying.) Let $E \subseteq [n]$ be the set of inputs. There are two cases: first, if E intersects $< t/2$ of the sets in $\{H_i\}_{i\in F}$, then the array x will be 1 at coordinate j iff $E \cap H_j = \emptyset$; and also $Y = \emptyset$. Then Line 18 will choose a value j for which $E \cap H_j = \emptyset$, so that for all possible values of $k \in [w]$, $(j-1)w + k \in H_j$ and hence the algorithm output is not in E. For the second case, we have that E intersects $\geq t/2$ of the sets in $\{H_i\}_{i\in F}$. Then for each $j \in F$, if $x_j = 1$, the set Y will contain coordinates for all integers which lie in $H_j \cap E$. Consequently, Line 18 will choose coordinates (j,k) for which the associated integer $(j-1)w+k$ is in $H_j \setminus E$. It is guaranteed that at least one such pair of coordinates exists, because there are $\geq t/2$ values in F with $x_j = 1$, and for each value there are w possible values of k; meanwhile,

$$wt/2 \geq \frac{1}{2}\min(\ell, \left\lfloor \frac{n}{8\ell} \right\rfloor)\max(4, \left\lceil \frac{32\ell^2}{n} \right\rceil) \geq \frac{1}{2}\begin{cases} \ell \cdot 4 & \ell \geq \sqrt{n/8} \\ \left\lfloor \frac{n}{8\ell} \right\rfloor \left\lceil \frac{32\ell^2}{n} \right\rceil & \text{otherwise} \end{cases} \geq 2\ell > \ell \geq |E|\,,$$

so E cannot possibly cover all pairs.

We now address the expected space usage of the algorithm. In any case, the variable B will use $S + O(1)$ bits of space; the variable F, $O(t \log \ell)$ bits; and the variable x, $O(t)$ bits. For Y, we again have two cases. Let L be the event that, at the end of the stream, the set E of all inputs intersects $\geq t/2$ of the sets in $\{H_i\}_{i\in F}$. If L does not occur, then the set Y will be empty, and can be encoded to add only $O(1)$ bits to the length of the space encoding. Otherwise, if L occurs, Y will in the worst case add $|F|w \cdot O(\log tw) = O(tw \log tw)$ bits of space.

We now bound the probability that the event L occurs *and* B does not fail. If B does not fail, then the algorithm will output using Line 16, and will *never* reveal the value of F to the algorithm. Consequently, the set E of inputs for the algorithm will be entirely independent of F. For all $i \in [8\ell]$, define $\{0,1\}$ random variable $X_i = \mathbb{1}_{i\in F}$. Note $\mathbb{E}[X_i] = t/8\ell$. Since the collection $\{X_i\}_{i\in[8\ell]}$ is negatively associated, by Lemma 2.3.3, no matter which collection of sets $\{H_i\}_{i\in A}$ is intersected by E:

$$\begin{aligned} \Pr[L \wedge B \text{ succeeds}] &\leq \max_{A\subseteq[8\ell],|A|\leq\ell} \Pr\left[\sum_{i\in A} X_i \geq t/2\right] \leq \max_A \exp\left(-\frac{3^2}{(2+3)(1+3)}t/2\right) \\ &\leq \exp\left(-\frac{9}{40}t\right) \leq \exp\left(-\frac{9}{40}\lceil 10\log\ell\rceil\right) \leq \frac{1}{\ell}\,. \end{aligned}$$

The expected space usage of the algorithm is, since $tw = O(\ell \log \ell)$

$$\begin{aligned}
&S + O(t \log \ell) + \Pr[L] O(tw \log tw) \\
&\quad \le S + O(t \log \ell) + (\Pr[L \wedge B \text{ succeeds}] + \Pr[B \text{ fails}]) O(tw \log tw) \\
&\quad \le S + O(t \log \ell) + \Big(\frac{1}{\ell} + \frac{1}{\ell}\Big) O(tw \cdot \log tw) \\
&\quad \le S + O\left(\left(\frac{\ell^2}{n} + \log \ell\right) \log \ell + \frac{1}{\ell} O(\ell \log \ell \cdot \log O(\ell \log \ell)\right) \\
&\quad \le S + O\left(\left(\frac{\ell^2}{n} + \log \ell\right) \log \ell\right) . \qquad \square
\end{aligned}$$

3.5 Deterministic space complexity

3.5.1 Lower bound: an embedded instance of AVOID

The following lower bound for deterministic algorithms for MIF(n, ℓ) works by showing that, if an algorithm uses too little space, then the set of possible future outputs must shrink geometrically as time progresses, leading to a contradiction once there are fewer possible outputs than the number of remaining inputs in the stream. For any partial stream $\sigma \in [n]^\star$ of length $|\sigma|$, let F_σ be the set of all possible outputs made at time ℓ for streams that start with with σ. A direct consequence of the definition is that if partial stream ρ is a prefix of σ, then $F_\rho \supseteq F_\sigma$. We also have the property that, given only the state v that the algorithm will be in after processing some $\gamma \in [n]^\star$, one can determine F_γ using *only* v and the value of $\ell - |\gamma|$ (and, of course, knowledge of the algorithm). Consequently, the number of possible output sets F_σ is limited by (mainly) the number of states of the algorithm.

The key observation of the proof is that, if an algorithm uses z bits of space, then for any $\sigma \in [n]^\star$ one can find a partial stream ρ which extends σ by $t = O(z)$ elements, for which $|F_\rho|/|F_\sigma| \le \frac{1}{2}$. This can be proven as a consequence of the lower bound for the AVOID communication problem (Definition 3.3.1); in short, using the algorithm we can construct a z-bit protocol for AVOID$(|F_\sigma|, t, \frac{1}{2}|F_\sigma|)$, wherein Alice sends a message encoding a state of the algorithm and Bob computes the set of possible outputs for that state. By Lemma 3.3.2, this protocol must use $z = \Omega(t)$ bits of space.

Theorem 3.5.1. *Every deterministic streaming algorithm for* MIF(n, ℓ) *requires* $\Omega(\sqrt{\ell} + \frac{\ell}{1+\log(n/\ell)})$ *bits of space.*

Proof of Theorem 3.5.1. Let Σ be the set of states of the algorithm, and let s_{init} be the initial state. Let $\tau : \Sigma \times [n]^\star \mapsto \Sigma$ be the (iterated) transition function of the algorithm, where $\tau(s, (e_1, \ldots, e_k)) = x$ means that if the algorithm is at state s, and the next k elements in the stream are $e_1, \ldots, e_k$, then after processing those elements the algorithm will reach state x. For each partial stream $\sigma \in [n]^\star$, abbreviate $\tau(s_{\text{init}}, \sigma)$ as $\Sigma[\sigma]$. For each state $s \in \Sigma$, we associate the output $\omega_s \in [n]$ which the

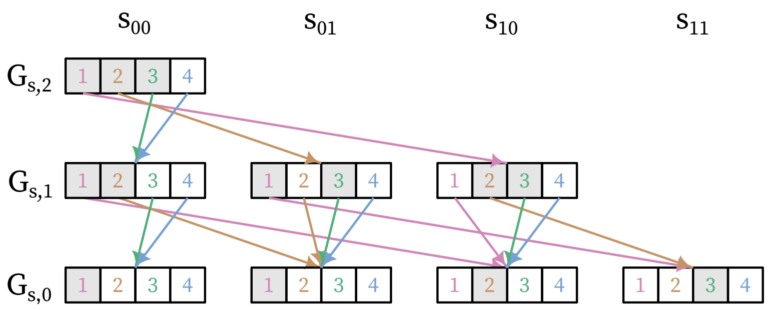


Figure 3.5: In the proof of Theorem 3.5.1, the quantities F_σ (defined in Eq. 3.9) are entirely determined by the values of $\Sigma[\sigma]$ and $\ell - |\sigma|$. More precisely, we have $F_\sigma = G_{\Sigma[\sigma],\ell-|\sigma|}$, where $G_{s,i} := \{\omega_x : \exists \alpha \in [n]^i : \tau(s,\alpha) = x\}$. This diagram shows the values of $G_{s,i}$ for Algorithm 3.1.1 solving the MIF$(4,2)$ problem. The sets $G_{s,i}$ are represented by the dark squares in the array of four cells. The transition function between states is indicated by the colored arrows; for example, green colored arrows (those emitting from squared numbered with a 3) correspond to transitions where the next stream element is a 3, i.e, from state s to state $s' = \tau(s,3)$.

algorithm would emit if the state is reached at the end of the stream. (If there is no stream of length ℓ leading to state s, we let ω_s be arbitrary.)

For each partial stream $\sigma \in [n]^\star$, let

$$F_\sigma = \{i : \exists x \in \Sigma, \exists \alpha \in [n]^{\ell-|\sigma|} : \tau(\Sigma[\sigma],\alpha) = x \wedge \omega_x = i\} \tag{3.9}$$

be the set of possible outputs of the algorithm when σ is extended to a stream of length ℓ. Because there are only $|\Sigma|$ states, and only $[n]$ possible output values, $|F_s^i| \le m$, where $m = \min(|\Sigma|, n)$.

Let t, q be integers chosen later, so that

$$tq \le \ell - m/2^q\,. \tag{3.10}$$

We claim that there exists a partial stream $\sigma \in [n]^\star$ satisfying $\forall \alpha \in [n]^t : |F_{\sigma.\alpha}| \ge \frac{1}{2}|F_\sigma|$.

Such a state can be found by an iterative process. Let τ_0 be the empty stream ε; for $i = 0, 1, 2, \ldots$, if there exists $\alpha \in [n]^t$ for which $|F_{\tau_i.\alpha}| \le \frac{1}{2}|F_{\tau_i}|$, let $\tau_{i+1} = \tau_i.\alpha$. Otherwise, stop, and let $\sigma = \tau_i$. This process must terminate before $i = q$, because otherwise we would have $|F_{\tau_q}| \le m/2^q \le \ell - qt$. Then letting $\gamma \in [n]^{\ell-qt}$ be a sequence of elements containing every element of F_{τ_q}, we observe that the algorithm cannot possibly output a correct answer for the stream $\tau_q.\gamma$. By the definition of F_{τ_q}, we must have $\omega_{\tau_q.\gamma} \in F_{\tau_q}$; but to be a correct missing item finding solution, we need $\omega_{\tau_q.\gamma} \notin \gamma$, hence $\omega_{\tau_q.\gamma} \notin F_{\tau_q}$, a contradiction. Thus, $\sigma = \tau_i$ for some $i \le q - 1$. Thus $|\sigma| \le (q-1)t \le \ell - t$, which ensures that the terms $\sigma.\alpha$ are streams of length $\le \ell$ and therefore well defined. Finally, the stopping condition of the process implies $\forall \alpha \in [n]^t : |F_{\sigma.\alpha}| \ge \frac{1}{2}|F_\sigma|$.

We will now construct a deterministic protocol for AVOID$(|F_\sigma|, t, \lceil \frac{1}{2}|F_\sigma| \rceil)$ using $\le \log|\Sigma|$ bits of communication. Alice, on being given a set $A \in \binom{F_\sigma}{t}$, arbitrarily orders it to form a sequence α in

$(F_\sigma)^t$; and then sends the state $s' = \tau(\Sigma[\sigma], \alpha)$ to Bob. This can be done using $\log|\Sigma|$ bits of space. Bob uses the encoded state to find $F_{\sigma.\alpha}$, by evaluating $\omega_{\tau(s',\beta)}$ for all sequences $\beta \in [n]^{|\sigma|-t}$, and reports the first $\lceil \frac{1}{2}|F_\sigma| \rceil$ elements of this set as B. This protocol works because as claimed above, we are guaranteed $|F_{\sigma.\alpha}| \geq |F_\sigma|$; and furthermore, $F_{\sigma.\alpha}$ must be disjoint from A; if it is not, then there exists some continuation of σ concatenated with α which leads the algorithm to a state z with $\omega_z \in A$, contradicting the correctness of the MIF protocol. Finally, applying the communication lower bound from Lemma 3.3.2, we find

$$\log|\Sigma| \geq \frac{1}{\ln 2} t \left\lceil \frac{1}{2}|F_\sigma| \right\rceil / |F_\sigma| \geq t/(2\ln 2)\,. \tag{3.11}$$

We now determine values of t and q satisfying Eq. 3.10. We can set

$$q = \lceil 1 + \log(m/\ell) \rceil \qquad \text{and} \qquad t = \left\lfloor \frac{1}{q}\left(\ell - \frac{m}{2^q}\right) \right\rfloor .$$

We must have $m \geq \ell + 1$, as otherwise $|F_\varepsilon| \leq m \leq \ell$, in which case we could easily make the algorithm give an incorrect output by running it on a stream $\gamma \in [n]^\ell$ which contains all elements of F_ε. Thus $\log(m/\ell) \geq 0$, and hence $q \geq 1$, making t well defined. Since $m = \min(|\Sigma|, n)$, we are also guaranteed $\log|\Sigma| \geq \log(\ell+1)$. Combining this with Eq. 3.11 gives:

$$\begin{aligned}
\log|\Sigma| &\geq \max\left(\log(\ell+1), \frac{1}{2\ln 2}\left\lfloor \frac{1}{q}\left(\ell - \frac{m}{2^q}\right)\right\rfloor\right) \\
&\geq \max\left(1, \frac{1}{2\ln 2}\left\lfloor \frac{\ell}{2q} \right\rfloor\right) && \text{since } 2^q \geq 2m/\ell \text{ and } \ell \geq 1 \\
&\geq \frac{1}{1 + 2\ln 2} \cdot \frac{\ell}{2q} && \text{since } \min(1, (z-1)/y) \geq \frac{z}{1+y} \\
&\geq \frac{\ell}{10 + 5\log(m/\ell)}\,. && \text{since } 1 + 2\ln 2 \leq 5/2
\end{aligned} \tag{3.12}$$

As $m = \min(|\Sigma|, n)$, we have $m \leq |\Sigma|$, so

$$\log|\Sigma| \geq \frac{\ell/5}{2 + \log|\Sigma| - \log\ell} \qquad \Longrightarrow \qquad (\log|\Sigma|)^2 + (2 - \log\ell)\log|\Sigma| - \ell/5 \geq 0\,.$$

Solving the quadratic inequality gives:

$$\log|\Sigma| \geq \sqrt{\frac{\ell}{5} + \left(1 - \frac{\log(\ell)}{2}\right)^2} - \left(1 - \frac{\log(\ell)}{2}\right) \geq \begin{cases} \sqrt{\ell/5} & \text{if } \ell \geq 4 \\ 0 & \text{otherwise} \end{cases}$$

As $\log|\Sigma| \geq \log(\ell+1) \geq \sqrt{\ell/5}$ also holds for $\ell \leq 4$, it follows that $\log|\Sigma| \geq \sqrt{\ell/5}$ for all values of

ℓ. Combining this result, Eq. 3.12, and the inequality $m \leq n$, we conclude:

$$\log |\Sigma| \geq \max\left(\sqrt{\frac{\ell}{5}}, \frac{\ell}{10 + 5\log(n/\ell)}\right) = \Omega\left(\sqrt{\ell} + \frac{\ell}{1 + \log(n/\ell)}\right)\square$$

Note: instead of associating "forward" looking sets of outputs F_s with each state $s \in \Sigma$, we could instead use "backward" looking states B_s defined (roughly) as $[n] \setminus \{i : \exists \sigma \text{ leading to } s \text{ with } i \in \sigma\}$.

3.5.2 Upper bound: filtering by coordinates

We will present a deterministic algorithm that comes close to the lower bound for MIF(n, ℓ). The algorithm will remap all values from $[n]$ to $[s]^t$ for some integers s and t. It will proceed in t stages; in the ith stage, it records the set of ith coordinates of inputs that arrived during the ith stage, until there is one "safe" value remaining for the ith coordinate, that no input from this stage has used. The output of the algorithm will be a remapped value in $[s]^t$ which matches known "safe" values on coordinate 1 through i, and has an arbitrary value on coordinates $i+1$ through t; this output will differ from every input seen so far on at least one coordinate.

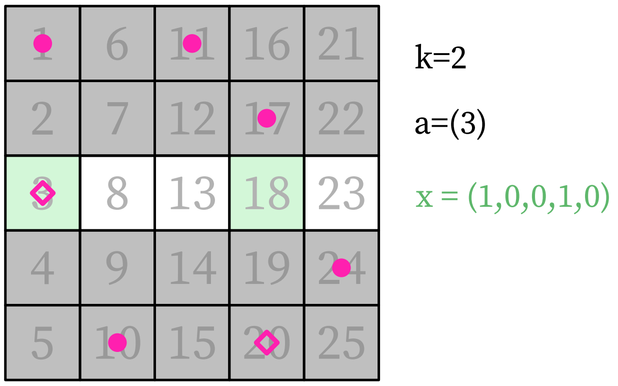


Figure 3.6: This diagram shows the behavior of Algorithm 3.5.1, with $s = 5$ and $t = 2$, on an example input. The pink circles and diamonds mark the elements currently covered by the stream. Cells shaded dark gray are those which are no longer possible outputs due to the current values of k and a. Cells shaded light green are no longer possible outputs due to the value of the vector x. Cells shaded white are possible output values. The algorithm proceeds in t phases; in this example, for the first phase, it maintained a bit vector tracking which of the s rows of the set $[n]$ contained an element from the stream; after the first five elements (1, 10, 11, 17, 24 in some order) arrived, only one row was left available, and the algorithm proceeded to the second phase – maintaining a bit vector x that records which columns within the chosen row may be unavailable.

Theorem 3.5.2. *Algorithm 3.5.1 is a deterministic algorithm that solves* MIF(n, ℓ) *using* $O(\sqrt{\ell}\log\ell + \frac{\ell\log\ell}{\log n})$ *bits of space.*

Proof of Algorithm 3.5.1. First, we establish that the variables (k, a) of the algorithm stay in their specified bounds. The condition in Line 5 ensures that k will not be increased beyond t. At the

Algorithm 3.5.1 A deterministic algorithm for $\textsc{mif}(n, \ell)$

```
    Let $s, t$ be integers satisfying $s^t \le n$, and $t(s-1) \ge \ell$.

    Initialization:
 1: $x \leftarrow (0, \ldots, 0)$ is a vector in $\{0,1\}^s$.
 2: $(k, a) \leftarrow (1, 0)$ is an element of $\bigcup_{j \in [t]} \{j\} \times \{0, \ldots, s^j - 1\}$

    Update($e \in [n]$):
 3: Let $i \leftarrow \left(\lfloor (e-1)/s^{k-1} \rfloor \bmod s\right) + 1$
 4: $x_i \leftarrow 1$
 5: if $k < t$ and there is exactly one $y \in [s] : x_y = 0$ then
 6:     $x \leftarrow (0, \ldots, 0)$
 7:     $k \leftarrow k + 1$
 8:     $a \leftarrow a + (y-1)s^{k-1}$

    Query:
 9: Let $i$ be the least value in $[s]$ for which $x_i = 0$
10: output: $a + (i-1)s^{k-1} + 1$
```

time Line 8 is executed, $a < s^{k-1}$; since $y \in [s]$, it follows $a + (y-1)s^{k-1} < (1 + (s-1))s^{k-1} \le s^k$, so the pair (k, a) stays in $\bigcup_{j \in [t]} \{j\} \times \{0, \ldots, s^j - 1\}$.

Next, we establish that the algorithm is correct; that the output from Line 10 does not overlap with current stream $e_1, \ldots, e_k$. For each element e_j in the stream, let k_j be the value of k at the time the element was added (i.e., at the start of the Update function). For all $h \in [t]$, define $C_h := \{j \in [t] : k_j = h\}$ to indicate the elements for which $k_j = h$. Because Line 5 only triggers when x has one zero entry, and x is reset to the all zero vector immediately afterwards, and each new element sets at most one entry of x to 1 (Line 4), we have $|C_h| \ge s - 1$ for all h less than or equal to the current value of k.

Let $c = a + (i-1)s^{k-1}$ be the current output of the algorithm (Line 10), minus 1. Note that $c \le s^t - 1 \le n - 1$, so the output is in $[n]$. The value of c can be written in base s as $(c_1, \ldots, c_t)$, so that $c = \sum_{j=1}^{t} c_j s^{j-1}$. For h less than the current value of k, c_h is equal to the value of y at the time the condition of Line 5 evaluated to true; in other words, at that time, $x_{c_h} = 0$. Now, for each $j \in C_h$, consider the value $e_j - 1$ written in base s as $(b_1, \ldots, b_t)$. When e_j was added, Line 3 set i equal to b_h, and so Line 4 ensured $x_{b_h} = 1$. Since $x_{c_h} = 0$ held afterwards, when the condition of Line 5 evaluated to true, it follows $b_h \ne c_h$. This implies $e_j - 1 \ne c$ holds for all $j \in C_h$. A similar argument will establish that for $j \in C_k$, we have $e_j - 1 \ne c$; since $C_1 \cup \ldots \cup C_k$ contains the entire stream so far, it follows the current output of the algorithm does not equal any of the $\{e_j\}_{j=1}^{k}$, and is thus correct.

Finally, we determine values of s and t which for which the algorithm uses little space. The vector x can be stored using s bits; since there are $\sum_{i=0}^{t-1} s^i \le s^t$ possible values of (k, a), this algorithm can be implemented using $\leq s + t \log s + 1$ bits of space.

Now let

$$q = \min\left(\sqrt{\ell \log(\ell+1)}, \log n\right) \qquad \text{and} \qquad t = \left\lfloor \frac{q}{\log(\ell+1)} \right\rfloor \qquad \text{and} \qquad s = \left\lceil \frac{\ell}{t} \right\rceil + 1\,,$$

Because $\ell \geq \log(\ell+1)$, and $\log n \geq \log(\ell+1)$, it follows $t \geq 1$. This implies $s \leq \ell + 1$. Then $t(s-1) = t\lceil \ell/t \rceil \geq \ell$, and

$$s^t \leq (\ell+1)^{\lfloor q/\log(\ell+1) \rfloor} \leq (\ell+1)^{q/\log(\ell+1)} \leq 2^q \leq n\,,$$

so the values of s and t satisfy the required conditions $s^t \leq n$ and $t(s-1) \geq \ell$. With these parameters, the space usage of the algorithm is:

$$\begin{aligned}
s + t\log s + 1 &\leq \left\lceil \frac{\ell}{t} \right\rceil + 2 + \left\lfloor \frac{q}{\log(\ell+1)} \right\rfloor \log(\left\lceil \frac{\ell}{t} \right\rceil + 1) \\
&\leq \frac{\ell}{\lfloor q/\log(\ell+1) \rfloor} + 3 + \frac{q}{\log(\ell+1)} \log(\ell+1) \\
&\leq \frac{2\ell\log(\ell+1)}{q} + q + 3 \\
&= \max\left(2\sqrt{\ell\log(\ell+1)}, \frac{2\ell\log(\ell+1)}{\log n}\right) + \min\left(\sqrt{\ell\log(\ell+1)}, \log n\right) + 3 \\
&= O\left(\sqrt{\ell\log\ell} + \frac{\ell\log\ell}{\log n}\right)\,.
\end{aligned}$$

□

3.5.3 Upper bound: using AVOID protocols to improve efficiency

Algorithm 3.5.1 does not perfectly match the structure of the lower bound in Theorem 3.5.1; for $\ell = n^{\Theta(1)}$, it uses $\Theta(\frac{\log \ell}{\log n}) = \Theta(1)$ stages, while the lower bound uses $\Theta(\log \frac{2n}{\ell})$. We will describe a deterministic algorithm that can use less space than Algorithm 3.5.1; however, this new algorithm has not been optimized for time. Instead of repeatedly finding a single safe coordinate, it repeatedly finds a safe subset of some previous, larger safe subset, using a deterministic protocol for AVOID.

Theorem 3.5.3. *There is a deterministic algorithm (Algorithm 3.5.2) for* MIF(n,ℓ) *using space:*

$$O\left(\frac{\ell \log\log\frac{4n}{\ell}}{\log\frac{4n}{\ell}} + \sqrt{\ell\log\ell}\right)\,.$$

See Figure 3.7 for an example of how Algorithm 3.5.2 runs on an example input.

Proof of Theorem 3.5.3. Consider the algorithm described in Algorithm 3.5.2. We will choose specific parameters h and v for it later. This algorithm proceeds in $h+1$ stages, numbered 0 through h. In the tth stage, the set $[v(2h)^{h+1}]$ of possible algorithm outputs is split into disjoint "blocks"

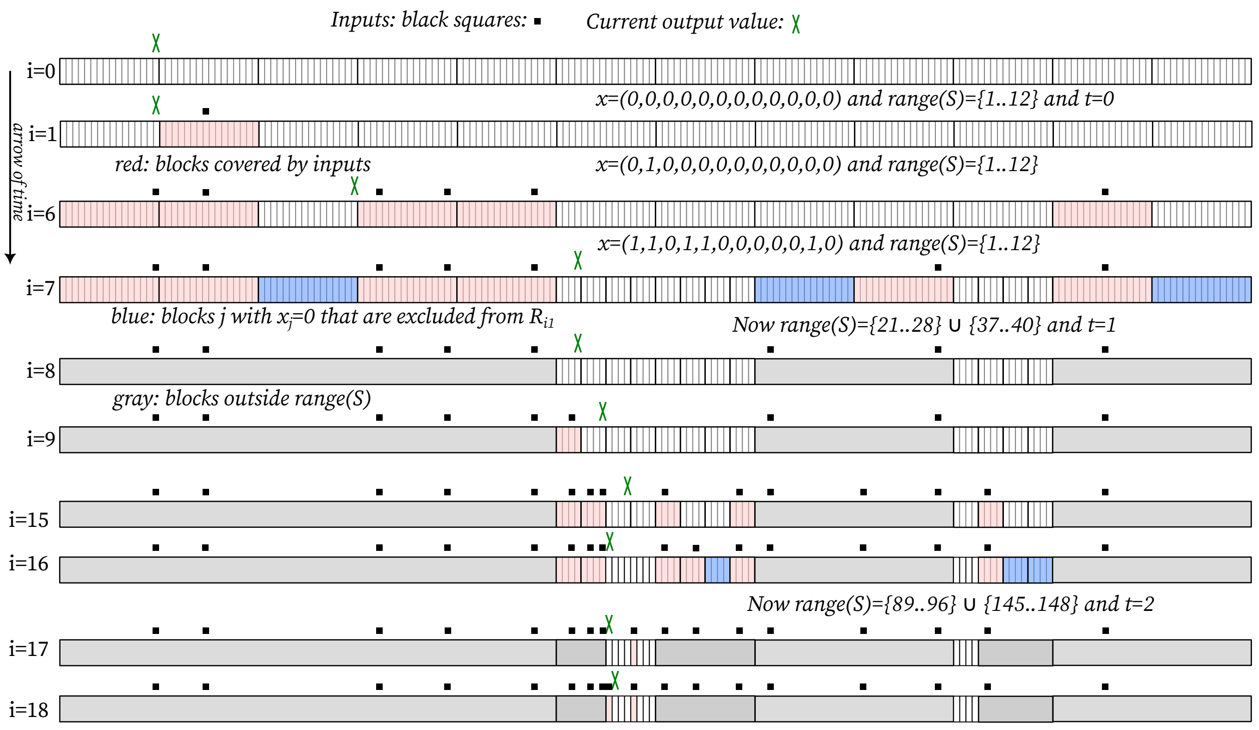

Inputs: black squares: ■
Current output value: X
i=0
i=1
i=6
i=7
i=8
i=9
i=15
i=16
i=17
i=18
arrow of time
x=(0,0,0,0,0,0,0,0,0,0,0,0) and range(S)={1..12} and t=0
x=(0,1,0,0,0,0,0,0,0,0,0,0) and range(S)={1..12}
red: blocks covered by inputs
x=(1,1,0,1,1,0,0,0,0,0,1,0) and range(S)={1..12}
blue: blocks j with x_j=0 that are excluded from R_{i1}
Now range(S)={21..28} ∪ {37..40} and t=1
gray: blocks outside range(S)
Now range(S)={89..96} ∪ {145..148} and t=2

Figure 3.7: This diagram shows the evolution of the state of Algorithm 3.5.2 on an example input for MIF$(n = 192, \ell = 18)$, with the algorithm configured to have parameters $v = 3$ and $h = 2$. In this case, whenever 6 out of 12 distinct tracked blocks have been covered, the algorithm picks a subset of 3 uncovered blocks which it will split into 4 subblocks each; it will then proceed to track incoming elements on the new set of 12 smaller blocks.

of size $(2h)^{h-t}$. (The jth block contains the values $\{(2h)^{h-t}(j-1)+1, \ldots, (2h)^{h-t}j\}$.) The values $a_1, \ldots, a_t$ are used to determine a set of $s = 2hv$ blocks which did not contain any inputs that were added during earlier stages. The vector $x \in \{0,1\}^s$ tracks which of the blocks of elements contain an input that was added during the tth stage.

The algorithm only proceeds to the next stage when the condition in Line 18 holds; in particular, $\sum_{i\in[s]} x_i = s/2$ is needed. After increasing the stage counter t, the algorithm resets x to the all-zero vector. As at most one coordinate of x is set to 1 for each element processed by the algorithm, each stage spans $\geq s/2 = hv$ inputs. Since $h(h+1)v \geq \ell$, after all ℓ elements have been processed, the maximum possible value of t will be h; and if $t = h$, the maximum value of $\sum_{i\in[s]} x_i$ will be $s/2$. Consequently, x will never be the all-1s vector, and whenever the algorithm is queried there will always exist some coordinate i for which $x_i = 0$.

Algorithm 3.5.2 always produces a valid output because it preserves the invariant that, for the set of blocks indicated by the range of the function S returned by GETMAP(), for each $i \in [s]$ with $x_i = 0$, the block indicated by $S(i)$ does not contain any elements of the input stream. In particular, when the if-condition at Line 18 holds and t, a_t, x are updated, the safe blocks in the range of the new value $\widetilde{S}$ of GETMAP() will be subsets of the safe blocks in the old value S of

Algorithm 3.5.2 A deterministic algorithm for $\textsc{mif}(n, \ell)$

```
    Requires: $\ell \le n/64$
 1: Let $h, v$ be integers satisfying $h(h+1)v \ge \ell$ and $v(2h)^{h+1} \le n$, and let $s = 2hv$

    Initialization:
 2: Define $R_1, \dots, R_z$ be an optimal set family for $\textsc{avoid}(s, s/2, s/2h)$ as per Lemma 3.3.4. Let
    $R_{i,j}$ be $j$th smallest element of $R_i$
 3: $a_1, \dots, a_h$ are values in $[z]$, all uninitialized
 4: $t \in \{0, \dots, h\}$ indicates the current stage
 5: $x \in \{0,1\}^s$, initially all zero, tracks which cells in $S$ are safe (0) or unsafe (1)

 6: ▷ Return a function $[s] \to [s(2h)^t]$ indicating which regions the algorithm is tracking
    GetMap():
 7: Let $S_0$ be the identity function from $[s]$ to $[s]$
 8: for $i$ in $1, \dots, t$ do
 9:     Let $S_i$ be the function from $[s] \to [s(2h)^i]$ mapping $y$ to
            $$2h(S_{i-1}(R_{a_i,\lceil y/(2h) \rceil}) - 1) + ((y-1) \bmod 2h) + 1 \qquad (3.13)$$
10: return $S_t$

    Update($e \in [n]$):
11: if $e > v(2h)^{h+1}$ then
12:     return
13: Let $c = \lceil e/(2h)^{h-t} \rceil$                    ▷ Project onto $[s(2h)^t]$
14: Let $S = \textsc{GetMap}()$
15: if $c \notin \text{image}(S)$ then
16:     return
17: $x_{S^{-1}(c)} \leftarrow 1$
18: if $\sum_{i \in [s]} x_i = s/2$ and $t < h$ then
19:     $t \leftarrow t + 1$
20:     $a_t \leftarrow j$, where $j$ is an index for which $R_j \cap \{i : x_i = 1\} = \emptyset$
21:     $x \leftarrow \vec{0}$

    Query:
22: $S \leftarrow \textsc{GetMap}()$
23: Let $i$ be the first index where $x_i = 0$
24: return $S(i)(2h)^{h-t}$
```

$\textsc{GetMap}()$.

Specifically, if i is contained in the block $\widetilde{S}(p)$ (of size $(2h)^{h-t-1} = \frac{1}{2h}(2h)^{h-t}$), then i is contained in the block $S(q)$ (of size $(2h)^{h-t}$), where $q = R_{a_i,\lceil p/(2h) \rceil}$, because (using Eq. 3.13)

$$\left\lceil \frac{i}{\frac{1}{2h}(2h)^{h-t}} \right\rceil = \widetilde{S}(p) = 2h(S(R_{a_i,\lceil p/(2h) \rceil}) - 1) + ((p-1) \bmod 2h) + 1$$

implies that

$$\left\lceil \frac{i}{(2h)^{h-t}} \right\rceil = \left\lceil \left\lceil \frac{i}{\frac{1}{2h}(2h)^{h-t}} \right\rceil \Big/ (2h) \right\rceil$$
$$= \left\lceil \big(2h(S(R_{a_i,\lceil p/(2h)\rceil}) - 1) + ((p-1) \bmod 2h) + 1\big) \Big/ (2h) \right\rceil = S(R_{a_i,\lceil p/(2h)\rceil})\,.$$

Note that the size $(2h)^{h-t}$ block corresponding to $S(q)$ does not contain any stream elements by the invariant, the choice of a_i, and the definition of the $\{R_j\}_{j\in[z]}$ from Lemma 3.3.4.

Parameters and space usage. One way to choose parameters h and v satisfying the conditions of Line 1, is to pick:

$$h = \left\lceil \min\left(\sqrt{\ell/\log(\ell+1)}, \frac{\log\frac{n}{32\ell}}{2+\log\log\frac{n}{32\ell}}\right) \right\rceil \qquad \text{and} \qquad v = \left\lceil \frac{\ell}{h(h+1)} \right\rceil$$

It follows immediately that $h(h+1)v \geq \ell$. We also have

$$h(h+1) \leq \left(h + \frac{1}{2}\right)^2 \leq \left(\left\lceil \sqrt{\ell/\log(\ell+1)} \right\rceil + \frac{1}{2}\right)^2 \leq (\sqrt{\ell} + \frac{3}{2})^2 \leq 8\ell\,.$$

Since $\lceil y \rceil \leq \max(2, \frac{1}{\min y})y$, this implies:

$$h^2 v = h^2 \left\lceil \frac{\ell}{h(h+1)} \right\rceil \leq h^2 \frac{8\ell}{h(h+1)} \leq 8\ell\,. \tag{3.14}$$

Then:

$$v(2h)^{h+1} = (2h)^2 v (2h)^{h-1} \leq 32\ell \cdot \left(2\left\lceil \frac{\log\frac{n}{32\ell}}{2+\log\log\frac{n}{32\ell}} \right\rceil\right)^{\left\lceil \frac{\log\frac{n}{32\ell}}{2+\log\log\frac{n}{32\ell}} - 1 \right\rceil}$$
$$\leq 32\ell \left(4 \log \frac{n}{32\ell}\right)^{\frac{\log\frac{n}{32\ell}}{2+\log\log\frac{n}{32\ell}}}$$
$$\leq 32\ell \cdot \frac{n}{32\ell} \leq n\,.$$

We will later use the facts that $\sqrt{\ell/\log(\ell+1)}$ is $\geq \frac{1}{2}$ for all $\ell \geq 1$, and that for $n \geq 64\ell$,

$$\frac{\log\frac{n}{32\ell}}{2+\log\log\frac{n}{32\ell}} \geq \frac{\log 2}{2 + \log\log 2} \geq \frac{1}{2}\,.$$

Using the bound on $\lceil \log z \rceil$ from Lemma 3.3.4, and the fact that there is a trivial $2hv$-bit protocol for AVOID$(2hv, hv, v)$, we find

$$\lceil \log z \rceil \leq \min\left(2hv, \frac{hv \cdot v}{2hv - hv - v + 1} + \log(2hv)\right) \leq 2v + \log(2hv) \leq 3v + \log h\,. \tag{3.15}$$

The algorithm stores three types of variables: x, using $s = 2hv$ bits of space; $a_1, \ldots, a_h$, using $\leq \lceil \log z \rceil$ bits each, and t, using $\lceil \log(h+1) \rceil$ bits. Applying Eq. 3.15 to the total space usage gives:

$$2hv + h\lceil \log z \rceil + \lceil \log(h+1) \rceil \leq 2hv + h(3v + \log h) + h \leq 6hv + h \log h \, .$$

By Eq. 3.14, $hv \leq 8\frac{\ell}{h}$. Thus the space is:

$$\begin{aligned}
&\leq \frac{48\ell}{h} + h \log h \\
&\leq \frac{48\ell}{\left\lceil \min(\sqrt{\ell / \log(\ell+1)}, \frac{\log \frac{n}{32\ell}}{2 + \log\log \frac{n}{32\ell}}) \right\rceil} + \left\lceil \sqrt{\ell / \log(\ell+1)} \right\rceil (3 + \log \left\lceil \sqrt{\ell / \log(\ell+1)} \right\rceil) \\
&\leq \frac{48\ell}{\sqrt{\ell / \log(\ell+1)}} + \frac{48\ell}{\frac{\log \frac{n}{32\ell}}{2 + \log\log \frac{n}{32\ell}})} + 2\sqrt{\ell / \log(\ell+1)}(3 + \log(2\sqrt{\ell / \log(\ell+1)})) \\
&\leq \frac{48\ell(2 + \log\log \frac{n}{32\ell})}{\log \frac{n}{32\ell}} + 64\sqrt{\ell \log(\ell+1)} = O\left(\frac{\ell \log\log \frac{4n}{\ell}}{\log \frac{2n}{\ell}} + \sqrt{\ell \log \ell} \right) . \qquad \square
\end{aligned}$$

Theorem 3.5.3 almost matches the asymptotic space complexity lower bound for deterministic algorithms for MIF(n, ℓ), from Theorem 3.5.1. The slight (factor $\Theta(\min(\log \ell, \log\log \frac{4n}{\ell}))$) difference in the lower and upper bounds corresponds to a possibility that the algorithms do not use and the lower bounds do not refute.

Specifically, the lower bound can be seen as showing that the algorithm embeds a sequence of protocols for AVOID, and proves a space lower bound using them. However, the proof does not account for the space cost of remembering the results of AVOID; it assumes the protocols may involve *all* states of the algorithm, and there is no state that needs to be preserved. On the other hand, Algorithm 3.5.2 repeatedly uses deterministic protocols for AVOID, and explicitly remembers the results of each AVOID subproblem. Consequently, it has a factor $O(h)$ less space to remember each AVOID solution, and instead of shrinking the space of safe outputs by a factor of 2, as the lower bound proof assumes, shrinking the space of safe outputs by a factor $O(h)$. If we could somehow efficiently encode a sequence of h solutions to AVOID, each individually needing $\Omega(v)$ bits, in significantly less than $O(hv)$ space, we might be able to shrink the space of outputs less strongly at each stage, and the deterministic space complexity might match our current lower bound. On the other hand, if there is no way to share information for a chain of AVOID outputs, then the deterministic space complexity might match the current upper bound.

3.6 Pseudo-deterministic lower bound

The proof presented in this section will generalize the space lower bound for deterministic MIF(n, ℓ) algorithms, which was given in Section 3.5. To briefly recap: in the deterministic lower bound, we defined, for each prefix $\sigma \in [n]^\star$ of a length ℓ input stream, the set F_σ containing all the possible values for outputs at time ℓ for streams starting with σ. This definition ensures that if $\tau \in [n]^\star$ is a continuation of σ, then $F_\tau \subseteq F_\sigma$. The deterministic proof used that fact that the algorithm implements $\textsc{mif}(n, \ell)$ to implement a protocol for $\textsc{avoid}(|F_\sigma|, t, \frac{1}{2}|F_\sigma|)$, for some integer t. By Lemma 3.3.2, this protocol must use $\Omega(t)$ bits of state; equivalently, if $z < Ct$ for some constant C, there *cannot* be a such a protocol, which will imply that there exists a stream ρ which appends t integers to σ, for which $|F_\rho| < \frac{1}{2}|F_\sigma|$. Repeating this argument, if $z < Ct$, it is possible to construct a sequence of stream prefixes $\sigma_1, \ldots, \sigma_s$, with each σ_i adding t more elements on to σ_{i-1}, and each F_{σ_i} no more than half the size of $F_{\sigma_{i-1}}$. If we had $s \geq \log n$, then $|F_{\sigma_s}| < |F_{\sigma_1}|/n = 1$, which is impossible, because the algorithm must make an output at time ℓ.[12] Thus, we can obtain an upper bound on s, which implies a lower bound on t, which implies a lower bound on z for any correct algorithms. See the proof of Theorem 3.5.1 for more details.

Relaxing "all outputs" to "common outputs". The deterministic lower bound does not work as-is for pseudo-deterministic algorithms. A key property needed to construct an AVOID protocol in the deterministic case is that, given just the state v which the algorithm will have after processing partial stream $\sigma \in [n]^\star$, and the value $\ell - |\sigma|$, one can compute F_σ. For each $\tau \in [n]^{\ell - |\sigma|}$, one can evaluate the algorithm on the concatenation $\sigma.\tau$ by running its state machine from state v, with inputs from τ, and then recording the output. To obtain F_σ, one takes the union of all outputs resulting from all $\tau \in [n]^{\ell - |\sigma|}$.

For pseudo-deterministic algorithms, we must use a different definition for F_σ. We cannot just define it to be "the set of all *canonical* outputs at time ℓ for continuations of σ", because this cannot be computed reliably from a single state: given a random state v associated to σ, on average a δ fraction of the outputs might be incorrect and have arbitrary values. Even a single bad output could corrupt the union calculation! To work around this issue, we define an algorithm that recursively computes the "most common outputs" at time ℓ for a certain distribution over continuations of σ. This algorithm, called FINDCOMMONOUTPUTS, or FCO for short, is parameterized by a sequence of randomly chosen thresholds.[13] We defer its definition to the formal proof. It can be run against both the "canonical" algorithm, which always produces the canonical outputs, and the actual algorithm, starting from some specific state. We will show that:

- With high probability over the random thresholds, FCO will produce the same outputs on the canonical and actual algorithms. In other words, FCO is robust to noise (algorithm errors).
- When applied to the canonical algorithm, the outputs of FCO will have a similar structure to the values F_σ from the deterministic proof: that is, the sizes of $\textsc{fco}(\sigma, ...)$ will shrink

[12]In fact, $|F_{\sigma_s}| \geq \ell - |\sigma_i| + 1$ must hold, so with suitable parameters we can set $s \approx \log \frac{2n}{\ell}$.
[13]The use of random thresholds is a standard trick for robustly computing quantities in the presence of noise.

geometrically as the length of σ is increased. This can be proven by implementing AVOID using FCO on (a state of) the actual algorithm as a subroutine.

Since FCO is defined recursively, these steps will be part of a proof by induction.

Error amplification and the case $n \gg \ell$. There is a catch to our design; the recursive method to estimate sets of common outputs requires that the algorithm have error probability as small as $1/n^{\Omega(\log n)}$. Because the algorithm is pseudo-deterministic, even if the original error probability is $1/3$, by running many independent copies of the algorithm and choosing the most common output we can obtain a new algorithm with the necessary error level, prove a space lower bound for that, and thereby derive a space lower bound for the original algorithm. This procedure is made more complicated by another feature of our lower bound. A pseudo-deterministic algorithm using z bits of state can be shown to have only $O(2^z)$ possible outputs; so if $n \gg \ell$, we can often obtain a stronger lower bound by assuming that n is actually $O(2^z)$, and then solving the system of inequalities to get a lower bound on z.

3.6.1 Definitions and parameters

We will consider only random-oracle algorithms, as lower bounds for this randomness type imply lower bounds for random tape and random seed algorithms. Let $\mathcal{A}$ be a random oracle pseudo-deterministic algorithm for MIF(n, ℓ) using z bits of state, which has worst case failure probability $\delta \leq \frac{1}{3}$. Let $\Pi : [n]^\ell \to [n]$ be the function giving the canonical output of $\mathcal{A}$ after processing a stream of length ℓ, and let $S = \Pi([n]^t)$ be the set of all canonical outputs. Clearly $|S| \leq n$, and the later Lemma 3.6.7 will also prove $|S| \leq 2^{z+1}$.

The lower bound proof has the limitation that its parameters are only meaningful if z is below some threshold; specifically, for an integer p chosen later, we require that $z \leq \frac{\ell}{18p}$. The threshold $\frac{\ell}{18p}$, it turns out, will be larger than the lower bound on z that we will prove, so in the end it has no visible effect. In the following text, we will assume $z \leq \frac{\ell}{18p}$, unless stated otherwise.

Let

$$p = \left\lceil \sqrt{\frac{5\ell \log(64|S|)}{3z \log 1/(2\delta)}} \right\rceil \qquad \text{and} \qquad d = \left\lfloor \frac{\ell}{18zp} \right\rfloor \leq \left\lfloor \frac{\ell}{2\lceil 4 \ln 2(zp+2) \rceil} \right\rfloor \tag{3.16}$$

$$t_d = \ldots = t_2 = \lceil 4 \ln 2(zp+2) \rceil \qquad \text{and} \qquad t_1 = \ell - \sum_{k=2}^{d} t_k \tag{3.17}$$

$$w_1 = t_1 + 1 \qquad \text{and} \qquad \forall i \geq 2, \quad w_i = \left\lceil \frac{t_i}{2 \ln 2(zp+2)} w_{i-1} \right\rceil . \tag{3.18}$$

The main proof in this section only applies to algorithms with very low error (potentially as small as $\frac{1}{n^{\Omega(\log n)}}$), so we will first apply Lemma 2.3.4, using p independent instances of $\mathcal{A}$ to construct a new algorithm $\mathcal{B}$ which uses zp bits of space and has much smaller error, $\leq \varepsilon$, where $\varepsilon \leq (2\delta)^{p/30}$. Using the above definitions, we have:

$$\log \frac{1}{\varepsilon} \geq \frac{p}{30} \log \frac{1}{2\delta} \geq \frac{\ell}{18zp} \log(64|S|) \geq d \log(64|S|) = \log \frac{1}{1/(64|S|)^d} ,$$

and hence $\varepsilon \leq \frac{1}{(64|S|)^d}$. For use in the future, we define, for $k \in [d]$, $\varepsilon_k := w_k(64|S|)^{k-1}\varepsilon$. One can show using calculations given later that $\varepsilon_d < \frac{1}{64}$.

Note that since $\mathcal{B}$ was derived from $\mathcal{A}$, its canonical outputs are still described by Π. Since we can view $\mathcal{B}$ as a distribution over deterministic streaming algorithms, where which algorithm is used depends on the value of $\mathcal{B}$'s oracle random string, we can define a corresponding distribution $\mathcal{D}$ over the set of functions of type $[n]^\ell \to [n]$, where the probability of a particular function A is the probability that $\mathcal{B}$ uses a deterministic algorithm which produces outputs according to A. Because $\mathcal{B}$ is pseudo-deterministic, we are guaranteed the following property:

$$\forall x \in [n]^\ell : \Pr_{A \sim \mathcal{D}}[A(x) \neq \Pi(x)] \leq \varepsilon . \tag{3.19}$$

We describe a randomized algorithm FindCommonOutputs (short: fco) which, when run on Π and a random matrix $C \in [1,2)^{d \times \mathbb{N}}$, produces a large set T which is a subset of S; and when run on a random function $A \sim \mathcal{D}$ and on C, produces the same set with positive probability. See Procedure 3.6.1. A lower bound on $|T|$ (which depends on z) then implies a lower bound on $|S|$; with some algebra we can use this to derive a lower bound for z.

Procedure 3.6.1 and later proofs use the following notation. If for some integers $m' < m$, A is a function from $[n]^m \to [n]$, and $x \in [n]^{m'}$, then we use the notation A_x to indicate the function mapping each $y \in [n]^{m-m'}$ to $A(x.y)$. A_x can be seen as the result of using partial function application on A.

3.6.2 Proof by induction

We will now show that we can find a set contained in S of size $\geq w_d$. The following lemma is central to the proof:

Lemma 3.6.1. *Let $k \in \{1, \ldots, d\}$, and $x \in [n]^{t_d + \cdots + t_{k+1}}$.*

$$\Pr_{A \sim \mathcal{D}, C \in_R [1,2)^{d \times \mathbb{N}}}[\text{fco}(A_x, C, k) = \text{fco}(\Pi_x, C, k)] \geq 1 - \varepsilon_k .$$

Furthermore, fco(Π_x, C, k) will be disjoint from x and a subset of S, and fco$(\cdot, \cdot, k)$ will always output some set of size w_k.

We now prove Lemma 3.6.1, by induction on k. First, we prove the case $k = 1$:

Procedure 3.6.1 The procedure to compute a set for Lemma 3.6.1

Let $t_1, \ldots, t_d$, $w_1, \ldots, w_d$ be integer parameters, and S the set of valid outputs

```
FindCommonOutputs($B$, $C$, $k$)                                   ▷ A.k.a. fco($B, C, k$)
1:  ▷ $B$ is a function from $[n]^{t_k+\cdots+t_1}$ to $[n]$
2:  ▷ $C$ is a vector in $[1,2)^{d\times\mathbb{N}}$
3:  ▷ The output will be a subset of $S$ of size $w_k$
4:  if $k = 1$ then
5:      $e_0 \leftarrow B((1,1,1,\ldots,1))$
6:      for $i$ in $1,\ldots,t_1$ do
7:          $e_i \leftarrow B((e_0,\ldots,e_{i-1},1,\ldots,1))$
8:      $E \leftarrow \{e_0, e_1, \ldots, e_{t_1}\}$
9:      if $|E| = w_1$ then return $E$
10:     else return arbitrary subset of $S$ of size $w_1$
11: else
12:     for each $y \in \text{SORT}\binom{S}{t_k}$ do
13:         ▷ Recall $B_y$ is notation for the partial application of $B$ with prefix $y$
14:         $T_{B,y} \leftarrow$ FindCommonOutputs($B_y, C, k-1$)          ▷ note $|T_{B,y}| = w_{k-1}$
15:     $Q_0 \leftarrow T_{B,(1,2,\ldots,t_k)}$                            ▷ choice of seed vector is arbitrary
16:     for $h$ in $1,2,\ldots,\left\lfloor \frac{4w_k}{w_{k-1}} \right\rfloor - 1$ do
17:         for each $j \in S$ do
18:             $f_j^{(h)} \leftarrow \left|\left\{y \in \text{SORT}\binom{Q_{h-1}}{t_k} : j \in T_{B,y}\right\}\right|$
19:         $P_h \leftarrow \left\{j \in S : f_j^{(h)} \geq \frac{C_{k,h} w_{k-1}}{16|S|} \left|\binom{Q_{h-1}}{t_k}\right|\right\}$
20:         $Q_h \leftarrow Q_{h-1} \cup P_h$
21:         if $|Q_h| \geq w_k$ then
22:             return the $w_k$ smallest elements in $Q_h$
23:     return arbitrary subset of $S$ of size $w_k$
```

Lemma 3.6.2. *Lemma 3.6.1 holds for $k = 1$.*

Proof of Lemma 3.6.2. Let $e_0, \dots, e_{t_1}$ be the values of the variables on Lines 5 to 7 of Procedure 3.6.1 when $\textsc{fco}(\Pi, C, 1)$ is called; note that these do not depend on C. Define for $i \in \{0, \dots, t_1\}$ vectors $s_i = (e_0, \dots, e_{i-1}, 1, \dots, 1)$, so that $s_0 = (1, 1, \dots, 1)$, and $s_{t_1} = (e_0, \dots, e_{t_1 - 1})$. Then if for all $i \in \{0, \dots, t_1\}$, $A_x(s_i) = \Pi_x(s_i)$, the value of $\textsc{fco}(A_x, C, 1)$ will exactly match $\textsc{fco}(\Pi_x, C, 1)$. By a union bound,

$$\Pr_{A \sim \mathcal{D}, C}[\textsc{fco}(A_x, C, k) \neq \textsc{fco}(\Pi_x, C, k)] \leq \sum_{i=0}^{t_1} \Pr_{A \sim \mathcal{D}}[A_x(s_i) \neq \Pi_x(s_i)] \underset{\text{Eq. (3.19)}}{\leq} (t_1 + 1)\varepsilon = \varepsilon_1 \,.$$

Because Π is the canonical output function for a protocol for $\textsc{mif}$, for any $z \in [n]^\ell$, we have $\Pi(z) \notin z$. Consequently, each $e_i = \Pi(x.e_0.\dots.e_{i-1}.1.\dots.1)$ is neither contained in x nor by $\{e_0, \dots, e_{i-1}\}$; thus $\{e_0, \dots, e_{t_1}\}$ has size $t_1 + 1 = w_1$ and is disjoint from x. □

Proving the induction step is more complicated. First, we observe that:

Lemma 3.6.3. *Let $x \in [n]^{t_d + \dots + t_{k+1}}$. When computing $\textsc{fco}(\Pi_x, C, k)$, in the hth loop iteration, if $|Q_{h-1}| < w_k$, then $|P_h \setminus Q_h| \geq \frac{1}{4}\lceil w_{k-1} \rceil$. Consequently, the algorithm will return using Line 22, not Line 23.*

Proof of Lemma 3.6.3. Assume for sake of contradiction that $|Q_{h-1}| < w_k$ and $|P_h \setminus Q_h| \leq \lfloor \frac{1}{4} w_{k-1} \rfloor$. Then we can use the algorithm $\mathcal{A}$ to implement a protocol for the one-way communication problem $\textsc{avoid}(|Q_{h-1}|, t_k, \lceil \frac{1}{2} w_{k-1} \rceil)$, with $\leq \frac{1}{2}$ probability of error.

We assume without loss of generality that $Q_{h-1} = [|Q_{h-1}|]$; if not, relabel coordinates so that this holds. In the protocol, after Alice is given a subset $W \subseteq Q_{h-1}$ with $|W| = t_k$, Alice constructs a sequence $v = x.\textsc{sort}(W)$ in $[n]^{t_d + \dots + t_k}$. Then Alice uses public randomness to instantiate an instance E of $\mathcal{A}$; inputs the sequence v to E; and sends the new state of E to Bob, using a zp-bit message. As Bob shares public randomness with Alice, Bob can use this state to evaluate the output of the algorithm on any continuation of the stream. In particular, Bob can evaluate the algorithm for any possible suffix, to produce a function $\tilde{A}_{x.\textsc{sort}(W)} : [n]^{t_{k-1} + \dots + t_1} \to [n]$; Bob then samples a random $C \in [1, 2)^{d \times \mathbb{N}}$, and computes $V = \textsc{fco}(\tilde{A}_{x.\textsc{sort}(W)}, C, k-1)$, which is a subset of S. If $|V \cap Q_{h-1}| \geq \lceil \frac{1}{2} w_{k-1} \rceil)$, Bob outputs the smallest $\lceil \frac{1}{2} w_{k-1} \rceil$ entries of $V \cap Q_{h-1}$. Otherwise, Bob outputs an arbitrary set of size $\lceil \frac{1}{2} w_{k-1} \rceil$.

First, we observe that for any value of $\textsc{sort}(W)$, the distribution of $\tilde{A}_{x.\textsc{sort}(W)}$ is exactly the same as the distribution of $A_{x.\textsc{sort}(W)}$, when A is drawn from $\mathcal{D}$; this follows because for a fixed setting of the random string of the algorithm, it behaves deterministically.

Applying Lemma 3.6.1 at $k - 1$, we observe that for any $W \in \binom{Q_{h-1}}{t_k}$,

$$\Pr[\textsc{fco}(\tilde{A}_{x.\textsc{sort}(W)}, C, k-1) = \textsc{fco}(\Pi_{x.\textsc{sort}(W)}, C, k-1)] \geq 1 - \varepsilon_{k-1} \geq \frac{3}{4} \,.$$

Furthermore, we are guaranteed that $\textsc{fco}(\Pi_{x.\textsc{sort}(W)}, C, k-1)$ has size w_{k-1} and is disjoint from W.

We now bound the probability, over a random $y \in \textsc{sort}\binom{Q_{h-1}}{t_k}$, that $|\textsc{fco}(\Pi_{x.y}, C, k-1) \cap Q_{h-1}| < \lceil \frac{1}{2} w_{k-1} \rceil$. Define T_y and, for each $j \in S$, $f_j^{(h)}$, as in Procedure 3.6.1. In particular, we have:

$$\begin{aligned}
\Pr_{y,C}\left[|T_y \cap Q_{h-1}| < \left\lceil \frac{1}{2} w_{k-1} \right\rceil\right] &= \Pr_{y,C}\left[|T_y \setminus Q_{h-1}| > \left\lfloor \frac{1}{2} w_{k-1} \right\rfloor\right] \\
&\leq \Pr_{y,C}\left[|T_y \setminus P_h \setminus Q_{h-1}| > \left\lfloor \frac{1}{2} w_{k-1} \right\rfloor - \left\lfloor \frac{1}{4} w_{k-1} \right\rfloor\right] \qquad (3.20) \\
&\leq \Pr_{y,C}\left[|T_y \setminus P_h| \geq \frac{1}{4} w_{k-1}\right].
\end{aligned}$$

(The inequality on Eq. (3.20) follows since we assumed $|P_h \setminus Q_{h-1}| \leq \lfloor \frac{1}{4} w_{k-1} \rfloor$.) As $\sum_{j \notin P_h} f_j^{(h)} \geq \frac{1}{4} w_k |\{y : |T_y \setminus P_h| \geq \frac{1}{4} w_{k-1}\}|$, it follows

$$\begin{aligned}
\Pr_{y,C}\left[|T_y \setminus P_h| \geq \frac{1}{4} w_{k-1}\right] &\leq \frac{4}{w_{k-1}} \frac{\sum_{j \notin P_h} f_j^{(h)}}{\binom{|Q_{h-1}|}{t_k}} \\
&= \frac{4}{w_{k-1}} (|S| - |P_h|) \frac{C_{k,h} w_{k-1}}{16|S|} \\
&\leq \frac{4 \cdot 2}{16} = \frac{1}{2}.
\end{aligned}$$

Thus the probability that $|T_y \cap Q_{h-1}| < \lceil \frac{1}{2} w_{k-1} \rceil$ holds is $\leq 1/2$. Since Bob only gives an incorrect output when this happens or when $\textsc{fco}(\tilde{A}_{x.\textsc{sort}(W)}, C, k-1) \neq \textsc{fco}(\Pi_{x.\textsc{sort}(W)}, C, k-1)$, it follows by a union bound that the total failure probability is $\leq \frac{1}{2} + \frac{1}{4} \leq \frac{3}{4}$.

Consequently, the protocol implementation has $\leq \frac{3}{4}$ error when inputs are drawn from the uniform distribution over $\binom{Q_{h-1}}{t_k}$; by Lemma 3.3.2, we obtain a lower bound on the required message length, giving

$$zp > \frac{t_k \lceil \frac{1}{2} w_{k-1} \rceil}{|Q_{h-1}| \ln 2} + \log(1 - 3/4) \geq \frac{t_k w_{k-1}}{|Q_{h-1}| \cdot 2 \ln 2} - 2.$$

Rearranging this slightly and using integrality of $|Q_{h-1}|$ gives:

$$|Q_{h-1}| \geq \left\lceil \frac{t_k w_{k-1}}{2 \ln 2 (zp + 2)} \right\rceil = w_k,$$

but as $|Q_{h-1}| < w_k$, this implies $w_k < w_k$, which is a contradiction; this proves that the assumption $|P_h \setminus Q_h| \leq \frac{1}{4} w_{k-1}$ must have been invalid.

Finally, we observe that since, in each iteration of the loop on Lines 16 to 22, $|Q_h| = |Q_{h-1} \cup P_h| = |Q_{h-1}| + |P_h \setminus Q_{h-1}| \geq |Q_{h-1}| + \lceil \frac{1}{4} w_{k-1} \rceil$, and we initially have $|Q_0| = w_{k-1}$, the size of Q_h

(assuming we haven't returned yet) must be $\geq w_{k-1}(1+h/4)$. As soon as $h \geq \frac{4w_k}{w_{k-1}} - 4$, we will have $|Q_h| \geq w_k$; as there are up to $\left\lfloor \frac{4w_k}{w_{k-1}} \right\rfloor - 1$ loop iterations, this will certainly happen. $\square$

Lemma 3.6.4. *For $k > 1$, $x \in [n]^{t_d+\ldots+t_{k+1}}$, $\textsc{fco}(\Pi_x, C, k)$ is disjoint from x and a subset of S; and for all A, C, k, $\textsc{fco}(A_x, C, k)$ outputs a set of size w_k.*

Proof of Lemma 3.6.4. That $\textsc{fco}(A_x, C, k)$ always outputs a set of size w_k follows from the structure of the algorithm: having finite loops, it always terminates, and either returns a set of size w_k via Step 22, or a set of size w_k via Step 23.

By Lemma 3.6.1 at $k-1$, the sets $T_{A_x,y}$ chosen on Line 14 are always subsets of S and disjoint from $x.y$, and hence disjoint from x. Per Lemma 3.6.3, $\textsc{FindCommonOutputs}$ will return a subset of Q_h using Line 22, where h is the last loop iteration number. Q_h only contains integers which were either in $T_{A_x,(1,2,\ldots,t_k)}$ (and hence also in S) or which were in $P_{h'}$ for some $h' \leq h$. Note that $P_{h'}$ only contains integers j for which $f_j^{(h')} > 0$; i.e., which were contained in one of the sets $(T_{A_x,y})_{y \in \textsc{sort}\binom{Q_{h'-1}}{t_k}}$, and are thereby also in S. $\square$

Lemma 3.6.5. *For $k > 1$, and all $x \in [n]^{t_d+\ldots+t_{k+1}}$,*

$$\Pr_{A\sim\mathcal{D},C}[\textsc{fco}(A_x, C, k) \neq \textsc{fco}(\Pi_x, C, k)] \leq \varepsilon_k\,.$$

Proof of Lemma 3.6.5. The proof of the lemma follows from the observation that, when computing $\textsc{fco}(A_x, C, k)$, even if a fraction of the recursive calls to $\textsc{fco}(A_{x.y}, C, k-1)$ produced incorrect outputs, the values for Q_0 and $(P_h)_{h\geq 1}$ will likely match those computed when $\textsc{fco}(\Pi_x, C, k)$ is called.

Henceforth, we indicate variables from the computation of $\textsc{fco}(\Pi_x, C, k)$ without a tilde, and variables from the computation of $\textsc{fco}(A_x, C, k)$ with a tilde. For example, $f_j^{(h)}$ is computed using $B = \Pi_x$, while $\tilde{f}_j^{(h)}$ is computed using $B = A_x$. We also define

$$\hat{f}_j^{(h)} = \left|\left\{ y \in \textsc{sort}\binom{Q_{h-1}}{t_k} : j \in T_{A,y} \right\}\right|$$
$$\hat{P}_h = \left\{ j \in S : \hat{f}_j^{(h)} \geq \frac{C_{k,h} w_{k-1}}{16|S|} \left|\binom{Q_{h-1}}{t_k}\right| \right\};$$

that is, $\hat{f}_j^{(h)}$ and $\hat{P}_h$ are the values that would be computed by $\textsc{fco}(A_x, C, k)$ if the set Q_{h-1} was used instead of the set $\widetilde{Q}_{h-1}$.

Say $\textsc{fco}(\Pi_x, C, k)$ returns from the loop at iteration $h^\star$. The output of $\textsc{fco}(A_x, C, k)$ will equal $\textsc{fco}(\Pi_x, C, k)$ if $Q_0 = \widetilde{Q}_0$ and $P_h = \hat{P}_h$ for all $h \in [h^\star]$. (If this occurs, then because $Q_0 = \widetilde{Q}_0$, $\hat{P}_1 = \widetilde{P}_1$, so $Q_1 = Q_0 \cup P_1 = \widetilde{Q}_0 \cup \widetilde{P}_1 = \widetilde{Q}_1$, and because $Q_1 = \widetilde{Q}_1$, $\hat{P}_2 = \widetilde{P}_2$, and so on.) By Lemma 3.6.1 at $k-1$, the probability that $Q_0 \neq \widetilde{Q}_0$ is $\leq \varepsilon_{k-1}$. Consider a specific $h \in [h^\star]$; the only way in which $\hat{P}_h \neq P_h$ is if there is some $j \in S$ for which $f_j^{(h)}$ and $\hat{f}_j^{(h)}$ are on opposite sides of the threshold $\frac{C_{k,h} w_{k-1}}{16|S|} |\binom{Q_{h-1}}{t_k}|$.

Let λ_h be the random variable indicating the fraction of $y \in \textsc{sort}\binom{Q_{h-1}}{t_k}$ for which $T_{A_x,y} \neq T_{\Pi_x,y}$. Note that the values $T_{A_x,y}$ are functions of the random variable A and of $C_{k',h}$ for $k' < k, h \in \mathbb{N}$; in particular $T_{A_x,y}$ is independent of $(C_{k,h})_{h\in\mathbb{N}}$. By Lemma 3.6.1 at $k-1$, $\Pr[T_{A_x,y} \neq T_{\Pi_x,y}] \leq \varepsilon_{k-1}$, which implies $\mathbb{E}\lambda_h \leq \varepsilon_{k-1}$.

Fix a particular setting of A and $(C_{k',h})_{k'<k,h\in\mathbb{N}}$. Since each set $T_{A_x,y}$ contributes 1 unit to each of w_{k-1} variables $\hat{f}_j^{(h)}$:

$$\sum_{j\in S} \left| f_j^{(h)} - \hat{f}_j^{(h)} \right| \leq w_{k-1} \left| \left\{ y \in \textsc{sort}\binom{Q_{h-1}}{t_k} : T_{A_x,y} \neq T_{\Pi_x,y} \right\} \right| = w_{k-1}\lambda_h \binom{Q_{h-1}}{t_k} .$$

Let F be the set of possible values in $[1,2)$ for $C_{k,h}$ for which $P_h \neq \hat{P}_h$; this is a union of intervals corresponding to each pair $\left(f_j^{(h)}, \hat{f}_j^{(h)}\right)$, for $j \in S$. A given value c is bad for j if

$$f_j^{(h)} < \frac{c w_{k-1}}{16|S|} \binom{|Q_{h-1}|}{t_k} \leq \hat{f}_j^{(h)} , \qquad \text{equivalently:} \qquad c \in \left(\frac{16|S| f_j^{(h)}}{w_{k-1}\binom{|Q_{h-1}|}{t_k}}, \frac{16|S|\hat{f}_j^{(h)}}{w_{k-1}\binom{|Q_{h-1}|}{t_k}} \right] ,$$

and similarly in the case where $\hat{f}_j^{(h)} < f_j^{(h)}$. The measure of F is

$$\leq \sum_{j\in S} \frac{16|S|}{w_{k-1}\binom{|Q_{h-1}|}{t_k}} |\hat{f}_j^{(h)} - f_j^{(h)}| \leq \frac{16|S|}{w_{k-1}} w_{k-1}\lambda_h = 16|S|\lambda_h .$$

This upper bounds the probability that $C_{k,h} \in F$ and $P_h \neq \hat{P}_h$. We then have:

$$\Pr[P_h \neq \hat{P}_h] = \mathbb{E}_{A,(C_{k',h})_{k'<k}} \Pr[C_{k,h} \in F] \leq \mathbb{E}_{A,(C_{k',h})_{k'<k}} (16|S|\lambda_h) = 16|S|\varepsilon_{k-1} .$$

By a union bound, the probability that $Q_0 \neq \widetilde{Q}_0$ or $P_h \neq \hat{P}_h$ for any $h \leq h^\star$ is

$$\leq \varepsilon_{k-1} + h^\star 16|S|\varepsilon_{k-1} \leq 16 \left\lfloor \frac{4w_k}{w_{k-1}} \right\rfloor |S|\varepsilon_{k-1} \leq \frac{64|S|w_k}{w_{k-1}} \varepsilon_{k-1} .$$

Thus $\Pr[\textsc{fco}(A_x, C, k) \neq \textsc{fco}(\Pi_x, C, k)] \leq \frac{64|S|w_k}{w_{k-1}} \varepsilon_{k-1} = \varepsilon_k$. □

Finally, we observe that Lemmas 3.6.2 to 3.6.5, together imply Lemma 3.6.1.

3.6.3 Calculating the lower bound

A consequence of Lemma 3.6.1 is that $\textsc{fco}(\Pi, C, d)$ will output a set of size w_d which is a subset of S.

Lemma 3.6.6. *Even if $18zp > \ell$, we have:*

$$z \geq \frac{\ell}{8460 \log \frac{2|S|}{\ell}} \min\left(1, \frac{\log(1/2\delta)}{\log(64|S|) \log \frac{2|S|}{\ell}}\right).$$

Proof of Lemma 3.6.6. If, as we have assumed, $18zp \leq \ell$, then for each $i \geq 2$, by Eqs. (3.17) and (3.18),

$$w_i = \left\lceil \frac{t_i}{2\ln 2(zp+2)} w_{i-1} \right\rceil = \left\lceil \frac{\lceil 4\ln 2(zp+2)\rceil}{2\ln 2(zp+2)} w_{i-1} \right\rceil \geq 2w_{i-1}\,.$$

Also, by Eqs. (3.16) and (3.17),

$$w_1 \geq \ell + 1 - \sum_{k=2}^{d} t_k = \ell + 1 - d\lceil 4\ln 2(zp+2)\rceil \geq \ell + 1 - \frac{\ell}{2} \geq \lceil \ell/2 \rceil\,.$$

Since $w_d \leq |S|$, we obtain:

$$|S| \geq w_d \geq 2^d \lceil \ell/2 \rceil \quad\Longrightarrow\quad \log\frac{2|S|}{\ell} \geq d \geq \frac{\ell}{36zp} \quad\Longrightarrow\quad zp \geq \frac{\ell}{36 \log \frac{2|S|}{\ell}}\,. \tag{3.21}$$

On the other hand, if $18zp \geq \ell$, then we tautologically have $zp \geq \frac{\ell}{18}$. Taking the minimum of this and Eq. (3.21) gives an inequality that holds in *all* cases:

$$zp \geq \min\left(\frac{\ell}{18}, \frac{\ell}{36 \log \frac{2|S|}{\ell}}\right) \geq \frac{\ell}{36 \log \frac{2|S|}{\ell}}\,, \qquad \text{since } \log\frac{2|S|}{\ell} \geq 1.$$

Next, we rearrange and expand the definition of p, using $\lceil x \rceil \leq \max(1, 2x)$:

$$\log\frac{2|S|}{\ell} \geq \frac{\ell}{36zp} \geq \min\left(\frac{\ell}{36z}, \frac{\ell}{72z}\sqrt{\frac{3z\log 1/2\delta}{5\ell\log(64|S|)}}\right) = \min\left(\frac{\ell}{36z}, \sqrt{\frac{\ell \log 1/2\delta}{8460z\log(64|S|)}}\right).$$

We now have two cases: if the left side of the minimum is smaller, then

$$z \geq \frac{\ell}{36\log\frac{2|S|}{\ell}}\,, \qquad \text{while otherwise,} \qquad z \geq \frac{\ell \log 1/(2\delta)}{8460\log(64|S|)(\log\frac{2|S|}{\ell})^2}\,.$$

Computing a common lower bound for the two cases gives:

$$z \geq \frac{\ell}{8460 \log \frac{2|S|}{\ell}} \min\left(1, \frac{\log(1/2\delta)}{\log(64|S|) \log \frac{2|S|}{\ell}}\right). \qquad \square$$

Lemma 3.6.7. $|S| < 2^{z+1}$.

Proof of Lemma 3.6.7. For each $a \in S$, let $x_a \in \Pi^{-1}(a)$. One can use $\mathcal{A}$ to provide a randomized

$\leq \delta$-error, z-bit encoding of the elements in S. Using public randomness, encoder and decoder choose the oracle random string for $\mathcal{A}$. Each $a \in S$ is encoded by sending x_a to $\mathcal{A}$ and outputting the algorithm state σ. To decode, given a state σ, one evaluates the output of $\mathcal{A}$ at state σ. Using the minimax principle, one can prove that the randomized encoding requires $\geq \log((1-\delta)|S|)$ bits of space, which implies $2^z \geq (1-\delta)|S|$. Since $\delta \leq \frac{1}{3}$, it follows $s \leq \frac{3}{2}2^z < 2^{z+1}$. □

We now establish the main result:

Theorem 3.6.8. *Pseudo-deterministic δ-error random oracle algorithms for* MIF(n, ℓ) *require*

$$\Omega\left(\min\left(\frac{\ell}{\log\frac{2n}{\ell}} + \sqrt{\ell}, \frac{\ell \log\frac{1}{2\delta}}{(\log\frac{2n}{\ell})^2 \log n} + \left(\ell \log \frac{1}{2\delta}\right)^{1/4}\right)\right)$$

bits of space when $\delta \leq \frac{1}{3}$. In particular, when $\delta = 1/\operatorname{poly}(n)$ and $\ell = \Omega(\log n)$, this is:

$$\Omega\left(\frac{\ell}{(\log\frac{2n}{\ell})^2} + (\ell \log n)^{1/4}\right) .$$

Using Lemma 3.6.6 and the fact that $S \subseteq [n]$, we obtain $|S| \leq \min(n, 2^{z+1})$. The theorem follows by combining this bound with the inequality of Lemma 3.6.7, and four each of four cases corresponding to different branches of min and max, solving to find a lower bound on z.

Proof of Theorem 3.6.8. By Lemma 3.6.6, we have:

$$z \geq \min\left(\frac{\ell}{8460 \log\frac{2|S|}{\ell}}, \frac{\ell}{8460 \log\frac{2|S|}{\ell}} \cdot \frac{\log(1/2\delta)}{\log(64|S|)\log\frac{2|S|}{\ell}}\right) . \tag{3.22}$$

By Lemma 3.6.7, $|S| \leq \min(n, 2^{z+1}) \leq \min(n, 4^z)$. We will apply this inequality to each branch of the minimum in Eq. 3.22. First, say that the left part of the minimum is larger than the right. Then $z \geq \ell/(8460 \log\frac{2|S|}{\ell})$. Applying $|S| \leq n$ and $|S| \leq 4^z$, this implies:

$$z \geq \frac{\ell}{8460 \log\frac{2n}{\ell}} , \qquad \text{and}$$

$$z \geq \frac{\ell}{8460 \log\frac{2 \cdot 4^z}{\ell}} \geq \frac{\ell}{8460 \cdot 3z} \qquad \Longrightarrow \qquad z \geq \sqrt{\frac{\ell}{25380}} .$$

Thus:

$$z \geq \max\left(\frac{\ell}{8460 \log\frac{2n}{\ell}}, \sqrt{\frac{\ell}{25380}}\right) . \tag{3.23}$$

Next, say that the right side of the minimum in Eq. 3.22 is larger. Then applying $|S| \leq n$ and

$|S| \leq 4^z$ to that side, we get:

$$z \geq \frac{\ell \log(1/2\delta)}{8460(\log \frac{2n}{\ell})^2 \log(64n)}, \qquad \text{and}$$

$$z \geq \frac{\ell \log(1/2\delta)}{8460(\log \frac{2 \cdot 4^z}{\ell})^2 \log(64 \cdot 4^z)} \geq \frac{\ell \log(1/2\delta)}{609120 z^3} \qquad \Longrightarrow \qquad z \geq \left(\frac{\ell \log(1/2\delta)}{609120}\right)^{1/4}.$$

Thus:

$$z \geq \max\left(\frac{\ell \log(1/2\delta)}{8460(\log \frac{2n}{\ell})^2 \log(64n)}, \left(\frac{\ell \log(1/2\delta)}{609120}\right)^{1/4}\right). \tag{3.24}$$

The minimum of the lower bounds from Eqs. 3.23 and 3.24 holds in all cases, so:

$$z \geq \min\left(\max\left(\frac{\ell}{8460 \log \frac{2n}{\ell}}, \sqrt{\frac{\ell}{25380}}\right), \max\left(\frac{\ell \log(1/2\delta)}{8460(\log \frac{2n}{\ell})^2 \log(64n)}, \left(\frac{\ell \log(1/2\delta)}{609120}\right)^{1/4}\right)\right).$$

□

Remark 3.6.9. For $\delta \leq 2^{-\ell}$, Theorem 3.6.8 reproduces the deterministic algorithm space lower bound for $\textsc{mif}(n, \ell)$ from Theorem 3.5.1 within a constant factor.

3.7 Random seed space complexity, adversarial setting

3.7.1 Lower bound: a general reduction to the pseudo-deterministic case

Our lower bound proof for random seed algorithms in the adversarial setting, relies on one key observation. At a given point in the stream, either the adversary is able to provide an input on which the algorithm has a high entropy output distribution (and the algorithm will learn a significant amount of information about the initial random bits of the algorithm), or for every possible input, the algorithm produces a canonical output with $\geq 2/3$ probability. As a result, we can design an adversary which in a number of steps, either learns new information about the algorithm's random seed, or identifies a range of inputs on which the algorithm will act like a pseudo-deterministic algorithm. As the random seed itself has limited entropy, the adversary cannot learn new information about the seed too often, and will (most likely) identify a region where the algorithm behaves pseudo-deterministically. Applying a space lower bound for pseudo-deterministic algorithms for MIF will complete the argument.

Lemma 3.7.1. *Define $S^{PD}_{1/3}(n, t)$ to be the space complexity of pseudo-deterministic random oracle algorithms for $\textsc{mif}(n, t)$ with $\leq 1/3$ error.*

Say $\mathcal{A}$ is a z-bit random oracle algorithm solving $\textsc{mif}(n, \ell)$ in the adversarial setting, with error $\leq \frac{1}{6}$, and let R be the random oracle string that it uses. Then $z \geq S^{PD}_{1/3}(n, \lfloor \ell / \lceil 4H(R) \rceil \rfloor)$, where

$H(R)$ is the entropy of R.

In particular, since any z-bit random seed algorithm can be implemented as a z-bit random oracle algorithm with $H(R) \le z$, all $\le 1/6$-error z-bit random seed algorithms for $\text{MIF}(n, \ell)$ in the adversarial setting satisfy $z \ge S^{PD}_{1/3}(n, \lfloor \ell/(4z) \rfloor)$.

Proof of Lemma 3.7.1. Write $B \sim \mathcal{A}$ to indicate that B is an instance of $\mathcal{A}$, initialized using random oracle string R. Let $h = \lceil 4H(R) \rceil$, and $t = \lfloor \ell/h \rfloor$. For any partial stream σ of elements, and instance B of $\mathcal{A}$, we let $B(\sigma)$ be the sequence of $|\sigma|$ outputs made by B after it processes each element in σ.

Consider an adversary $\mathcal{E}$ which does the following. Defining σ to be the stream it has already passed to the algorithm, and ν the sequence of outputs that $\mathcal{A}$ produced in response to σ, the adversary checks if there exists any $x \in [n]^t$ for which

$$\forall y \in [n]^t : \Pr_{B\sim\mathcal{A}}[B(\sigma.x) = \nu.y \mid B(\sigma) = \nu] \le \frac{2}{3}\,. \tag{3.25}$$

If so, it sends x to $\mathcal{A}$, appends x to σ and the returned t elements to ν, and repeats the process. If no such x exists, then the adversary identifies the $w \in [n]^t$ which maximizes:

$$\Pr_{B\sim\mathcal{A}}[B(\sigma.w) \text{ is incorrect} \mid B(\sigma) = \nu]\,. \tag{3.26}$$

and sends it to the algorithm. (The adversary gives up if either the algorithm manages to give a valid output after w, or after it has sent h sets of t elements to the algorithm.)

We claim that if $z < S^{PD}_{1/3}(\text{MIF}(n,t))$, then $\mathcal{E}$ makes the algorithm fail with probability $\ge 1/6$. There are two ways that $\mathcal{E}$ will not make the algorithm fail: if it tries more than $h-1$ times to find a point where there is no $x \in [n]^t$ satisfying Eq. 3.25, or if the w it sends fails to produce an error.

Assume that the adversary finds a value of x satisfying Eq. 3.25, for each of the h tries it makes. Let $x_1, \ldots, x_h \in [n]^t$ be these values, and let $y_1, \ldots, y_h \in [n]^t$ be the outputs of the algorithm. By applying Eq. 3.25 repeatedly, we have:

$$\begin{aligned}
&\Pr_{B\sim\mathcal{A}}[B(x_1.\ldots.x_h) = y_1.\ldots.y_h] \\
&\quad= \Pr_{B\sim\mathcal{A}}[B(x_1.\ldots.x_h) = y_1.\ldots.y_h \mid B(x_1.\ldots.x_{h-1}) = y_1.\ldots.y_{h-1}] \\
&\qquad\cdot \Pr_{B\sim\mathcal{A}}[B(x_1.\ldots.x_{h-1}) = y_1.\ldots.y_{h-1} \mid B(x_1.\ldots.x_{h-2}) = y_1.\ldots.y_{h-2}] \\
&\qquad\cdot\ldots\cdot \Pr_{B\sim\mathcal{A}}[B(x_1) = y_1] \\
&\quad\le (2/3)^h\,.
\end{aligned}$$

Let C be the event that the adversary finds a sequence satisfying Eq. 3.25, h times. Because the adversary is deterministic, we can define $\tau(R)$ to map values of the algorithm oracle random string to the input-output transcripts in $[n]^\star \times [n]^\star$, when the algorithm is run against the adversary. Let $\mathcal{T}$ be the set of transcripts that could occur under event C. Then we have:[14]

$$\begin{aligned} H(R) \geq H(\tau(R)) &\geq \sum_{T \in \mathcal{T}} \Pr[\tau(R) = T] \log \frac{1}{\Pr[\tau(R) = T]} \\ &\geq \sum_{T \in \mathcal{T}} \Pr[\tau(R) = T] \log (3/2)^h = \Pr[C] \cdot h \log \frac{3}{2} \,. \end{aligned}$$

Consequently,

$$\Pr[C] \leq \frac{H(R)}{h \log \frac{3}{2}} = \frac{H(R)}{\lceil 4H(R) \rceil \log \frac{3}{2}} \leq \frac{1}{4 \log(3/2)} < \frac{1}{2} \,.$$

Thus, the chance that the adversary $\mathcal{E}$ fails to find a point where no x satisfying Eq. 3.25 exists is $< \frac{1}{2}$.

To bound the second way in which $\mathcal{E}$ can fail, we let (σ, υ) be a partial transcript of the algorithm for which no $x \in [n]^t$ satisfies Eq. 3.25. Assume the probability that w produces an error is $< 1/3$. Then we have:

$$\forall w \in [n]^t, \exists y_w \in [n]^t : \qquad \Pr_{B \sim \mathcal{A}}[B(\sigma.w) = \upsilon.y_w \mid B(\sigma) = \upsilon] \geq \frac{2}{3} \tag{3.27}$$

$$\forall w \in [n]^t : \qquad \Pr_{B \sim \mathcal{A}}[B(\sigma.w) \text{ is correct} \mid B(\sigma) = \upsilon] \geq \frac{2}{3} \,. \tag{3.28}$$

These conditions together imply that $\upsilon.y_w$ is a correct $\textsc{mif}(n, \ell)$ output sequence for $\sigma.w$. As a result, we can use $\mathcal{A}$'s behavior after (σ, υ) to construct a pseudo-deterministic algorithm Ψ for $MIF(n, t)$. To initialize Ψ, we sample an initial state $B \sim \mathcal{A}$ conditioned on the event that $B(\sigma) = \upsilon$, and then send the elements of σ to B. After this, when Ψ receives an element e, we send e to B, and report the element B outputs as the output of Ψ. By Eqs. 3.27 and 3.28, the sequence of outputs produced by Ψ on any input x in $[n]^t$ will, with probability $\geq 2/3$, be the (correct) output y_w. Thus, Ψ solves $MIF(n, t)$ with $\leq 1/3$ error – which, under the assumption that $z < S^{PD}_{1/3}(\textsc{mif}(n, t))$, is impossible. Thus the w chosen by the adversary makes the algorithm err with probability $\geq 1/3$, conditional on it having found (σ, υ) with no $x \in [n]^t$ satisfying Eq. 3.25. The probability that the adversary succeeds/algorithm fails is then $\geq 1/3 \cdot 1/2 = 1/6$; this contradicts the assumption that $\mathcal{A}$ has error $\leq 1/6$ against any adversary, which implies that we must instead have $z \geq S^{PD}_{1/3}(\textsc{mif}(n, t))$. □

Remark 3.7.2. In the proof of Lemma 3.7.1, we only used the self-similarity property of $\textsc{mif}$: that when an algorithm solves $\textsc{mif}(n, \ell)$, it also solves $\textsc{mif}(n, t)$ for any subsequence of t consecutive inputs. In fact, Lemma 3.7.1 can be applied to any streaming problem with such a property. More

[14]Note: if $\textsc{support}(R) \leq 2^z$, then we in fact get $\Pr[C] = \sum_{T \in \mathcal{T}} \Pr[\tau(R) = T] \leq 2^z (2/3)^h$, which gives a stronger upper bound on $\Pr[C]$.

generally, if a random-seed algorithm $\mathcal{A}$ solves problem P with $\leq 1/6$-error, then its space usage is lower-bounded by the minimum space needed, for any stream prefix σ, to implement an algorithm that pseudo-deterministically solves a problem Q_σ with $1/3$-error, where Q_σ is the task of solving P on input streams that start with σ and have t additional elements.

Combining Lemma 3.7.1 and Theorem 3.6.8 gives us an unconditional lower bound for adversarially robust random seed algorithms. The proof is straightforward, with some casework. (This lower bound is mainly useful for $\ell \leq n^{2/3}$; for larger ℓ, the $\Omega(\ell^2/n)$ lower bound of Theorem 3.3.5 is stronger.)

Corollary 3.7.3. *Adversarially robust random seed algorithms for* $\textsc{mif}(n, \ell)$ *with error* $\leq \frac{1}{6}$ *require* $\Omega\big(\sqrt{\ell/(\log n)^3} + \ell^{1/5}\big)$ *bits of space.*

Proof of Corollary 3.7.3. The lower bound from Theorem 3.6.8 for $\textsc{mif}(n, t)$ with error $\delta = 1/3$, showing constants, is:

$$\max\left(\frac{t\log(3/2)}{8460(\log\frac{2n}{t})^2\log(64n)}, \left(\frac{t\log(3/2)}{609120}\right)^{1/4}\right). \tag{3.29}$$

If $z \geq \ell/4$, then we tautologically have a lower bound of $\ell/4$. Otherwise, we have $4z \leq \ell$. Applying Lemma 3.7.1 gives, for the left branch of the max in Eq. 3.29, with $t = \lfloor\frac{\ell}{4z}\rfloor \geq \frac{1}{8z}$:

$$z \geq \left\lfloor\frac{\ell}{4z}\right\rfloor \frac{\log(3/2)}{8460(\log\frac{2n}{t})^2\log(64n)} \geq \frac{\ell}{8z}\frac{\log(3/2)}{8460(\log(2n))^2\log(64n)}$$

which implies

$$z \geq \sqrt{\frac{\ell}{135360(\log(2n))^2\log(64n)}} \geq \sqrt{\frac{\ell}{3790080(\log n)^3}}.$$

For the right branch of Eq. 3.29, we obtain:

$$z \geq \left(\left\lfloor\frac{\ell}{4z}\right\rfloor\frac{\log(3/2)}{609120}\right)^{1/4} \geq \left(\frac{\ell}{8z}\frac{1}{2\cdot 609120}\right)^{1/4},$$

which implies:

$$z^{5/4} \geq \left(\frac{\ell}{9745920}\right)^{1/4} \qquad\implies z \geq \left(\frac{\ell}{9745920}\right)^{1/5}.$$

Combining the two lower bounds gives:

$$z \geq \max\left(\sqrt{\frac{\ell}{3790080(\log n)^3}}, \left(\frac{\ell}{9745920}\right)^{1/5}\right).$$

This lower bound is always smaller than $\ell/4$, so it also holds in the case where $z \geq \ell/4$. □

Remark 3.7.4. While in this chapter we do not need it, Lemma 3.7.1 can be generalized to work for error thresholds $\leq 1/6$, if one distinguishes between two types of error for a pseudo-deterministic algorithm: the probability that the pseudo-deterministic algorithm makes a *non-canonical* output, and the probability that the pseudo-deterministic algorithm makes a *mistake* (output which is not correct). One can get a δ-error lower bound for z-random seed algorithms for $\textsc{mif}(n, \ell)$ as a function of a 1/3-non-canonical 2δ-mistake lower bound for pseudo-deterministic algorithms for $\textsc{mif}(n, \lfloor \ell/4z \rfloor)$.

This may well be a dead end for the sake of finding better $\textsc{mif}$-lower bounds; we suspect that Theorem 3.6.8 can be improved to match the deterministic lower bound within constants for even 1/3-non-canonical protocols, in which case investigating a separate mistake rate is of little use. The possibility is still important to note: a similar distinction, between mistakes and out-of-domain-outputs, proved useful for the random tape lower bound of Theorem 3.11.7.

Remark 3.7.5. Lemma 3.7.1 can not just be used to imply space lower bounds for random seed algorithms in the adversarial setting, but for *all* algorithms for $\textsc{mif}(n, \ell)$ using $\leq \ell/4$ "bits of randomness". If a random oracle algorithm $\mathcal{A}$ for $\textsc{mif}(n, \ell)$ uses an oracle random string R from some distribution with $H(R) \leq \lfloor \ell/4 \rfloor$, then applying Lemma 3.7.1 and Theorem 3.6.8 implies $\mathcal{A}$ needs at least $\Omega(\frac{\ell}{H(R)\,\text{polylog}(n)})$ bits of space. In particular, any $\leq 1/6$-error, $O(\text{polylog}(n))$ space algorithm requires $\Omega(\ell/\text{polylog}(n))$ bits of randomness. A similar tradeoff can be proven for random tape algorithms in the adversarial setting.

3.7.2 Upper bound: hidden list of subsets

We now present a random-seed algorithm for the adversarial setting whose total space *with random bits included* can be better than Algorithm 3.3.1. This new algorithm, instead of having a hidden list of individual elements in $[n]$, stores a hidden list L of disjoint blocks of elements, and instead of just outputting a single element from a block, will recursively run a deterministic algorithm for $\textsc{mif}$ inside the current "active" block. The algorithm is given by Algorithm 3.7.1, and a figure illustrating its state is given by Figure 3.8.

Theorem 3.7.6. *Algorithm 3.7.1 is a random seed algorithm that solves $\textsc{mif}(n, \ell)$ in the adversarial setting, with error $\leq \delta$, and can be implemented using $O\left((\frac{\ell^2}{n} + \sqrt{\frac{\ell}{\log n}} + \ell^{1/3} + \log \frac{1}{\delta}) \log \ell\right)$ bits of space.*

Proof of Theorem 3.7.6. Consider the algorithm formed by combining Algorithm 3.7.1 with the nested deterministic algorithm of Algorithm 3.5.1.

As with the proof of Theorem 3.3.5, we observe that Algorithm 3.7.1 will always either produce a valid output or abort (on Line 13). This follows from the runtime invariant that the algorithm

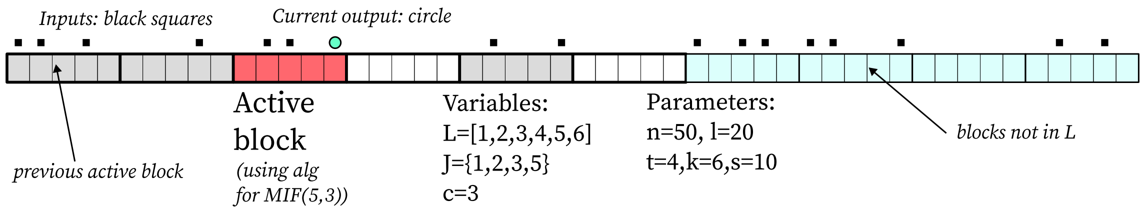


Figure 3.8: A diagram illustrating the state of an instance of Algorithm 3.7.1 on an example input. Positions on the horizontal axis correspond to integers in $[n]$; the set of values in the input stream $(\{1, 2, 4, 9, 12, 13, \ldots\})$ is marked with black squares; the current output value (15) with a circle. (The value of L used is for ease of presentation; L does not need to be contiguous or in sorted order.)

Algorithm 3.7.1 An adversarially robust, random-seed algorithm for $\textsc{mif}(n, \ell)$ with error $\leq \delta$

```
Assume $\ell \geq 16$, $\ell \leq n/16$ and $\delta \geq 2^{-\ell/15}$ – otherwise, use Algorithm 3.1.1
Let $t = \left\lceil \max\left(\frac{4\ell^2}{n}, \sqrt{\frac{\ell \log \ell}{\log n}}, (\ell \log \ell)^{1/3}\right)\right\rceil$     ▷ number of blocks used up by nested algorithm
Let $s = 2\ell$,     ▷ total number of blocks to split $[n]$ into
Let $k = \max(4t, \lceil 15 \log \frac{1}{\delta} \rceil)$     ▷ the length of the random list of blocks
Let $\mathcal{A}$ be a deterministic algorithm (like Algorithm 3.5.1) for $\textsc{mif}(\lfloor n/s \rfloor, \lfloor \ell/t \rfloor)$, which can report when it is "full" (has no more outputs)

**Initialization**:
 1: Let $L = (L_1, \ldots, L_k)$ be a sequence of $k$ distinct elements from $[s]$ chosen uniformly at random.
 2: $c \leftarrow 1$, an integer in the range from 1 to $k$
 3: $J \leftarrow \emptyset$, a subset of $[k]$
 4: $A \leftarrow$ a new instance of $\mathcal{A}$

**Update**($e \in [n]$):
 5: Let $h = \lceil e/\lfloor n/s \rfloor \rceil$     ▷ h is the number of the block containing e
 6: **if** there exists $j$ where $L_j = h$ **then**
 7:     $J \leftarrow J \cup \{j\}$
 8: **if** $h = L_c$ **then**
 9:     $A$.Update$(e - \lfloor n/s \rfloor (L_c - 1))$
10: **if** $A$.Query$() =$ full **then**     ▷ A will take $\geq 1 + \lfloor \ell/t \rfloor$ updates to fill
11:     $c \leftarrow$ least integer which is $> c$ and not in $J$
12:     **if** $c > t$ **then**
13:         **abort**
14:     $A \leftarrow$ a new instance of $\mathcal{A}$

**Query**:
15: Let $j \leftarrow A$.Query(), an integer in $[\lfloor n/s \rfloor]$
16: **output**: $\lfloor n/s \rfloor (L_c - 1) + j$
```

maintains: that for all $i \in [k] \setminus J$, *none* of the elements in L_i have been part of the input stream so far. As a result, when a new value for c is chosen, L_c will not contain any inputs. The deterministic algorithm, when run on the substream with domain L_c, will (until it returns Full) always report some element from L_c which was not in the stream since the deterministic algorithm instance started, and hence not in the stream at all. (In Lines 9 and 16, Algorithm 3.7.1 maps the subset L_c to $[\lfloor n/s \rfloor]$ and back.)

That the parameter $k \le \ell$ follows because, due to the constraint $\delta \ge 2^{-\ell/15}$, we have $\lceil 15 \log \frac{1}{\delta} \rceil \le \ell$; and because the three terms in the maximum within the definition for for t are all individually $\le \ell/4$ – the first because $\ell \le n/16$ implies $\frac{4\ell^2}{n} \le \ell/4$, and the other two because $\ell \ge 16$, which implies $\sqrt{\ell \frac{\log \ell}{\log n}} \le \sqrt{\ell} \le \ell/4$. Then $(\ell \log \ell)^{1/3} \le \ell/4$.

We observe that because $t \ge 4\frac{\ell^2}{n}$ and $n \ge 16\ell$, we have:

$$\left\lfloor \frac{\ell}{t} \right\rfloor + 1 \le \left\lfloor \frac{n}{4\ell} \right\rfloor + 1 \le \left\lfloor \frac{n}{2\ell} \right\rfloor = \left\lfloor \frac{n}{s} \right\rfloor,$$

so the nested algorithms for MIF are initialized with valid parameters.

The maximum number of times that a deterministic algorithm instance can report FULL, on Line 10, is $\lfloor \ell/(\lfloor \ell/t \rfloor + 1) \rfloor < t$, since each instance is updated at most once per call to UPDATE, and each instance is guaranteed to work for at least $\lfloor \ell/t \rfloor$ inputs, and will thus at earliest report FULL after $\lfloor \ell/t \rfloor + 1$. Consequently, the nested deterministic algorithm will be initialized a total of $\le t$ times, and over the course of the algorithm c will take $\le t$ different values.

The algorithm aborts precisely when $J = [k]$. We note that at any time, the adversary only has two useful options: to pick some input in the block associated to L_c ("walk"), or to pick an input from some block in $[s]$ which has not contained any input so far, and which is also not L_c ("guess"). Any other decision will have no effect on the state of the algorithm. We observe that the adversary can make at most ℓ "walk" inputs and at most ℓ "guess" inputs, in some order. Let F be the random variable giving the number of "guess" inputs. For each $i \in [\ell]$, let X_i be the indicator random variable for the event that the ith guess input (if $i \le F$) was in $L[[k] \setminus J \setminus \{c\}]$, and let U_i be the number of times the nested algorithm was initialized[15], before the ith guess. Then for each i,

$$\begin{aligned}\Pr[X_i = 1 | X_1, \ldots, X_{i-1}] &= \Pr[i\text{th guess in } L[[k] \setminus J \setminus \{c\}] \mid X_1, \ldots, X_{i-1}] \\ &= \frac{k - \sum_{j=1}^{i-1} X_j - U_i}{s - i + \sum_{j=1}^{i-1} X_j - U_i} \le \frac{k - \sum_{j=1}^{i-1} X_j}{s - i + \sum_{j=1}^{i-1} X_j},\end{aligned}$$

since the ith "guess" input will be some new value, and the $k - \sum_{j=1}^{i-1} X_j - U_i$ elements of L which were not guessed or otherwise revealed are uniformly random among the $s - i + \sum_{j=1}^{i-1} X_j - U_i$ elements of $[s]$ which were not guessed or otherwise revealed. Now, let T be a uniformly randomly chosen subset of $[s]$ of size ℓ, and for each $i \in [\ell]$, define indicator random variable Y_i to be 1 iff

[15] If W is the number of walk inputs, then $U_i \le 1 + \lfloor W/\lfloor n/s \rfloor \rfloor$.

$T \in [k]$. Note $\mathbb{E}|Y \cap [k]| = \frac{\ell k}{s} = \frac{1}{2}k$. We have for each i, and for $w_1, \dots, w_{i-1} \in \{0,1\}$:

$$\Pr[Y_i = 1 | Y_1 = w_1, \dots, Y_{i-1} = w_{i-1}] = \frac{k - \sum_{j=1}^{i-1} w_j}{s - i - \sum_{j=1}^{i-1} w_j} \geq \Pr[X_i = 1 | X_1 = w_1, \dots, X_{i-1} = w_{i-1}]\,.$$

Consequently:

$$\begin{aligned}
\Pr[\sum_{i=1}^{\ell} X_i \geq k - t] &\leq \Pr[\sum_{i=1}^{\ell} Y_i \geq k - t] \\
&\leq \Pr[\sum_{i=1}^{\ell} Y_i \geq \frac{3}{2} \cdot \frac{1}{2} k] && \text{because } t \geq k/4 \\
&\leq \exp\left(-\frac{1}{10} \cdot \frac{1}{2} k\right) && \text{by Chernoff bound, Lemma 2.3.3} \\
&\leq 2^{-\frac{1}{20 \ln 2} k} \leq 2^{-k/15} \leq \delta\,.
\end{aligned}$$

This proves that the probability of the adversary adding $\geq k - t$ entries to J from its guesses is $\leq \delta$. Since the adversary can make at most ℓ "walk" inputs, at most t different entries of J will be added by the "walk" strategy; so it follows the probability that $J = [k]$ will be $\leq \delta$.

Let $S(\hat{n}, \hat{\ell})$ be the space used by Algorithm 3.5.1 to solve $\textsc{mif}(\hat{n}, \hat{\ell})$. Then the space used by Algorithm 3.7.1 is:

$$\begin{aligned}
&\leq \underbrace{k \log s}_{\text{for } L} + \underbrace{k}_{\text{for } J} + \underbrace{\log k}_{\text{for } c} + \underbrace{S(\lfloor n/s \rfloor, \lfloor \ell/t \rfloor)}_{\text{for } A} + \underbrace{1}_{\text{for } \lceil\cdot\rceil \text{ of preceding terms}} \\
&\leq (t + \left\lceil 15 \log \frac{1}{\delta} \right\rceil) \log(4\ell) + 1 + \log \ell + S(\lfloor n/s \rfloor, \lfloor \ell/t \rfloor) \\
&\leq t \log(4\ell) + \log \frac{2}{\delta} \log(4\ell) + S\left(\left\lfloor \frac{n}{3\ell} \right\rfloor, \lfloor \ell/t \rfloor\right)\,.
\end{aligned}$$

The value of t balances the contributions of $S(\hat{n}, \hat{\ell}) = O(\frac{\hat{\ell} \log \hat{\ell}}{\log \hat{n}} + \sqrt{\hat{\ell} \log \hat{\ell}})$ with the term $t \log(4\ell)$. The space used is:

$$\begin{aligned}
O\Big(t \log \ell + \log \frac{1}{\delta} \log \ell + \frac{(\ell/t) \log(\ell/t)}{\log((2n/3\ell)/(\ell/t))} + \sqrt{\frac{\ell}{t} \log \frac{\ell}{t}}\Big) \\
\leq O\left(t \log \ell + \log \frac{1}{\delta} \log \ell + \frac{\ell \log \ell}{t \log(2n/\ell)} + \sqrt{\frac{\ell}{t} \log \ell}\right) \\
\leq O\left(t \log \ell + \log \frac{1}{\delta} \log \ell + \frac{\ell \log \ell}{t \log n} + \sqrt{\frac{\ell}{t} \log \ell}\right)
\end{aligned}$$

(because $\log \frac{2n}{\ell} = \Omega(\log n)$ when $\frac{\ell}{t \log n} \geq t$ holds)

$$\begin{aligned}
&\leq O\Bigg(\left(\frac{\ell^2}{n} + \sqrt{\frac{\ell \log \ell}{\log n}} + (\ell \log \ell)^{1/3} \right) \log \ell + \log \frac{1}{\delta} \log \ell \\
&\quad + \frac{\ell \log \ell}{\sqrt{\frac{\ell \log \ell}{\log n}} \log n} + \sqrt{\frac{\ell}{(\ell \log \ell)^{1/3}} \log \ell} \Bigg) \\
&\leq O\left(\frac{\ell^2}{n} \log \ell + \log \frac{1}{\delta} \log \ell + \sqrt{\frac{\ell \log \ell}{\log n}} + (\ell \log \ell)^{1/3} \right) \\
&\leq O\left(\left(\frac{\ell^2}{n} + \sqrt{\frac{\ell}{\log n}} + \ell^{1/3} + \log \frac{1}{\delta} \right) \log \ell \right) . \qquad \square
\end{aligned}$$

Remark 3.7.7. If there were a $1/\operatorname{poly}(\ell)$-error pseudo-deterministic algorithm for MIF(n, ℓ) using significantly less space than the best deterministic algorithms, we could reduce the space usage of the random seed algorithm in the $\ell = \operatorname{polylog}(n)$ regime further by instantiating copies of the pseudo-deterministic algorithm, configured for $O(\frac{\delta}{\ell})$ error probability. By Newman's theorem [New91], one can implement all the instances of the pseudo-deterministic algorithm, at additional error $O(\frac{\delta}{\ell})$ each, using a fixed random seed of size $O(\log \ell + \log \log N + \log(\ell/\delta)^2) = O(\log(\ell/\delta)^2)$. In the last expression, N is the number of possible valid algorithm outputs,[16] which for a $\leq \ell$-bit algorithm is $\leq 2^\ell$. The cost of Newman's theorem is negligible compared to the remaining space cost of the pseudo-deterministic algorithm instances.

3.8 White-box adversarial lower bound

In this section, we prove a lower bound for algorithms in the white-box adversarial setting. The lower bound uses a very similar argument to the deterministic lower bound for MIF (Theorem 3.5.1), and can be seen as a generalization of it.

For the deterministic lower bound, we defined, for each stream $\sigma \in [n]^\star$ of length $\leq \ell$, the set F_σ containing all the possible outputs at time ℓ when the algorithm is run on streams that start with σ. The set F_σ can be computed from just the state v that the algorithm reaches after processing σ, and the number $\ell - |\sigma|$ of future inputs. For the white-box case, we will define, for specific states v, a set H_v containing "safe" outputs. To be more accurate, because the algorithm has discarded information about its input sequence by projecting down to a limited number of states, every integer j in $[n] \setminus H_v$ will be "unsafe" to output: there will be an $\Omega(\ell/n)$ probability that j was an earlier input. Just as for deterministic algorithms, by a consequence of a reduction from AVOID, the sets F_σ shrunk exponentially as $|\sigma|$ increased, for the white-box adversarial setting, the

[16]For MIF, one can safely ignore all *inputs* which the algorithm can never output, so the number of valid output values is the same as the number of relevant input values.

sizes of the sets H_v will shrink exponentially as the adversary sends more inputs. In both cases, the exponent of shrinkage depends on the value of z, and because the sets cannot grow too small for a correct algorithm, one can derive a lower bound on z.

The adversary we design uses $O(\log \frac{n}{\ell})$ rounds of interaction; having this many appears to be unavoidable. The paper [ABJ+22], which introduced the white-box adversarial setting, found how to reduce white-box adversarially robust algorithms to deterministic 2-player communication protocols. Unfortunately, for Missing Item Finding, the natural 2-player communication game is $\textsc{avoid}(n, \ell/2, \ell/2)$, whose deterministic communication complexity is very close to its randomized communication complexity. (See Lemmas 3.3.2 and 3.3.3.) Using the reduction to 2-player communication would at best yield an $\Omega(\ell^2/n)$ lower bound, which is much weaker than the lower bound that we prove.

Theorem 3.8.1. *Random tape algorithms for $\textsc{mif}(n, \ell)$ in the white-box adversarial setting with error $\delta \le \min\left(\frac{1}{10}, \frac{\ell^2}{400n}\right)$ require space*

$$\Omega\left(\frac{\ell}{1 + \log \frac{n}{\ell}} + \sqrt{\ell}\right) .$$

The $O(\min(1, \ell^2/n))$ upper bound on the error can not be improved by much: there is a $\log n$ bit algorithm which just produces a random output on every step, and obtains $\Theta(\min(1, \ell^2/n))$ error. Thus, we appear to have a relatively sharp threshold between the error levels at which guessing the next value is possible, and the error levels at which it isn't. For the latter case, one cannot do much better than deterministic algorithms.

In our proof of Theorem 3.8.1 we will use the following lemma. It will be used to argue that, after the adversary sends a random subset to the algorithm, if the algorithm does not have enough memory to remember the entire subset, it must forget enough information that it will only have a small "safe" set of possible output values.

Lemma 3.8.2. *Let W and Σ be sets, d an integer, and P a function from $\binom{W}{d}$ to $\triangle[\Sigma]$, where $|\Sigma| \le 2^z$. Let F be a random function $\binom{W}{d} \to \Sigma$ in which for all $x \in \binom{W}{d}$ and $\sigma \in \Sigma$, $\Pr[F(x) = \sigma] = P(x)(\sigma)$, and let X be a uniformly random element of $\binom{W}{d}$, chosen independently of F. For each $\sigma \in \Sigma$, let H_σ be greedily constructed, starting from $\emptyset$, by repeatedly adding sets Q disjoint from the current value of H_σ for which $|Q| \le \left\lfloor \frac{|W|}{d} \right\rfloor$ and*

$$\Pr[Q \cap X \neq \emptyset \mid F(X) = \sigma] \le \frac{d|Q|}{4|W|} . \tag{3.30}$$

This ensures that for all sets Q of size $\le \left\lfloor \frac{|W|}{d} \right\rfloor$ which are disjoint from H_σ, Eq. 3.30 does not hold.

Adversary 3.8.1 A white-box adversary for z-bit algorithms for $\textsc{mif}(n, \ell)$

```
Let $U$ be the set of all possible outputs of the algorithm
Let $t = \left\lceil \log \frac{8|U|}{\ell} \right\rceil$

Adversary:
1: Let $H_0 = U$
2: extract current algorithm state $\sigma_0$
3: $k \leftarrow 0$
4: while $|H_k| \geq \ell/6$ do
5:     if $k \geq t$ then
6:         abort                                ▷ more iterations needed than planned
7:     $k \leftarrow k + 1$
8:     Let $d_k = \max(\lfloor \ell/(3t) \rfloor, \lfloor \ell/(6 \cdot 2^k) \rfloor)$        ▷ as $t \geq 4$, $d_1 = \lfloor \ell/12 \rfloor$
9:     Let $S_k$ be a random subset of $H_{k-1}$ of size $d_k$
10:    send $\textsc{sort}(S_k)$ to the algorithm
11:    Let $P_k : \binom{H_{k-1}}{d_k} \to \triangle\Sigma$ map each possible value of $S_k$ to the resulting distribution over
       states, if we start at state $\sigma_{k-1}$
12:    extract current algorithm state $\sigma_k$
13:    Let $H_k := H_{\sigma_k}$, where $H_{\sigma_k}$ is defined from $P_k$ according to Lemma 3.8.2.
14: ▷ Make the algorithm output $\lceil \ell/3 \rceil$ distinct elements
15: for $j = 1, \ldots, \lceil \ell/3 \rceil$ do
16:     extract current algorithm output $e_j$
17:     send $e_j$ to algorithm
```

Then for any $\alpha \in (0, 1)$:

$$\Pr\left[\left|H_{F(X)}\right| \geq \hat{w}\right] \leq \alpha \qquad \textit{where} \qquad \hat{w} := \left\lceil 50 \frac{z + 2 + \log \frac{1}{\alpha}}{d} n \right\rceil .$$

Proof of Lemma 3.8.2. This proof slightly generalizes that of Lemma 3.9.2. First, we observe that for any set Q of size $\leq \left\lfloor \frac{|W|}{d} \right\rfloor$,

$$\begin{aligned}
\Pr[X \cap Q \neq \emptyset] = 1 - \Pr[X \cap Q = \emptyset] &= 1 - \frac{\binom{|W|-d}{|Q|}}{\binom{|W|}{|Q|}} = 1 - \frac{(|W| - d) \cdots (|W| - d - |Q| + 1)}{|W| \cdots (|W| - |Q| + 1)} \\
&\geq 1 - \left(\frac{|W| - d}{|W|}\right)^{|Q|} \geq 1 - \exp\left(-\frac{|Q|d}{|W|}\right) \\
&\geq \frac{|Q|d}{2|W|} ,
\end{aligned}$$

where the last inequality follows because $\frac{|Q|d}{|W|} \leq \left\lfloor \frac{|W|}{d} \right\rfloor \frac{d}{|W|} \leq 1$:

For any $\sigma \in \Sigma$, as a consequence of the greedy construction of H_σ, we can decompose H_σ into

t disjoint sets $Q_1, \ldots, Q_t$ each satisfying Eq. 3.30. By linearity of expectation:

$$\mathbb{E}\left[\sum_{i\in[t]} \mathbb{1}_{X\cap Q_i\neq\emptyset} \mid F(X)=\sigma\right] = \sum_{i\in[t]} \Pr[X\cap Q_i \neq \emptyset \mid F(X)=\sigma] \leq \sum_{i\in[t]} \frac{d|Q_i|}{4|W|} \leq \frac{d|H_\sigma|}{4|W|}\,.$$

Then by the complement of Markov's inequality:

$$\Pr\left[\sum_{i\in[t]} \mathbb{1}_{X\cap Q_i\neq\emptyset} \leq \frac{4}{3}\frac{d|H_\sigma|}{4|W|}\middle| F(X)=\sigma\right] \geq 1-\frac{3}{4} = \frac{1}{4}\,.$$

Since X is drawn uniformly at random from $\binom{W}{d}$, the random variables $\{\mathbb{1}_{i\in X}\}_{i\in W}$ are negatively associated. Since the maximum function is nondecreasing, for any collection of s disjoint sets $R_1, \ldots, R_s$ of size $\leq \left\lfloor \frac{|W|}{d} \right\rfloor$, the random variables $\{\mathbb{1}_{X\cap R_i\neq\emptyset}\}_{i\in S}$ are also negatively associated. Then applying Lemma 2.3.3:

$$\Pr\left[\sum_{i\in[s]} \mathbb{1}_{X\cap R_i\neq\emptyset} \leq \frac{2}{3}\frac{d\sum_{i\in[s]}|R_i|}{2|W|}\right] \leq \exp\left(-\left(\frac{1}{3}+\frac{2}{3}\ln\frac{2}{3}\right)\frac{d\sum_{i\in[s]}|R_i|}{2|W|}\right) \leq 2^{-\frac{d\sum_{i\in[s]}|R_i|}{50|W|}}\,.$$

We now bound the probability that σ was chosen:

$$\Pr[F(X)=\sigma] \leq \frac{\Pr\left[\sum_{i\in[s]} \mathbb{1}_{X\cap R_i\neq\emptyset} \leq \frac{d|H_\sigma|}{3|W|}\right]}{\Pr\left[\sum_{i\in[s]} \mathbb{1}_{X\cap R_i\neq\emptyset} \leq \frac{d|H_\sigma|}{3|W|} \mid F(X)=\sigma\right]} \leq \frac{2^{-\frac{d|H_\sigma|}{50|W|}}}{1/4}\,.$$

Finally, let $B = \{\sigma \in \Sigma : |H_\sigma| \geq \hat{w}\}$. Then:

$$\begin{aligned}\Pr[|H_{F(X)}| \geq \hat{w}] &= \sum_{\sigma\in B} \Pr[F(X)=\sigma] \\ &\leq 2^z \cdot 4 \cdot 2^{-\frac{d|H_\sigma|}{50|W|}} \leq 2^{z+2}\cdot 2^{-\left(z+2+\log\frac{1}{\alpha}\right)} = \alpha\,.\end{aligned}$$

□

Proof of Theorem 3.8.1. Say the algorithm uses z bits of space. Since it is a random tape algorithm, it has at most $\min(n, 2^z)$ possible outputs. Let t be as defined in Adversary 3.8.1. We prove that, if $z \leq \frac{1}{1600}\left\lfloor\frac{\ell}{3t}\right\rfloor$, the white-box adversary described in Adversary 3.8.1 will make the algorithm err with probability $\geq \min\left(\frac{1}{10}, \frac{\ell^2}{400|U|}\right)$.

The adversary runs the loop starting at Line 4 at most t times, so the total number of elements it sends to the algorithm is

$$\sum_{k=1}^{t} d_k + \lceil \ell/3 \rceil \leq \sum_{k=1}^{t} \lfloor \ell/(3t) \rfloor + \sum_{k=1}^{t} \left\lfloor \ell/(6\cdot 2^k) \right\rfloor + \lceil \ell/3 \rceil \leq \left\lfloor\frac{\ell}{3}\right\rfloor + \left\lfloor\frac{\ell}{6}\right\rfloor + \lceil \ell/3 \rceil \leq \ell\,.$$

Let G be the event in which, on every loop iteration, the set H_k constructed has size $\leq \frac{1}{2}|H_{k-1}|$.

If G holds, then $|H_t| \leq |U|2^{-t} \leq |U|\frac{\ell}{8|U|} \leq \frac{\ell}{8}$, and the adversary will exit the loop at or before the tth iteration. We now bound the probability that G does not hold. For each $k \in [t]$, by Lemma 3.8.2, with probability $\geq 1 - \frac{1}{400t|U|}$, we have:

$$|H_k| \leq \left\lceil 50\frac{z+2+\log(400t|U|)}{d_k}|H_{k-1}| \right\rceil - 1 \leq 50\frac{z+2+\log(400t|U|)}{d_k}|H_{k-1}|\,.$$

Applying $t = \left\lceil \log\frac{8|U|}{\ell} \right\rceil \leq 4|U|$, $|U| \leq 2^z$, and (by Lemma 3.1.2) $z \geq \log(\ell+1) \geq 1$:

$$\begin{aligned}|H_k| &\leq 50\frac{3z+13}{d_k}|H_{k-1}| \leq \frac{800z|H_{k-1}|}{d_k}\\ &\leq \frac{800\frac{1}{1600}\left\lfloor\frac{\ell}{3t}\right\rfloor|H_{k-1}|}{\max(\left\lfloor\frac{\ell}{3t}\right\rfloor, \lfloor \ell/(6\cdot 2^{k-1})\rfloor)} \leq \frac{1}{2}|H_{k-1}|\,.\end{aligned}$$

Then by a union bound, the probability that $|H_k| > \frac{1}{2}|H_{k-1}|$ in some iteration is $\leq \frac{1}{400t|U|}t \leq \frac{1}{400|U|}$.

Assuming event G holds, let τ be the value of k at the time Line 4 exits the loop; then $|H_\tau| \leq \frac{\ell}{6}$. After the adversary reaches Line 15, it will make the algorithm output $\lceil \ell/3 \rceil$ distinct values (or else repeat an element and make a mistake). Let Q be the set of outputs made by the algorithm. Then $|Q \setminus H_\tau| \geq \frac{\ell}{3} - \frac{\ell}{6} \geq \frac{\ell}{6}$. Conditioned on the values of $\sigma_0, \sigma_1, \ldots, \sigma_\tau$, the random sets $S_1, \ldots, S_\tau$ and Q are *independent* of each other. For $i \in [\tau]$, let $Q_i = (Q \cap H_i) \setminus H_{i-1}$. Intuitively, each H_i gives a "safe" subset for outputs in H_{i-1}, and $|Q_i|$ is the number of outputs in H_{i-1} missing that safe subset. Now, for each $k \in [\tau]$, we have

$$\frac{d_k}{4|H_{k-1}|} \geq \frac{\max\left(\lfloor \ell/(3t)\rfloor, \lfloor \ell/(6\cdot 2^k)\rfloor\right)}{|U|2^{-(k-1)}} \geq \frac{\frac{1}{2}\ell/(6\cdot 2^k)}{|U|/2^{k-1}} \geq \frac{\ell}{24|U|}\,. \tag{3.31}$$

Furthermore, letting $\widetilde{Q}_k$ be the first $\lfloor |H_{k-1}| \rfloor d_k$ elements of Q_k (or all of them if Q_k is smaller),

$$\begin{aligned}\Pr[\widetilde{Q}_k \cap S_k \neq \emptyset] &\geq \frac{d_k|\widetilde{Q}_k|}{4|H_{k-1}|} \geq \frac{d_k \min(|Q_k|, \left\lfloor\frac{|H_{k-1}|}{d_k}\right\rfloor)}{4|H_{k-1}|}\\ &\geq \min\left(\frac{|Q_k|d_k}{4|H_{k-1}|}, \frac{|H_{k-1}|-d_k}{4|H_{k-1}|}\right) \geq \min\left(|Q_k|\frac{\ell}{24|U|}, \frac{1}{8}\right)\,,\end{aligned}$$

where in the last step we applied Eq. 3.31 and the fact that $|H_{k-1}| \geq \frac{\ell}{6} \geq 2d_k$. The probability that the algorithm does not produce an invalid output is (still assuming that G holds):

$$\begin{aligned}\Pr\left[Q \cap \bigcup_{k\in[\tau]} S_k = \emptyset\right] &\leq \Pr\left[\bigwedge_{k\in[\tau]} Q_k \cap S_i = \emptyset\right] = \prod_{k\in[\tau]} \Pr[Q_k \cap S_k = \emptyset]\\ &\leq \prod_{k\in[\tau]}\left(1 - \min\left(|Q_k|\frac{\ell}{24|U|}, \frac{1}{8}\right)\right)\end{aligned}$$

$$\leq \max\left(\frac{7}{8}, \exp\left(-\sum_{k\in[\tau]} |Q_k| \frac{\ell}{24|U|}\right)\right)$$
$$\leq \max\left(\frac{7}{8}, \exp\left(-\frac{\ell^2}{144|U|}\right)\right)$$
$$\leq \max\left(\frac{7}{8}, 1 - \frac{\ell^2}{200|U|}\right) .$$

Thus the probability that G holds *and* the algorithm makes a mistake is:

$$\geq \min\left(\frac{1}{8}, \frac{\ell^2}{200|U|}\right) - \frac{1}{400|U|} > \min\left(\frac{1}{10}, \frac{\ell^2}{400|U|}\right) .$$

If the algorithm has error probability $\leq \min\left(\frac{1}{10}, \frac{\ell^2}{400n}\right)$, then we must have

$$z > \frac{1}{1600}\left\lfloor \frac{\ell}{3t} \right\rfloor \geq \frac{1}{1600}\frac{\ell}{6t} = \frac{1}{9600}\frac{\ell}{\left\lceil \log\frac{8|U|}{\ell}\right\rceil} \geq \frac{1}{19200}\frac{\ell}{\log\frac{8|U|}{\ell}} .$$

Since $|U| \leq n$, replacing $|U|$ with n on the right hand side gives a lower bound for z. But as also $|U| \leq 2^z$, we have:

$$z > \frac{1}{19200}\frac{\ell}{z+3} \qquad \Longrightarrow \qquad \left(z + \frac{3}{2}\right)^2 \geq z(z+3) \geq \frac{\ell}{19200} ,$$

which implies $z \geq \sqrt{\ell/19200} - 3/2 \geq \sqrt{\ell/120000}$ (since $z \geq 1$). Thus:

$$z \geq \max\left(\frac{1}{19200}\frac{\ell}{\log(8n/\ell)}, \sqrt{\frac{\ell}{120000}}\right) . \qquad \square$$

3.9 Classical lower bounds

In this section, we present two lower bounds for random oracle algorithms for $\textsc{mif}(n, \ell)$ in the static setting. Each of these bounds is stronger in a different parameter regime; it is still open how to unify them or exactly match the algorithms for the static setting presented in Section 3.2.

3.9.1 By reduction from deterministic

When the worst-case error level δ of a random oracle algorithm is smaller than $1/M$, where M is the number of possible algorithm inputs, then there will exist a fixing of the oracle random string so that the resulting deterministic algorithm works on all inputs. Using this principle, we obtain:

Theorem 3.9.1. *For any $\delta \leq 1/(2n)$, the space complexity for a random oracle algorithm solving*

MIF(n,ℓ) *with error* $\le \delta$ *is*

$$\geq \Omega\left(\sqrt{\min\left(\ell, \frac{\log(1/\delta)}{\log n}\right)} + \min\left(\ell, \frac{\log(1/\delta)}{\log n}\right)\frac{1}{1+\log(n/\ell)}\right).$$

Proof of Theorem 3.9.1. Let t be an integer satisfying $\lceil \frac{n}{t} \rceil^{\lfloor \frac{\ell}{t} \rfloor} < \frac{1}{\delta}$; setting $t = \left\lceil \frac{\ell \log n}{\log \frac{1}{2\delta}} \right\rceil$ suffices, because

$$\log\left(\left\lceil\frac{n}{t}\right\rceil^{\lfloor\frac{\ell}{t}\rfloor}\right) \leq \left\lfloor\frac{\ell}{t}\right\rfloor \log\left\lceil\frac{n}{t}\right\rceil \leq \frac{\ell}{t}\log n \leq \frac{\ell}{\lceil \ell \log n / \log(1/2\delta)\rceil}\log n \leq \frac{\log(1/2\delta)}{\log n}\log n < \log\frac{1}{\delta}.$$

Note also that because $\delta \le 1/(2n)$, $t \le \ell$, and $\lfloor \frac{\ell}{t} \rfloor \ge 1$.

Given a randomized algorithm Π that solves MIF(n,ℓ) with error $\le \delta$ on any input stream, we will show how to construct a randomized algorithm Ψ which solves MIF$(\lceil n/t\rceil, \lfloor \ell/t \rfloor)$ with the same error probability. As there are only $\lceil n/t\rceil^{\lfloor \ell/t\rfloor}$ possible input streams for the MIF$(\lceil n/t\rceil, \lfloor \ell/t \rfloor)$ task, the probability (over randomness used by Ψ) of the event E than an instance A of Ψ succeeds on *any* of the streams in $[\lceil n/t\rceil]^{\lfloor \ell/t\rfloor}$ is $\ge 1-\delta\lceil \frac{n}{t}\rceil^{\lfloor\frac{\ell}{t}\rfloor} > 0$. Therefore, by fixing the random bits of Ψ to some value for which the event E occurs, we obtain a deterministic protocol Φ for MIF$(\lceil n/t\rceil, \lfloor \ell/t \rfloor)$.

We now explain the construction of Ψ given Π. Let $f : [n] \mapsto [\lceil n/t \rceil]$ be the function given by $f(x) = \lfloor x/t \rfloor$. For any $y \in [\lceil n/t\rceil]$, we have that $f^{-1}(y)$ is a nonempty set of size $\le t$. The protocol Ψ starts by initializing an instance A of Π, and sending it $\ell - t\lfloor \ell/t\rfloor$ arbitrary stream elements.

When Ψ receives an element $e \in [\lceil n/t\rceil]$, it sends a sequence of t elements of $[n]$ to A, namely, the elements of $f^{-1}(e)$, in arbitrary order, repeating elements if $|f^{-1}(e)| < t$. To output an element, Ψ queries A to obtain $i \in [n]$, and reports $f(i)$. Assuming A did not fail, $f(i)$ is guaranteed to be a correct answer. If we assume for sake of contradiction that $f(i) = e$ for some element e sent to Ψ earlier, then A must have been sent all elements in $f^{-1}(e)$ – which implies that $i \in f^{-1}(e)$ and that A gave an incorrect output, contradicting the assumption that $f(i) = e$. Thus, we have proven that Ψ fails with no greater probability than Π, which is all that is needed to complete this part of the proof.

Let $S(a,b)$ be the space complexity of deterministic algorithms for MIF(a,b). Having shown that Π needs $\ge S(\lceil \frac{n}{t}\rceil, \lfloor \frac{\ell}{t}\rfloor)$ space, we now substitute in the lower bound from Theorem 3.5.1.

$$\geq S\left(\left\lceil\frac{n}{t}\right\rceil, \left\lfloor\frac{\ell}{t}\right\rfloor\right) \geq \Omega\left(\sqrt{\left\lfloor\frac{\ell}{t}\right\rfloor} + \frac{\lfloor \ell/t\rfloor}{1+\log(\lceil n/t\rceil/\lfloor \ell/t\rfloor)}\right).$$

Because $\lfloor \ell/t\rfloor = \Theta\left(\min\left(\ell, \frac{\log(1/\delta)}{\log n}\right)\right)$, and $\lceil n/t\rceil/\lfloor \ell/t\rfloor = \Theta(n/\ell)$, this simplifies to:

$$\Omega\left(\sqrt{\min\left(\ell, \frac{\log(1/\delta)}{\log n}\right)} + \min\left(\ell, \frac{\log(1/\delta)}{\log n}\right)\frac{1}{1+\log(n/\ell)}\right). \qquad \square$$

3.9.2 By modified AVOID lower bound

The lower bound for random oracle algorithms for $\textsc{mif}(n, \ell)$ in the adversarial setting used a reduction to $\textsc{avoid}$. In this section, we use a variant of the communication lower bound for $\textsc{avoid}$ to obtain a lower bound for random oracle algorithms in the static setting.

Lemma 3.9.2. *Let Σ be a set, n and q integers, and P a function from $\binom{[n]}{q}$ to $\triangle[\Sigma]$, where $|\Sigma| \le 2^z$. Let F be a random function $\binom{[n]}{q} \to \Sigma$ in which for all $x \in \binom{[n]}{q}$ and $\sigma \in \Sigma$, $\Pr[F(x) = \sigma] = P(x)(\sigma)$, and let X be a uniformly random element of $\binom{[n]}{q}$, chosen independently of F. For each $\sigma \in \Sigma$, define*

$$H_\sigma = \left\{ i \in [n] : \Pr\left[i \in X \mid F(X) = \sigma\right] \le \frac{q}{4n} \right\} .$$

For any $\alpha \in (0,1)$,

$$\Pr\left[\left|H_{F(X)}\right| \ge \hat{w}\right] \le \alpha \qquad \textit{where} \qquad \hat{w} := \left\lceil \frac{z + 1 + \log\frac{1}{\alpha}}{q} n \frac{2\ln 2}{1 - \ln 2} \right\rceil .$$

Proof of Lemma 3.9.2. Consider a specific $\sigma \in \Sigma$. By linearity of expectation:

$$\mathbb{E}\left[\sum_{i \in H_\sigma} \mathbb{1}_{i \in X} \,\middle|\, F(X) = \sigma\right] = \sum_{i \in H_\sigma} \Pr\left[i \in X \mid F(X) = \sigma\right] \le \frac{q}{4n} |H_\sigma| .$$

Then by Markov's inequality,

$$\Pr\left[\sum_{i \in H_\sigma} \mathbb{1}_{i \in X} \ge 2 \cdot \frac{q}{4n} |H_\sigma| \,\middle|\, F(X) = \sigma\right] \le \frac{1}{2}$$

$$\text{which implies} \qquad \Pr\left[\sum_{i \in H_\sigma} \mathbb{1}_{i \in X} \le \frac{q}{2n} |H_\sigma| \,\middle|\, F(X) = \sigma\right] \ge \frac{1}{2} .$$

Since X is drawn uniformly at random from $\binom{[n]}{q}$, the random variables $\{\mathbb{1}_{i \in X}\}_{i \in [n]}$ are negatively associated, with $\mathbb{E}\mathbb{1}_{i \in X} = \frac{q}{n}$ for each $i \in [n]$. For any set $A \subseteq [n]$, we use the multiplicative Chernoff bound (Lemma 2.3.3) to bound the probability that X's overlap with A is much smaller than the expected value:

$$\Pr\left[\sum_{i \in A} \mathbb{1}_{i \in X} \le \left(1 - \frac{1}{2}\right) \cdot \frac{q}{n} |A|\right] \le \left(\frac{e^{-1/2}}{(1/2)^{1/2}}\right)^{\frac{q}{n}|A|} = \exp\left(-\frac{1}{2}\left(1 - \ln(2)\right) \frac{q}{n} |A|\right) .$$

We now bound:

$$\Pr\left[F(X) = \sigma\right] \le \frac{\Pr\left[\sum_{i \in H_\sigma} \mathbb{1}_{i \in X} \le \frac{q}{2n} |H_\sigma|\right]}{\Pr\left[\sum_{i \in H_\sigma} \mathbb{1}_{i \in X} \le \frac{q}{2n} |H_\sigma| \,\middle|\, F(X) = \sigma\right]}$$

$$\leq 2\exp\left(-\frac{1}{2}\left(1-\ln(2)\right)\frac{q}{n}\left|H_\sigma\right|\right).$$

Finally, let $B = \{\sigma \in \Sigma : |H_\sigma| \geq \hat{w}\}$. Then:

$$\begin{aligned}
\Pr\left[\left|H_{F(X)}\right| \geq \hat{w}\right] &= \sum_{\sigma\in B} \Pr\left[F(X) = \sigma\right] \\
&\leq 2^z \cdot 2\exp\left(-\frac{1}{2}\left(1-\ln(2)\right)\frac{q}{n}\hat{w}\right) \\
&\leq 2^z \cdot 2\exp\left(-\left(z+1+\log\frac{1}{\alpha}\right)\ln 2\right) \\
&\leq 2^z \cdot 2 \cdot 2^{-\left(z+1+\log\frac{1}{\alpha}\right)} = \alpha\,.
\end{aligned}$$

□

Theorem 3.9.3. *For any $\delta \leq 1/2$ and $\ell \geq 4$, the space complexity of a random oracle algorithm solving* MIF(n,ℓ) *with error $\leq \delta$ (even if just evaluated at the end) is*

$$\geq \Omega\left(\max\left(0, \frac{\ell}{n}\min\left(\ell, \frac{\log\frac{2}{\delta}}{\log\frac{2n}{\ell}}\right) - 50\right)\right).$$

Proof of Theorem 3.9.3. We will consider the performance of a given algorithm using z bits of state when the input consists of a uniformly random subset S in $\binom{[n]}{\lceil \ell/2\rceil}$, presented in sorted order, followed by a uniformly random subset T in $\binom{[n]}{\lfloor \ell/2\rfloor}$, again in sorted order, and show that if z is too small, the algorithm will produce an incorrect output with $\geq \delta$ probability at the end of the stream. Since we are considering a fixed input distribution, without loss of generality we can assume the given algorithm Υ is deterministic.

Let Σ be the set of states of Υ. Let $t = \lceil \ell/2\rceil$ and $t' = \lfloor \ell/2\rfloor$, and define the function $f : \binom{[n]}{t} \mapsto \Sigma$, where $f(S)$ is the state which Υ reaches after being given S. For each state $\sigma \in \Sigma$, define

$$H_\sigma = \left\{i \in [n] : \Pr[i \in S | P(S) = \sigma] \leq \frac{t}{4n}\right\}.$$

Then by Lemma 3.9.2, with $P : \binom{[n]}{t} \mapsto \triangle[\Sigma]$ defined so that set S is mapped to the constant distribution which takes value $f(S)$ with probability 1:

$$\Pr\left[|H_{f(S)}| < w\right] \geq \frac{1}{2} \qquad \text{where} \qquad w := \left\lceil \frac{z+2}{t} n \frac{2\ln 2}{1-\ln 2}\right\rceil.$$

In other words, there is a $\geq 1/2$ probability that, for the state resulting from the random set S, most coordinates are "unsafe", and if chosen by the algorithm at this point will be incorrect with $\geq \frac{t}{4n}$ probability. Say that $t' \geq w - 1$. (We will address the case $t' \leq w-2$ later.) Then in the event that $|H_{f(S)}| < w$, it is in fact possible that the random set $T \supseteq H_{f(S)}$, in which case there is no “safe” output for the algorithm. Formally,

$$\Pr\left[T \supseteq H_{f(S)} | |H_{f(S)}| \leq (w-1)\right] \geq \frac{\binom{n-(w-1)}{t'-(w-1)}}{\binom{n}{t'}} = \frac{\binom{t'}{w-1}}{\binom{n}{w-1}} \geq \frac{(\frac{t'}{w-1})^{w-1}}{(\frac{en}{w-1})^{w-1}} \geq \left(\frac{t'}{en}\right)^{w-1} . \tag{3.32}$$

Let $I \in [n]$ be the random variable for the choice of the algorithm. The failure probability of the algorithm is:

$$\begin{aligned}\Pr[I \in S \cup T] &\geq \sum_{\sigma \in \Sigma : |H_\sigma| < w} \Pr[I \in S \cup T \wedge f(S) = \sigma] \\ &= \sum_{\sigma \in \Sigma : |H_\sigma| < w} \Pr[f(S) = \sigma] \Pr[I \in S \cup T \mid f(S) = \sigma] . \end{aligned} \tag{3.33}$$

We now proceed to lower bound the terms $\Pr[I \in S \cup T \mid f(S) = \sigma]$. First, note that

$$\begin{aligned}\Pr[I \in S \cup T \mid f(S) = \sigma] &\geq \Pr[I \in S \cup T \wedge T \supseteq H_\sigma \mid f(S) = \sigma] \\ &= \Pr[I \in S \cup T \mid T \supseteq H_\sigma \wedge f(S) = \sigma] \Pr[T \supseteq H_\sigma \mid f(S) = \sigma] \\ &= \Pr[I \in S \cup T \mid T \supseteq H_\sigma \wedge f(S) = \sigma] \Pr[T \supseteq H_\sigma] . \end{aligned}$$

where the last line uses the independence of S and T. Next, because $\Pr[A \cup B] \geq \Pr[B] + \Pr[A \mid \neg B](1 - \Pr[B]) \geq \Pr[A \mid \neg B]$, we have:

$$\Pr[I \in S \cup T \mid T \supseteq H_\sigma \wedge f(S) = \sigma] \geq \Pr[I \in S \mid I \notin T \wedge T \supseteq H_\sigma \wedge f(S) = \sigma] .$$

The random variable I is a function of T and σ alone. Also, $I \notin T$ and $T \supseteq H_\sigma$ implies $I \in [n] \setminus H_\sigma$. Consequently, also using the fact that T is uniformly random,

$$\begin{aligned}\Pr[I \in S \mid I \notin T \wedge T \supseteq H_\sigma \wedge f(S) = \sigma] &\geq \min_{j \in [n] \setminus H_\sigma} \Pr[j \in S \mid j \notin T \wedge T \supseteq H_\sigma \wedge f(S) = \sigma] \\ &= \min_{j \in [n] \setminus H_\sigma} \Pr[j \in S \mid f(S) = \sigma] . \end{aligned}$$

By the definition of H_σ, this last quantity is $\geq \frac{t}{4n}$. Recalling Eqs. 3.33 and 3.32:

$$\begin{aligned}\Pr[I \in S \cup T] &\geq \sum_{\sigma \in \Sigma : |H_\sigma| < w} \Pr[f(S) = \sigma] \frac{t'}{4n} \Pr[T \supseteq H_\sigma] \\ &\geq \sum_{\sigma \in \Sigma : |H_\sigma| < w} \Pr[f(S) = \sigma] \frac{t}{4n} \left(\frac{t'}{en}\right)^{w-1} \\ &= \frac{t}{4n} \left(\frac{t'}{en}\right)^{w-1} \Pr[|H_{f(S)}| < w] \\ &\geq \frac{1}{8e} \left(\frac{t'}{en}\right)^{w} . \end{aligned}$$

(Since $t = \lceil \ell/2 \rceil \geq \lfloor \ell/2 \rfloor = t'$.) We have assumed that the algorithm fails with probability $\leq \delta$, hence:

$$\delta \geq \frac{1}{8e} \cdot \left(\frac{t}{en}\right)^w \qquad \implies w \geq \frac{\log(8e/\delta)}{\log(8n/t')} .$$

Now, if it is not the case that $t' \geq w - 1$, then we have $t' < w - 1$, and hence $t' + 2 \leq w$. Thus:

$$\min\left(t' + 2, \frac{\log(8e/\delta)}{\log(8n/t')}\right) \leq w = \left\lceil \frac{(z+2)n}{t} \frac{2\ln 2}{1 - \ln 2} \right\rceil .$$

Solving for z, we obtain:

$$\begin{aligned} z &\geq (w-1)\frac{t}{n} \cdot \frac{1 - \ln 2}{2 \ln 2} - 2 \\ &\geq \min\left(\lfloor \ell/2 \rfloor + 2, \frac{\log(8e/\delta)}{\log(8n/\lfloor \ell/2 \rfloor)}\right) \frac{\lceil \ell/2 \rceil}{n} \frac{1 - \ln 2}{2 \ln 2} - 2 \\ &\geq \min\left(\frac{\ell}{2}, \frac{\log(2/\delta)}{5\log(2n/\ell)}\right) \frac{\ell}{5n} - 2 \\ &= \Omega\left(\max\left(0, \frac{\ell}{n} \min\left(\ell, \frac{\log \frac{2}{\delta}}{\log \frac{2n}{\ell}}\right) - 50\right)\right) . \end{aligned}$$

This lower bound is zero for constant δ and low ℓ/n. This is not a mistake, as there is a zero bit, one state randomized protocol that outputs a random integer in $[n]$, and succeeds with $\geq \ell/n$ probability in the static setting. □

3.10 Random tape upper bound, adversarial setting

In this section, we describe an adversarially robust random tape algorithm for MIF(n, ℓ) which obtains error $\leq \delta$ in the adversarial setting. In order to explain why this algorithm has the structure it does, it is helpful to first revisit the random oracle and random seed algorithms for the problem.

The random oracle algorithm and its adversaries. Recall from Section 3.3.3 the simple random oracle algorithm for MIF(n, ℓ), Algorithm 3.3.1. It interprets its oracle random string as a uniformly random sequence L containing $\ell + 1$ distinct elements in $[n]$. As this algorithm processes the input stream, it keeps track of which elements in L were in the input stream so far (were "covered"). The algorithm reports as its output the first element of L which is not covered. Because L comes from the oracle random string, the space cost of the algorithm is just the cost of keeping track of the set J of covered positions in L. We will explain why that can be done using only $O((\ell^2/n + 1)\log \ell)$ space, in expectation.

An adversary for the algorithm only has two reasonable strategies for choosing the next input. It can "echo" back the current algorithm output to be the next input to the algorithm. It can also choose the next input to be a value from the set U of values that are neither an earlier input nor the current output – but because L is chosen uniformly at random, one can show that the adversary can do no better than picking the next input uniformly at random from U. (The third option, of choosing an old input, has no effect on the algorithm.) When the algorithm is run against an algorithm that chooses inputs using a mixture of the echo and random strategies, the set J will be structured as the union of a contiguous interval starting at 1 (corresponding to the positions in L covered by the echo strategy) and a random set of size $O(\ell^2/n)$ (corresponding to positions in L covered by the random strategy.) Together, these parts of J can be encoded using $O((\ell^2/n + 1) \log \ell)$ bits, in expectation.

Delaying the echo strategy. If we implemented the above random oracle algorithm as a random seed algorithm, we would need $\Omega(\ell)$ bits of space, just to store the random list L. But why does L need to have length $\ell + 1$? This length is needed for the algorithm to be resilient to the echo strategy, which covers one new element on L on every step; if L were shorter, the echo strategy could entirely cover it, making the algorithm run out of possible values to output. The random seed algorithm for $\textsc{mif}(n, \ell)$ works by making the echo strategy less effective, ensuring that multiple steps are needed for it to cover another element of L. It does this by partitioning $[n]$ into $\Theta(\ell)$ disjoint subsets ("blocks") of size $\Theta(n/\ell)$; let $\mathcal{B}$ be the set of blocks. Then, instead of having L be a random list over element of $[n]$, it has L be a random list over elements of $\mathcal{B}$. We will now say that a block is covered if *any* element of that block was an input. Instead of outputting the first uncovered element in L, the algorithm will run a deterministic algorithm for MIF *inside* the block corresponding to the first uncovered block of L, and report outputs from that; and will only move on to the next uncovered block when the nested algorithm stops. See Algorithm 3.7.1 from Section 3.7 for the details of this design. Because the analogue of the echo strategy now requires many more inputs to cover a block, we can make the list L shorter. This change will not make the random strategy much more effective. In the end, after balancing the length of the list with the cost of the nested algorithm, the optimal list length for random seed algorithm will be $O(\ell^2/n + \sqrt{\ell})$.

The recursive random tape algorithm. The random seed algorithm for $\textsc{mif}(n, \ell)$ used the construction of Algorithm 3.7.1 to build on top of an "inner" deterministic algorithm.[17] To get an efficient random tape algorithm, we can recursively apply the construction of Algorithm 3.7.1 $d-1$ times, for $d = O(\min(\log \ell, \log n / \log \ell))$ times; at the end of this recursion, we can use a simple

[17]The construction initializes the inner algorithm multiple times. For the random seed model, every time an inner randomized algorithm is initialized would require a new batch of random bits, and all batches would be counted toward the space cost of the algorithm. Using a deterministic inner algorithm avoids this cost.

deterministic algorithm for MIF. The optimal lengths of the random lists used at each level of the recursion are determined by balancing the costs of the different recursion levels. We end up choosing list lengths that all bounded by a quantity that lies between $O(\ell^{1/d})$ and $O(\ell^{1/(d-1)})$. The exact details we defer to the final version of our algorithm, Algorithm 3.10.1. This final version looks somewhat different from the recursive construction in Algorithm 3.7.1, because we have unraveled the recursive framing to allow for better control over parameters and to permit a simpler error analysis that must only bound the probability of a single bad event.

3.10.1 Definitions and the random tree view

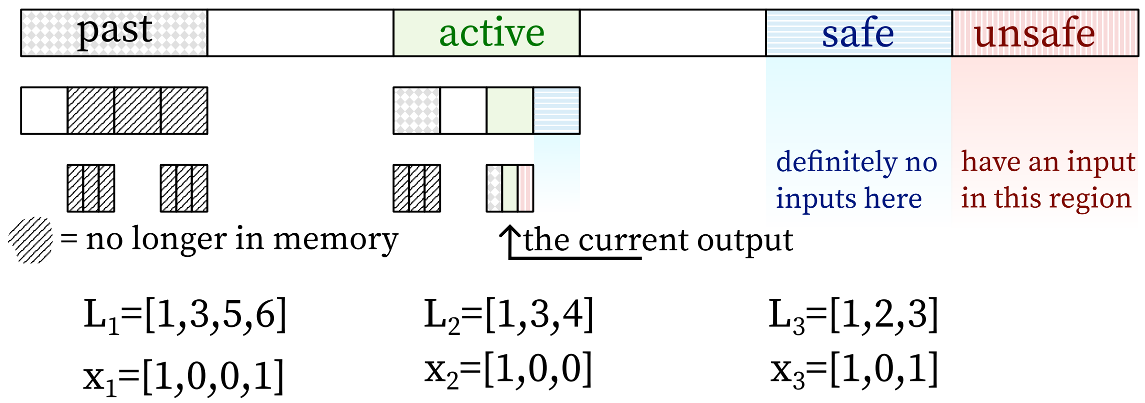


Figure 3.9: Diagram showing the state of the algorithm in Algorithm 3.10.1 and how it relates to the parts of the implicit random tree that the algorithm traverses. This example uses parameters $d = 3$, $w_1 = 6, w_2 = 4, w_3 = 3$, and $b_1 = 4, b_2 = 3, b_3 = 3$.

In order to prove that the algorithm in Algorithm 3.10.1 is correct, we will need some additional notation. Let $d, \alpha, b_1, \ldots, b_d, w_1, \ldots, w_d$ be as defined in Algorithm 3.10.1. It is helpful to view this algorithm as traversing over the leaves of a random tree of height d, in which:

- Every node v in the tree is associated with a subset S_v of $[n]$. We say a node is at level i if it is at depth $i-1$.
- The root ρ of the tree has S_ρ of size $\prod_{i=1}^{d} w_i$, and is at level 1
- Each node v at level i (depth $i-1$) has b_{i+1} children; the set S_v is partitioned into w_{i+1} parts of equal size, and each child of v is associated with a random and unique one of these parts
- Each leaf node u, at depth d, has $|S_u| = 1$ and is associated with a unique integer in $[n]$. There are $\prod_{i=1}^{d} b_i$ leaf nodes in total.

See for example Figure 3.9. The algorithm maintains a view of just the branch of the tree from the root to the current leaf node. Its output will be the number associated to this leaf. For each node v on this branch, at level $i \in [d]$ (depth $i-1$), it keeps a record of the positions $L_i \in [w_i]^{b_i}$ of its children, and a record $x_i \in \{0,1\}_i^b$ indicating their status. There are four categories for child nodes:

- A node is PAST if the traversal over the tree passed through and leaf the node; past nodes are marked with a 1.
- A node is ACTIVE if it is on the branch to the current leaf node; this is the node with the lowest index which is marked with a 0
- A node v is SAFE if it comes after the active node, and the adversary has never sent an input in S_v; safe nodes are marked with a 0
- A node v is UNSAFE if it comes after the active node, and the adversary did send an input in S_v; unsafe nodes are marked with a 1

The algorithm maintains these records as the adversary sends new inputs, marking safe child nodes v as unsafe if an element in S_v is received. The current leaf node is found by, from the root, following the chain of active nodes. If the adversary sends the value of the current leaf node, the algorithm will mark it by setting the corresponding entry in x_d to 1, thereby changing the value of the current active node. If it turns out that x_d is an all-1s vector, then the adversary has sent an input for every child of the level d node on the current branch, so the algorithm marks the current active child of the level $d-1$ node with a 1, thereby moving the current branch to a new level d node, u. (If the level $d-1$ node has no children marked with a 0 after this, we repeat the process at level $d-2$, and so on.) It then "loads the positions of the children of u"– the tree being randomly generated, this is implemented by L_d being randomly sampled and x_d set to be all zeros – and proceeds.

While there are $\prod_{i=1}^{d} b_i$ leaf nodes in the ideal random tree, the algorithm's traversal of them may skip a fraction, because they (or one of their ancestors) were marked as unsafe. We say that all leaf nodes skipped or traversed over have been KILLED.

3.10.2 Setting parameters and bounding space

Algorithm 3.10.1 uses the following lemma to set some of its parameters; the specific rounding scheme for the values $b_2, \ldots, b_{d-1}$ ensures that b_1 can decrease relatively smoothly as ℓ decreases. [18]

[18] Setting all $b_2 = \ldots = b_{d-1}$ to $\lfloor \alpha \rfloor$ can lead to having b_1 be significantly larger than necessary (by up to a factor $(3/2)^d = \ell^{\Omega(1)}$ for certain ℓ,n); setting all $b_2 = \ldots = b_{d-1}$ to $\lceil \alpha \rceil$ would violate the $\prod_{i=1}^{d} w_i \leq n$ constraint.

Algorithm 3.10.1 Adversarially robust random tape algorithm for $\textsc{mif}(n, \ell)$ with error $\leq \delta$

```
Requirements: $\ell \leq n/64$ and $\ell \geq 4$.
Parameters: $d = \min(\lceil \log \ell \rceil, \left\lfloor 2\frac{\log(n/4)}{\log 16\ell} \right\rfloor)$.
$\alpha = \begin{cases} 2 & \text{if } \lceil \log \ell \rceil < \left\lfloor 2\frac{\log(n/4)}{\log 16\ell} \right\rfloor \\ \frac{(4\ell)^{2/(d-1)}}{(n/4)^{2/(d(d-1))}} & \text{otherwise} \end{cases}$
Let $u$ be chosen via Lemma 3.10.1, so that $\alpha^{d-2} \leq \prod_{i=2}^{d-1} b_i \leq 2\alpha^{d-2}$
$b_2 = \ldots = b_u = \lceil \alpha \rceil$, and $b_{u+1} = \ldots = b_{d-1} = \lfloor \alpha \rfloor$
$b_1 = \min(\ell + 1, \lceil 8\alpha \rceil + \lceil 3 \log 1/\delta \rceil)$; and $b_d = \lceil \frac{\ell}{\alpha^{d-1}} \rceil$
$w_1 = 16\ell$; and for each $i \in \{2, \ldots, d\}$, $w_i = \prod_{j=i}^{d} b_j$.
Let $\iota : [w_1] \times [w_2] \times \cdots \times [w_d] \to [n]$ be an arbitrary injective function

Initialization:
1:  for $i \in [d]$ do
2:      $L_i \leftarrow$ random sequence without repetition in $[w_i]^{b_i}$
3:      $x_i \leftarrow (0, \ldots, 0) \in \{0,1\}_i^b$

Update($a \in [n]$):
4:  if $a \notin \iota^{-1}([n])$ then return      ▷ Any integer not in $\iota^{-1}([n])$ can never be an output
5:  $v_1, \ldots, v_d = \iota^{-1}(a)$      ▷ Map input into $[w_1] \times \ldots \times [w_d]$
6:  For $i \in [d]$, define $c_i = \min |\{j : x_i[j] = 0\}|$
7:  if for all $i \in [d]$, $v_i = L_i[c_i]$ then
8:      ▷ Move to the next leaf node, sampling new child node positions as necessary
9:      for $i = d, \ldots, 1$ do
10:         $x_i[c_i] \leftarrow 1$
11:         if $x_i$ is the all-1s vector then
12:             if $i = 1$ then abort      ▷ If we reach $i = 1$, then even the root node is full
13:             $L_i \leftarrow$ random sequence without repetition in $[w_i]^{b_i}$
14:             $x_i \leftarrow (0, \ldots, 0)$
15:         else break
16: else
17:     ▷ Mark a branch as unsafe, if there was a hit
18:     Let $j$ be the smallest integer in $[d]$ for which $v_j \neq L_j[c_j]$.
19:     if $\exists y \in [b_j]$ for which $L_j[y] = v_j$ then
20:         $x_j[y] \leftarrow 1$

Output $\to [n]$:
21: For $i \in [d]$, define $c_i = \min |\{j : x_i[j] = 0\}|$
22: return $\iota[(L_1[c_1], L_2[c_2], \ldots, L_d[c_d])]$
```

Lemma 3.10.1. *Let $\alpha \geq 1$. Then for all $k \geq 0$, there exists an integer u depending on α and k so that*

$$\alpha^k \leq \lceil \alpha \rceil^u \lfloor \alpha \rfloor^{k-u} \leq 2\alpha^k \,.$$

Proof of Lemma 3.10.1. If α is an integer, we are done. Otherwise, with

$$u = \left\lceil \frac{k \log(\alpha/\lfloor \alpha \rfloor)}{\log(\lceil \alpha \rceil / \lfloor \alpha \rfloor)} \right\rceil, \quad \text{we have} \quad \lceil \alpha \rceil^u \lfloor \alpha \rfloor^{k-u} = \lfloor \alpha \rfloor^k (\lceil \alpha \rceil / \lfloor \alpha \rfloor)^u \geq \lfloor \alpha \rfloor^k (\lceil \alpha / \lfloor \alpha \rfloor \rceil)^k = \alpha^k \,.$$

Similarly, $\lceil \alpha \rceil^u \lfloor \alpha \rfloor^{k-u} \leq \alpha^k \lceil \alpha \rceil / \lfloor \alpha \rfloor$, which is $\leq 2\alpha^k$ because $\alpha \geq 1$ implies $\lceil \alpha \rceil / \lfloor \alpha \rfloor \leq 2$. $\square$

The following lemma establishing properties of the algorithm parameters is straightforward but tedious:

Lemma 3.10.2. *The parameters of Algorithm 3.10.1 satisfy the following conditions:*

$$\prod_{i=2}^{d} b_i \geq \frac{\ell}{\alpha} \tag{3.34}$$

$$\prod_{i=2}^{d} b_i \leq \frac{4\ell}{\alpha} \tag{3.35}$$

$$\prod_{i \in [d]} w_i \leq n \,. \tag{3.36}$$

Proof of Lemma 3.10.2. First, we handle the case where $\lceil \log \ell \rceil < \left\lfloor 2^{\frac{\log(n/4)}{\log 16\ell}} \right\rfloor$. Then $d = \lceil \log \ell \rceil \leq \left\lfloor 2^{\frac{\log(n/4)}{\log 16\ell}} \right\rfloor - 1$, and $\alpha = 2$. Note that $\frac{\ell}{2^{d-1}} \geq 1$ since $2^{d-1} \leq 2^{\lceil \ell \rceil - 1} \leq 2\ell/2 = \ell$.

$$\prod_{i=2}^{d} b_i = \left\lceil \frac{\ell}{2^{d-1}} \right\rceil 2^{d-2} \geq \frac{\ell}{2} = \frac{\ell}{\alpha}$$

$$\prod_{i=2}^{d} b_i = \left\lceil \frac{\ell}{2^{d-1}} \right\rceil 2^{d-2} \leq 2\frac{\ell}{2} \leq \frac{4\ell}{\alpha} \,.$$

Since $b_d = \lceil \frac{\ell}{2^{d-1}} \rceil = \left\lceil \frac{2\ell}{2^{\lceil \log \ell \rceil}} \right\rceil \leq 2$,

$$\begin{aligned}
\prod_{i \in [d]} w_i = (16\ell) \prod_{i=2}^{d} \prod_{j=i}^{d} b_j \leq (16\ell) \prod_{i=2}^{d} 2^{d-i+1} &= (16\ell) 2^{d(d-1)/2} \\
&\leq (16\ell)(2^{\lceil \log \ell \rceil})^{(d-1)/2} \leq (16\ell)(2\ell)^{(d-1)/2} \\
&\leq (16\ell)(2\ell)^{(\left\lfloor 2^{\frac{\log(n/4)}{\log 16\ell}} \right\rfloor - 2)/2} \\
&\leq (16\ell)(2\ell)^{\frac{\log(n/4)}{\log 16\ell} - 1}
\end{aligned}$$

$$\leq (16\ell)^{\frac{\log(n/4)}{\log 16\ell}} \leq \frac{n}{4} \leq n\,.$$

Second, we consider the case where $d = \left\lfloor 2\frac{\log(n/4)}{\log(16\ell)} \right\rfloor$. Because $n \geq 64\ell$, $d \geq 2$, and so

$$d = \left\lfloor 2\frac{\log(n/4)}{\log 16\ell} \right\rfloor \geq \frac{2}{3} \cdot 2\frac{\log(n/4)}{\log 16\ell} = \frac{\log(n/4)}{\frac{3}{4}\log 16\ell} \geq \frac{\log(n/4)}{\log 4\ell}\,.$$

The second inequality used that $\frac{3}{4}(4+\log \ell) \leq (2+\log \ell)$ for $\ell \geq 4$. Consequently,

$$\alpha = \left(\frac{(4\ell)^d}{n/4}\right)^{\frac{2}{d(d-1)}} \geq \left(\frac{(4\ell)^{\frac{\log(n/4)}{\log 4\ell}}}{n/4}\right)^{\frac{2}{d(d-1)}} = \left(\frac{n/4}{n/4}\right)^{\frac{2}{d(d-1)}} = 1\,.$$

We now prove Eq. (3.34). Because $\prod_{i=2}^{d-1} b_i \geq \alpha^{d-2}$,

$$\prod_{i=2}^{d} b_i = \left\lceil \frac{\ell}{\alpha^{d-1}} \right\rceil \prod_{i=2}^{d-1} b_i \geq \frac{\ell}{\alpha \prod_{i=2}^{d-1} b_i} \cdot \prod_{i=2}^{d-1} b_i = \frac{\ell}{\alpha}\,.$$

For Eq. (3.36), we observe that

$$d \leq 2\frac{\log(n/4)}{\log(16\ell)} \qquad \Longrightarrow \qquad 16\ell \leq (n/4)^{2/d} \qquad \Longrightarrow \qquad \ell \geq \alpha^{d-1} = \frac{(4\ell)^2}{(n/4)^{2/d}}\,,$$

and thus $\ell/\alpha^{d-1} \geq 1$, so $b_d = \lceil \ell/\alpha^{d-1} \rceil \leq 2\ell/\alpha^{d-1}$. Then since $\prod_{i=2}^{d-1} b_i \leq 2\alpha^{d-2}$,

$$\prod_{i=2}^{d} b_i \leq \frac{2\ell}{\alpha^{d-1}} \prod_{i=2}^{d} b_i \leq \frac{4\ell}{\alpha \prod_{i=2}^{d-1} b_i} \cdot \prod_{i=2}^{d-1} b_i \leq \frac{4\ell}{\alpha}\,.$$

Finally, we prove Eq. (3.36). As noted above,

$$b_d \leq \frac{2\ell}{\alpha^{d-1}} \leq \frac{4\ell}{\alpha \prod_{i=2}^{d-1} b_j}\,.$$

Applying this fact to bound the left hand side of Eq. (3.36) gives:

$$\begin{aligned}
\prod_{i\in[d]} w_i &= (16\ell) \prod_{i=2}^{d} \prod_{j=i}^{d} b_j = 16\ell (b_d)^d \prod_{i=2}^{d-1} \prod_{j=i}^{d-1} b_j \\
&\leq 16\ell \left(\frac{4\ell}{\alpha \prod_{i=2}^{d-1} b_j}\right)^{d-1} \prod_{i=2}^{d-1} \prod_{j=i}^{d-1} b_j \\
&\leq \frac{4 \cdot (4\ell)^d}{\alpha^{d-1}} \frac{1}{\prod_{i=2}^{d-1} \prod_{j=2}^{i-1} b_j}
\end{aligned}$$

$$\le \frac{4\cdot(4\ell)^d}{\alpha^{d-1}}\frac{1}{\alpha^{(d-1)(d-2)/2}} \qquad \text{since } \prod_{j=2}^{d-1} b_j \ge \alpha^{d-2} \text{ and } b_2 \ge b_3 \ge \ldots \ge b_{d-1}$$

$$= \frac{4\cdot(4\ell)^d}{\alpha^{d(d-1)/2}} = \frac{4\cdot(4\ell)^d}{\frac{(4\ell)^d}{n/4}} = n\,. \qquad \square$$

Lemma 3.10.3. *Algorithm 3.10.1 uses*

$$O\left(\left\lceil \frac{(4\ell)^{2/(d-1)}}{(n/4)^{2/(d(d-1))}} \right\rceil (\log \ell)^2 + \min(\ell, \log 1/\delta)\log \ell\right) \tag{3.37}$$

bits of space, where $d = \min\left(\lceil \log \ell \rceil, \left\lfloor 2\frac{\log(n/4)}{\log(16\ell)} \right\rfloor\right)$. *A weaker upper bound on this is:*

$$O\left(\ell^{\frac{\log \ell}{\log n}}(\log \ell)^2 + \min(\ell, \log 1/\delta)\log \ell\right).$$

Proof of Lemma 3.10.3. Algorithm 3.10.1 only stores two types of data: for each $i \in [d]$, the vectors $L_i \in [w_i]^{b_i}$, and the vectors $x_i \in \{0,1\}^{b_i}$. These can be stored using $b_i \log w_i$ and b_i, bits respectively, for a total of:

$$\sum_{i\in[d]} b_i \log(2w_i) \le b_1 \log(2w_1) + \sum_{i=2}^{d} b_i \log(2w_i)$$
$$\le b_1 \log(32\ell) + \sum_{i=2}^{d} b_i \log(2\prod_{j=i}^{d} b_i) \le \sum_{i=1}^{d} b_i \log(32\ell)\,,$$

since by Eq. 3.35, $\prod_{j=i}^{d} b_i \le 4\ell$.

We now observe that $b_d \le \lceil \alpha \rceil$. If $d = \lceil \log \ell \rceil$, then $\alpha = b_2 = \ldots = b_{d-1} = 2$ and $b_d = \lceil \ell/\alpha^{d-1} \rceil \le 2$. On the other hand, if $d = \left\lfloor 2\frac{\log(n/4)}{\log(16\ell)} \right\rfloor$, then $\alpha = \frac{(4\ell)^{2/(d-1)}}{(n/4)^{2/(d(d-1))}}$. We have:

$$(4\ell)^{d+1} = (4\ell)^{\left\lfloor 2\frac{\log(n/4)}{\log(16\ell)} \right\rfloor + 1} \ge (4\ell)^{2\frac{\log(n/4)}{\log(16\ell)}} = (n/4)^2\,,$$

which implies

$$\alpha = \frac{(4\ell)^{2/(d-1)}}{(n/4)^{2/(d(d-1))}} \ge \frac{(4\ell)^{2/(d-1)}}{(4\ell)^{(d+1)/(d(d-1))}} = (4\ell)^{(2-\frac{d+1}{d})\cdot\frac{1}{d-1}} = (4\ell)^{\frac{1}{d}}\,.$$

Consequently,

$$b_d = \left\lceil \frac{\ell}{\alpha^{d-1}} \right\rceil = \left\lceil \frac{1}{4}\frac{4\ell}{\alpha^{d-1}} \right\rceil \le \left\lceil \frac{1}{4}(4\ell)^{1/d} \right\rceil \le (4\ell)^{1/d} \le \alpha\,.$$

With the bound on b_d, and the fact that $\alpha \ge 1$ in both cases, and that $d \le \lceil \log \ell \rceil$ we obtain a bound on the sum of the list lengths:

$$\begin{aligned}\sum_{i\in[d]} b_i &\le \min(\ell+1, \lceil 8\alpha\rceil + \lceil 3\log 1/\delta\rceil) + (d-1)\lceil\alpha\rceil \\ &\le \min(\ell, \lceil 3\log 1/\delta\rceil) + (7+d)2\alpha \\ &\le \min(\ell, \lceil 3\log 1/\delta\rceil) + 32\log\ell \left\lceil \frac{(4\ell)^{2/(d-1)}}{(n/4)^{2/(d(d-1))}} \right\rceil .\end{aligned}$$

Multiplying this last quantity by $\log(32\ell)$ gives a space bound.

To obtain a much weaker, but somewhat more comprehensible upper bound on α, when $d = \left\lfloor 2\frac{\log(n/4)}{\log(16\ell)} \right\rfloor$, we note that:

$$\begin{aligned}\max_{\lambda\in\mathbb{N}\cap[2,\infty)} \log\frac{(4\ell)^{2/(\lambda-1)}}{(n/4)^{2/(\lambda(\lambda-1))}} &\le \log \max_{\lambda\in\mathbb{R}\cap[2,\infty)} \left(\frac{2}{\lambda-1}\log(4\ell) - \frac{2}{\lambda(\lambda-1)}\log(n/4)\right) \\ &\le \log\left(2\log(4\ell) \max_{\lambda\in\mathbb{R}\cap[2,\infty)} \left(\frac{1}{\lambda-1} - \frac{1}{\lambda(\lambda-1)}\frac{\log(n/4)}{\log(4\ell)}\right)\right) .\end{aligned}$$

Let $\gamma = \frac{\log(n/4)}{\log(4\ell)}$; this is ≥ 1. Let $f(x) = \frac{1}{x-1}(1-\frac{\gamma}{x})$. We will now prove that $\max_{x\ge 2} f(x) \le \frac{1}{2\gamma}$. We note that when $x = 2$, we have:

$$f(2) = 1 - \frac{\gamma}{2} \le \frac{1}{2\gamma} .$$

Checking the other endpoint, we have:

$$\lim_{x\to\infty} \frac{1}{x-1}(1-\frac{\gamma}{x}) = 0 .$$

Since $f(x)$ is differentiable on $[2,\infty)$, if it has a maximum other than at the endpoints, then it will occur when $\frac{d}{dx}f(x) = 0$. Solving this equation, we obtain:

$$\frac{d}{dx}f(x) = -\frac{1}{(x-1)^2} + \frac{\gamma\,(2x-1)}{(x\,(x-1))^2} = -\frac{1}{(x-1)^2}\left[1 - \frac{\gamma\,(2x-1)}{x^2}\right] = 0 ,$$

which is true iff $x^2 = \gamma(2x-1)$. The solutions to the quadratic equation are

$$x = \gamma - \sqrt{\gamma\,(\gamma-1)} \qquad \text{and} \qquad x = \gamma + \sqrt{\gamma\,(\gamma-1)} .$$

Since $\gamma \ge 1$, the $-$ branch has $x \le 1$, which is not in $[2,\infty)$. The $+$ branch is only in $[2,\infty)$ if $\gamma \ge \frac{4}{3}$. The value of $f(x)$ in this case is:

$$f(\gamma + \sqrt{\gamma\,(\gamma-1)}) = \frac{1}{\gamma + \sqrt{\gamma\,(\gamma-1)} - 1}\left(1 - \frac{\gamma}{\gamma+\sqrt{\gamma\,(\gamma-1)}}\right)$$

$$= \frac{\sqrt{\gamma(\gamma-1)}}{2\gamma - 1 + 2\sqrt{\gamma(\gamma-1)}}$$

$$\le \frac{1}{4\sqrt{\gamma(\gamma-1)}} \qquad \text{(since } \sqrt{\gamma(\gamma-1)} \le 2\gamma - 1 \text{ for all } \gamma \ge 1\text{)}$$

$$\le \frac{1}{2\gamma}\,. \qquad \text{(since } \gamma \le 2\sqrt{\gamma(\gamma-1)} \text{ for all } \gamma \ge \frac{4}{3}\text{)}$$

Thus, if $f(x)$ does have a maximum in $[2, \infty)$, it is $\le \frac{1}{2\gamma}$. We conclude that $f(x) \le \frac{1}{2\gamma}$ in all cases. This proves:

$$\log \alpha \le \max_{\lambda \in \mathbb{N}} \log \frac{(4\ell)^{2/(\lambda-1)}}{(n/4)^{2/(\lambda(\lambda-1))}} \le \log \left(2 \log(4\ell) \frac{1}{2^{\frac{\log(n/4)}{\log(4\ell)}}} \right) \le \log \frac{\log(4\ell)^2}{\log(n/4)}\,. \qquad \square$$

3.10.3 The error bound

We now prove the main lemma:

Lemma 3.10.4. *Algorithm 3.10.1 has error $\le \delta$ in the adversarial setting.*

Proof of Lemma 3.10.4. We prove, using a charging scheme, that the probability of all leaf nodes in the random tree traversed by the algorithm being killed is $\le \delta$.

The input of the adversary at any step falls into one of $d+2$ categories. For each $i \in [d]$, the adversary could add an input which intersects the list of unrevealed child positions of the level i node, possibly killing $\prod_{j=i+1}^{d} b_i$ leaf nodes if it guesses correctly. It could also send the value of the current leaf node, thereby killing it (and only it). Finally, the adversary's input could be entirely wasted (outside $\iota([w_1] \times \ldots \times [w_d]$, repeating an input it made before, or in the region corresponding to one of the past nodes in the random tree); then no leaf nodes would be killed.

As the algorithm proceeds, for each node in the random tree (other than the root), we accumulate charge. When the algorithm's current branch changes to use new nodes, the charge on the old nodes is kept, and the new nodes start at charge 0.

When the algorithm makes a query at level i, for $i \in \{2, \ldots, d\}$, it *first* deposits one unit of charge at the active level i node. Then, if the query was a hit (i.e, ruled out some future subtree and made a child of a node in the current branch change from "safe" to "unsafe"), increase the number of killed nodes by the number of leaves for the subtree (namely, $\prod_{j=i+1}^{d} b_j$), and remove up to that amount of charge from the node. The definitions of $(w_j)_{j=2,\ldots,d}$ ensure that $\prod_{j=i+1}^{d} b_j = w_j / b_j$.

For the tth query, let K_t be the number of killed leaf nodes on the query, minus any accumulated charge on the node. Say the adversary picks a node at level i for $i \in \{2, \ldots, d-1\}$, and that node has $\hat{w}$ unexplored subtree regions (i.e, neither revealed because the algorithm produced outputs in them, nor was there a query at that subtree region in the past), and $\hat{b}$ gives the number of subtrees within this unexplored region. If $\hat{b} = 0$, $\mathbb{E}[K_t | K_1, \ldots, K_{t-1}] = 0$. Otherwise, let u be the number of subtrees which were revealed by the algorithm so far; we have $\hat{w} \leq w_j - u$ and $\hat{b} \leq b_j - u$. Then when we condition on the past increases in charge, the subtree regions within the unexplored region are still uniformly random; hence the probability of hitting a subtree is $\hat{b}/\hat{w}$. The number of leaf nodes killed by a hit is w_j/b_j. The total charge currently at the node must be $\geq (w_j - u - \hat{w}) - (\frac{w_j}{b_j})(b_j - u - \hat{b}) = \hat{b}\frac{w_j}{b_j} - \hat{w} + (\frac{w_j}{b_j} - 1)u \geq \hat{b}\frac{w_j}{b_j} - \hat{w}$, since each removed node consumes at most $\frac{w_j}{b_j}$ of the existing charge. Consequently, the increase in killed leaf nodes if we hit is $\max(0, \frac{w_j}{b_j} - \max(0, \hat{b}\frac{w_j}{b_j} - \hat{w}))$, so the expected[19] payoff is:

$$\mathbb{E}[K_t | K_1, \dots, K_{t-1}] = \frac{\hat{b}}{\hat{w}} \max(0, \frac{w_j}{b_j} - \max(0, \hat{b}\frac{w_j}{b_j} - \hat{w})) \underset{\text{by Lemma 3.10.5}}{\leq} 1 \,.$$

If the level is 1, then let $J \subseteq [16\ell]$ give the set of probed subtree positions, and H give the set of revealed subtree positions; since there are $\leq \ell$ queries, $|J|, |H|$ are both $\leq \ell$, and the probability of a query in an unexplored region to hit is $\leq \frac{b_1}{16\ell - |J \cup H|} \leq \frac{b_1}{14\ell}$. The charging scheme does not apply at this level, so:

$$\mathbb{E}[K_t | K_1, \dots, K_{t-1}] \leq \frac{b_1}{14\ell} \cdot \prod_{j=2}^{d} b_j \,.$$

Thus in all cases, $\mathbb{E}[K_t | K_1, \dots, K_{t-1}] \leq \max(1, \prod_{j=1}^{d} b_j / 14\ell)$.

Note: the total charge deposited on mid-level nodes is $\leq \ell$. The algorithm is guaranteed to succeed if the total number of leaves killed is less than the total number of leaves; i.e, if $\sum_{i \in [\ell]} K_t + \ell \leq \prod_{j=1}^{d} b_j$. Note that by Lemma 3.10.2 and the definition of b_1, $\prod_{j=1}^{d} b_j \geq b_1 \ell / \alpha \geq 8\ell$. Consequently,

$$\ell + 7\mathbb{E}[\sum_{t=1}^{\ell} K_t] \leq 8 \max(\ell, \frac{1}{14} \prod_{j=1}^{d} b_j) \leq 8 \max(\frac{1}{8}, \frac{1}{14}) \prod_{j=1}^{d} b_j \leq \prod_{j=1}^{d} b_j \,.$$

Now let $D_t = K_t / \prod_{j=2}^{d} b_j$, so that each $D_t \in [0, 1]$. Writing events in terms of D_t lets us use Lemma 2.3.1 to bound the probability that too many leaves are killed:

$$\begin{aligned}
\Pr\left[\ell + \sum_{t \in [\ell]} K_t \geq \prod_{i=1}^{d} b_i\right] &\leq \Pr\left[\sum_{t \in [\ell]} K_t \geq 7 \max(\ell, \frac{1}{14} \prod_{j=1}^{d} b_j)\right] \\
&\leq \Pr\left[\sum_{t \in [\ell]} D_t \geq 7 \max(\ell / \prod_{j=2}^{d} b_j, \frac{b_1}{14})\right] \\
&\leq \exp(-\frac{6^2}{2+6} \max(\ell / \prod_{j=2}^{d} b_j, \frac{b_1}{14}))
\end{aligned}$$

[19] When the level is d and $b_j = w_j$, we in fact we have $K_t = 1$ always; but we do not need this stronger fact.

$$\leq \exp(-\frac{9b_1}{28}) \leq 2^{b_1 \frac{9}{28 \ln 2}} \leq 2^{\lceil 3 \log 1/\delta \rceil \frac{9}{28 \ln 2}} \leq \delta \,. \qquad \square$$

In the preceding proof, we used the following:

Lemma 3.10.5. *Let $\hat{b}, \hat{w}, b, w$ be positive, and $\hat{b} \geq 1$. Then:*

$$\frac{\hat{b}}{\hat{w}} \left(\max(0, \frac{w}{b} - \max(\hat{b}\frac{w}{b} - \hat{w}, 0)) \right) \leq 1 \,.$$

Proof of Lemma 3.10.5. If $\frac{\hat{b}}{\hat{w}} \leq \frac{b}{w}$, then:

$$\frac{\hat{b}}{\hat{w}} \left(\max(0, \frac{w}{b} - \max(\hat{b}\frac{w}{b} - \hat{w}, 0)) \right) \leq \frac{\hat{b}}{\hat{w}} \frac{w}{b} \leq 1 \,.$$

Otherwise, $\frac{\hat{b}}{\hat{w}} \geq \frac{b}{w}$, and:

$$\begin{aligned} &\frac{\hat{b}}{\hat{w}} \left(\max(0, \frac{w}{b} - \max(\hat{b}\frac{w}{b} - \hat{w}, 0)) \right) \\ &\leq \frac{\hat{b}}{\hat{w}} (\frac{w}{b} - (\hat{b}\frac{w}{b} - \hat{w})) = \frac{\hat{b}}{\hat{w}} (\frac{w}{b} - \hat{b}(\frac{w}{b} - \frac{\hat{w}}{\hat{b}})) \\ &\leq \frac{\hat{b}}{\hat{w}} (\frac{w}{b} - 1(\frac{w}{b} - \frac{\hat{w}}{\hat{b}})) = \frac{\hat{b}}{\hat{w}} \frac{\hat{w}}{\hat{b}} = 1 \,. \qquad \text{since } \hat{b} \geq 1 \text{ and } \frac{w}{b} - \frac{\hat{w}}{\hat{b}} \geq 0 \qquad \square \end{aligned}$$

If $\ell \geq 4$ and $\ell \leq n/64$, then Lemma 3.10.3 and Lemma 3.10.4 together show that Algorithm 3.10.1 has error $\leq \delta$ and space usage as bounded by Eq. 3.37. To handle the cases where $\ell < 4$ and $\ell > n/64$, one can instead use the simple deterministic algorithm for $\textsc{mif}(n, \ell)$ from Algorithm 3.1.1, using only ℓ bits of space. As this is in fact less than the space upper bound from Eq. 3.37, it follows that Eq. 3.37 gives an upper bound on the space needed for a random tape, adversarially robust $\textsc{mif}(n, \ell)$ algorithm for any setting of parameters. Formally:

Theorem 3.10.6. *There is a family of adversarially robust random tape algorithms, where for $\textsc{mif}(n, \ell)$ the corresponding algorithm has $\leq \delta$ error and uses*

$$O\left(\left\lceil \frac{(4\ell)^{\frac{2}{d-1}}}{(n/4)^{\frac{2}{d(d-1)}}} \right\rceil (\log \ell)^2 + \min(\ell, \log \tfrac{1}{\delta}) \log \ell \right)$$

bits of space, where $d = \max\left(2, \min\left(\lceil \log \ell \rceil, \left\lfloor 2\frac{\log(n/4)}{\log(16\ell)} \right\rfloor \right)\right)$. At $\delta = 1/\operatorname{poly}(n)$ this space bound is $O\left(\ell^{\log_n \ell} (\log \ell)^2 + \log \ell \log n\right)$.

Remark 3.10.7. Algorithm 3.10.1 does not use the most optimal assignment of the parameters $b_d, \ldots, b_2$; constant-factor improvements in space usage are possible if one sets $b_d, \ldots, b_2$ to be roughly in an increasing arithmetic sequence, but this would make the analysis more painful.

Remark 3.10.8. In exchange for a constant factor space increase, one can adapt Algorithm 3.10.1 to produce an increasing sequence of output values. Similar adjustments can be performed for other MIF algorithms.

3.11 Random tape lower bound, adversarial setting

In this section, we prove a space lower bound for random tape algorithms in the adversarial setting, focusing on the regime where $\delta = O(\ell/n)$. In particular, we show how to write the space lower bound for algorithms for $\textsc{mif}(n, \ell)$ as a function of a space lower bound for $\textsc{mif}(w, t)$, where, if z is the number of bits of state of an algorithm, $w = \Theta(zn/\ell)$ and $t = \Theta(n/z)$. For small enough z, $t/w \gg \ell/n$, so by repeating this reduction step a few times we can increase the ratio of the stream length to the input domain size until we can apply the $\Omega(\hat{\ell}^2/\hat{n})$ lower bound for $\textsc{mif}(\hat{n}, \hat{\ell})$ from Theorem 3.3.5. With the right number of reduction steps, one obtains the lower bound formula of Theorem 3.11.7.

One way to see the reduction step is as showing that every z-bit algorithm for $\textsc{mif}(n, \ell)$ "contains" a z-bit algorithm for $\textsc{mif}(w, t)$, which on being given t elements from a set $W \subseteq [n]$ of size w, will repeatedly produce elements in W which were not in the input. That such a set *exists* can be seen as a consequence of the lower bound for the AVOID communication problem: if a z-bit algorithm for $\textsc{mif}(n, \ell)$ receives a random sequence S of $\ell/2$ elements in $[n]$, then because there are many more subsets in $\binom{[n]}{\ell/2}$ than the algorithm has states, most of the 2^z states of the algorithm will need to work for an $\Omega(2^{-z})$ fraction of all subsets. In particular, once it has reached a given state σ, for the algorithm to have a low probability of outputting a colliding element, it must avoid most of the sets of inputs that could lead to σ. In Lemma 3.9.2, we proved by a counting argument that after the random sequence S is sent, each state σ has an associated set H_σ of possible "safe" outputs which are unlikely to collide with the inputs from S, and that $|H_\sigma|$ is typically $O(zn/\ell)$.

For an algorithm to have error probability $O(\ell/n)$, it must (most of the time) keep all of its outputs inside H_σ; in other words, the algorithm must contain a "sub-algorithm" solving $\textsc{mif}(O(zn/\ell), \ell/2)$ on the set $W = H_\sigma$, for some σ. However, even though there exists a set W on which the algorithm will concentrate its outputs, it may not be possible for an adversary to find it. In particular, for *random oracle* algorithms, there may be a different value of W for each possible setting of the random string, making W practically unguessable. However, as shown in the proof of Lemma 3.7.1, the limited number of states makes it possible to learn information about *random seed* algorithms. Here, for random tape algorithms, we can do something similar.

In our core lemma, Lemma 3.11.3, we design an adversary (Adversary 3.11.1) that can with $\Omega(1)$ probability identify a set W of size $\Theta(zn/\ell)$ for which the next $\Theta(\ell/z)$ outputs of the algorithm will be contained in W, with $\Omega(1)$ probability, no matter what inputs the adversary sends next. In other words, our adversary will identify a region where the algorithm solves $\textsc{mif}(\Theta(zn/\ell), \Theta(\ell/z))$. The general strategy is to use a win-win type iterative search. First, the adversary will send a random subset of size $\ell/2$ to the algorithm, to ensure that afterwards the outputs of the algorithm are contained in some (unknown) set H_ρ. Because the algorithm has $\leq 2^z$ states, from the adversary's perspective there are $\leq 2^z$ possible candidates for H_ρ. Then, the adversary will make $O(z)$ steps of $t = O(\ell/z)$ elements each, in which either:

1. There exists a "sub-adversary" (function to choose the next t inputs, one by one) which will probably make the algorithm output an element that rules out a constant fraction of the candidate values for H_ρ. (The details are relatively simple: output i rules out set J if $i \notin J$.)

2. No matter how the adversary picks the next t inputs, there will be a set W (roughly, an "average" of the remaining candidate sets) which will probably contain the next t outputs of the algorithm.

As the set of candidate sets can only shrink by a constant fraction $O(z)$ times, the first case can only happen $O(z)$ times, with high probability. Thus, eventually, the adversary will identify a set W. Once it has done so, it runs the optimal adversary for $\textsc{mif}(\Theta(zn/\ell), \Theta(\ell/z))$. This essentially reduces the lower bound for $\textsc{mif}(\ell, n)$ to that of $\textsc{mif}(\Theta(zn/\ell), \Theta(\ell/z))$.

Definition 3.11.1. It is helpful to distinguish between two types of failure for an algorithm for $\textsc{mif}(n, \ell)$. An algorithm $\mathcal{A}$ makes an incorrect output, or mistake, for $\textsc{mif}(n, \ell)$ if outputs an element that was already in the input stream. It also has the option of "aborting" – outputting an error value like $\perp$, which is clearly not valid.

This distinction is useful because, if we take an algorithm for $\textsc{mif}(n, \ell)$, conditioned on producing some initial transcript of outputs in response to an input sequence, we may obtain an algorithm for $\textsc{mif}(|W|, t)$ for some $t \leq \ell$ and $W \subseteq [n]$; the probability that the algorithm "aborts" (produces an output outside of W) can be much larger than the probability that the algorithm makes an incorrect output (output in W that collides with an earlier input.). In the following proofs the algorithm aborting will be bad for the adversary, and it making an incorrect output will be good.

For integers n, ℓ, z with $1 \leq \ell < n$, and $\gamma \in [0, 1]$, let $\texttt{Algs}_{n,\ell,z,\gamma}$ be the set of all z-bit *random tape* algorithms for $\textsc{mif}(n, \ell)$ which on *any* adversary abort with probability $\leq \gamma$.

$$\Delta(n, \ell, \gamma, z) := \min_{\mathcal{A} \text{ in } \texttt{Algs}_{n,\ell,z,\gamma}} \delta(\mathcal{A}, n, \ell, \gamma)\,,$$

where $\delta(\mathcal{A}, w, t)$ is the maximum, over all possible adversaries, probability that $\mathcal{A}$ makes an incorrect output. As a consequence of the definition, $\Delta(n, \ell, \gamma, z)$ is non-increasing in γ and z.

Lemma 3.11.2. *Random tape algorithms for* MISSINGITEMFINDING(n, ℓ) *which do not abort often have high error if they use too little space:*

$$\Delta(n, \ell, \gamma, z) \geq \frac{1}{4} \mathbb{1}_{z \leq \ell^2/(16n \ln 2)} \mathbb{1}_{\gamma \leq 1/2} . \tag{3.38}$$

Proof of Lemma 3.11.2. This follows from the space lower bound for random oracle algorithms in the adversarial setting. A random tape algorithm for MIF(n, ℓ) with z bits of space, worst-case probability of error δ, and worst-case probability of aborting γ can used to implement a z-bit public coin communication protocol for AVOID$(n, \lceil \ell/2 \rceil, \lfloor \ell/2 \rfloor + 1)$ with success probability $\geq 1 - \gamma - \delta$ as described in the proof of Theorem 3.3.5. By the lower bound for AVOID, Lemma 3.3.2,

$$z > \frac{\ell^2}{4n \ln 2} + \log(1 - \gamma - \delta) .$$

For $\gamma \leq 1/2$, if $\delta \leq 1/4$, then $z \geq \frac{\ell^2}{4n \ln 2} - 2$. Furthermore, by Lemma 3.1.2, $z \geq \log(\ell + 1) > 1$, so then $z \geq \max(1, \frac{\ell^2}{4n \ln 2} - 2) > \frac{\ell^2}{16n \ln 2}$, since $\max(1, x - 2) > x/4$. Taking the contrapositive, if $z \leq \frac{\ell^2}{16n \ln 2}$, then $\delta > 1/4$. This proves Eq. 3.38. □

3.11.1 The induction step

First, recall:

Lemma 3.9.2. *Let Σ be a set, n and q integers, and P a function from $\binom{[n]}{q}$ to $\triangle[\Sigma]$, where $|\Sigma| \leq 2^z$. Let F be a random function $\binom{[n]}{q} \to \Sigma$ in which for all $x \in \binom{[n]}{q}$ and $\sigma \in \Sigma$, $\Pr[F(x) = \sigma] = P(x)(\sigma)$, and let X be a uniformly random element of $\binom{[n]}{q}$, chosen independently of F. For each $\sigma \in \Sigma$, define*

$$H_\sigma = \left\{ i \in [n] : \Pr\left[i \in X \mid F(X) = \sigma \right] \leq \frac{q}{4n} \right\} .$$

For any $\alpha \in (0, 1)$,

$$\Pr\left[\left| H_{F(X)} \right| \geq \hat{w} \right] \leq \alpha \qquad \textit{where} \qquad \hat{w} := \left\lceil \frac{z + 1 + \log \frac{1}{\alpha}}{q} n \frac{2 \ln 2}{1 - \ln 2} \right\rceil .$$

Lemma 3.11.3. *Let $1 \leq \ell < n$, z be integers, and $\gamma \in [0, \frac{1}{2}]$. Let $\alpha > 0$ be a real value satisfying $z \geq 2 \log(8/\alpha)$. Define, matching definitions in Adversary 3.11.1,*

$$w = 2 \left\lfloor 32 \frac{zn}{\ell} \right\rfloor \qquad \textit{and} \qquad h_{\max} = \lceil 8z/\alpha \rceil \qquad \textit{and} \qquad t = \left\lfloor \frac{\ell}{2 h_{\max}} \right\rfloor .$$

If $t < w$, then there is a distribution $\mu \in \triangle[0,1]$ for which $\mathbb{E}_{G\sim\mu} G \le \gamma + \alpha$ and:

$$\Delta(n,\ell,\gamma,z) \ge \min\left(\frac{\ell\alpha}{32n}, \left(\frac{1}{2} - \alpha\right)\mathbb{E}_{G\sim\mu}\Delta(w,t,G,z)\right). \tag{3.39}$$

Adversary 3.11.1 Adversary for a random tape $\textsc{mif}(n,\ell)$ algorithm, with parameter $\alpha \in (0,1)$.

```
Let: $w = 2\lfloor 32\frac{zn}{\ell}\rfloor$, $h_{\max} = \lceil 8z/\alpha\rceil$, and $t = \left\lfloor \frac{\ell}{2h_{\max}} \right\rfloor$
Let $\Sigma$ be the set of states of the algorithm

ADVERSARY
 1: $v \leftarrow$ a uniformly random vector in $\textsc{sort}\binom{[n]}{\lceil \ell/2\rceil}$.
 2: send $v$ to the algorithm
 3: Let $P : \textsc{sort}\binom{[n]}{\lceil \ell/2\rceil} \to \triangle[\Sigma]$ map possible values of $v$ to the resulting distribution over states in $\Sigma$[20]
 4: Compute $H_\sigma$ for each state $\sigma$ per Lemma 3.9.2, using $P$.
 5: Let $Q_0 = \{\sigma \in \Sigma : |H_\sigma| \le \frac{1}{2}w\}$
 6: for $h$ in $1,\dots,h_{\max}$ do
 7:     Let $\mathcal{D}$ be the distribution over alg. states conditioned on the transcript so far
 8:     if $\exists$ a $(\alpha/2)$-splitting $t$-length deterministic adversary $\Upsilon$ for $Q_{h-1}$ given $\mathcal{D}$ then
 9:         run $\Upsilon$ against the algorithm, and let $y \in [n]^t$ be the output
10:         $Q_h \leftarrow \{\sigma \in Q_{h-1} : y \subseteq H_\sigma\}$    ▷ With $\ge \alpha/2$ probability, $|Q_h| \le \frac{1}{2}|Q_{h-1}|$
11:         if $Q_h = \emptyset$ then abort
12:     else
13:         $W \leftarrow \{i \in [n] : |\{\sigma \in Q_{h-1} : i \in H_\sigma\}| \ge \frac{1}{2}|Q_{h-1}|\}$.
14:         Let $W' \leftarrow W$ plus $w - |W|$ padding elements
15:         Define $t$-step sub-algorithm $\mathcal{B}$ to behave like the given algorithm, conditioned on the exact transcript made so far
16:         Let $\Xi$ be a adversary (making inputs in $W'$) maximizing the probability that $\mathcal{B}$ makes an incorrect output
17:         ▷ If $\mathcal{B}$ produces an output outside of $W'$, we interpret this as $\mathcal{B}$ having aborted, not as having made an incorrect output
18:         run adversary $\Xi$ against the algorithm
19:         return
20: abort
```

In order to explain Adversary 3.11.1, we will need the following definitions:

Definition 3.11.4. Let Q be a set of states, where each state has an associated set H_σ.

A sequence y in $[n]^t$ is said to be DIVISIVE for Q if $|\{\sigma \in Q : y \subseteq H_\sigma\}| \le \frac{1}{2}|Q|$.

Say Υ is a t-length deterministic adversary. (That is, a function which maps vectors in $[n]^\star$ of length $\le t-1$ (including the zero-length vector) to values in $[n]$.) For any state $\sigma \in \Sigma$ of the algorithm, let $G(\sigma,\Upsilon)$ be the random variable in $[n]^t$ which gives the output if we run the random

[20]For example: let A be an instance of the algorithm. Then for any vector $v \in \textsc{sort}\binom{[n]}{\lceil \ell/2\rceil}$, $P(v)(\sigma) =$ Pr[the state of A just after receiving v is σ]

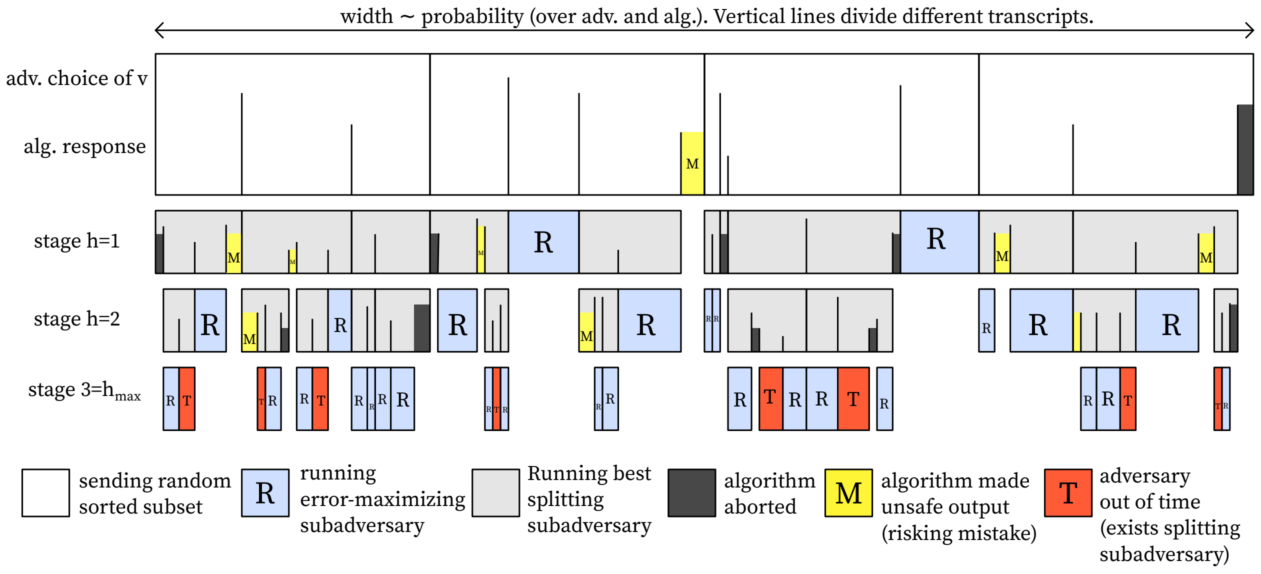

width ~ probability (over adv. and alg.). Vertical lines divide different transcripts.
adv. choice of v
alg. response
stage h=1
stage h=2
stage 3=h_max
R
M
T
sending random sorted subset
running error-maximizing subadversary
Running best splitting subadversary
algorithm aborted
algorithm made unsafe output (risking mistake)
adversary out of time (exists splitting subadversary)

Figure 3.10: Warning: This is a simplified example, and the proof of Lemma 3.11.3 includes subtleties not shown here. The diagram shows how, as time (downward direction) progresses, the algorithm-adversary pair produce different transcripts of (input, output) pairs. The y-value at which a vertical line starts indicates the time at which the transcript could diverge. The probabilities of events are proportional to the width of their rectangles. The four terminal events (marked by R, M, T, black) in this diagram roughly correspond to Line 18 of Adversary 3.11.1 being reached; the event B_{REPEAT} occurring (in which case the algorithm has *might* make a mistake); the event $B_{\text{INCOMPLETE}}$ occuring; and the event B_{ABORT} occurring, respectively. In reality, the algorithm can still abort or have B_{REPEAT} occur during the R and T events.

seed algorithm, starting at state σ, against the adversary Υ. (If after running for a few steps the algorithm has output vector $v \in [n]^\star$, its next input will be $\Upsilon(v)$.) We define an adversary to be β-SPLITTING for Q against a distribution $\mathcal{D} \in \triangle[\Sigma]$ if, when we choose a random state S from $\mathcal{D}$,

$$\Pr[G(S, \Upsilon) \text{ is divisive for } Q] \geq \beta\,.$$

Proof of Lemma 3.11.3. To prove the lower bound in Eq. 3.39, we show that when Adversary 3.11.1 is run against a z-bit random-tape algorithm $\mathcal{A}$ for $\text{MIF}(n, \ell)$ which has $\leq \gamma$ worst-case probability of aborting, the probability that $\mathcal{A}$ produces an incorrect output is at least the right hand side of 3.39. Note that the adversary described in Adversary 3.11.1 makes at most $\lceil \ell/2 \rceil + th_{\max} = \lceil \ell/2 \rceil + \lfloor \ell/(2h_{\max}) \rfloor h_{\max} \leq \lceil \ell/2 \rceil + \lfloor \ell/2 \rfloor = \ell$ inputs. A figure giving an example of how the adversary-algorithm pair could evolve is given in Figure 3.10.

When we run Adversary 3.11.1 against $\mathcal{A}$, let ρ be the state of $\mathcal{A}$ after v is sent. We define a number of events:

- B_{REPEAT} occurs if $\mathcal{A}$ produces an output in $[n] \setminus H_\rho$.

- B_{BIG} occurs if the state ρ has $H_\rho > \frac{1}{2}w$.
- $B_{\text{INCOMPLETE}}$ occurs if the *adversary* aborts without executing Line 18.
- B_{ABORT} occurs if $\mathcal{A}$ aborts *before* the adversary reaches Line 18.
- R_{ABORT} occurs if $\mathcal{A}$ aborts *while* the adversary is executing Line 18.
- R_{ERROR} occurs if $\mathcal{A}$ produces an incorrect output *while* the adversary is executing Line 18.

We individually consider some of the events listed above.

B_{REPEAT} The probability that this occurs depends on the algorithm. If B_{REPEAT} occurs, then some $i \in [n] \setminus H_\rho$ is output, and there is a $\geq \frac{\lceil \ell/2 \rceil}{4n} \geq \frac{\ell}{8n}$ probability that the set v from Line 1 contained ρ, conditioned on the algorithm reaching state ρ. Consequently, the probability that the algorithm fails because it produces an output which overlaps with the past input v is $\geq \Pr[B_{\text{REPEAT}}]\frac{\ell}{8n}$. If $\Pr[B_{\text{REPEAT}}] \geq \alpha/4$, then $\Delta(n, \ell, \gamma, z) \geq \frac{\ell\alpha}{32n}$, which implies Eq. 3.39. For the rest of this proof we will consider the case in which $\Pr[B_{\text{REPEAT}}] \leq \alpha/4$.

B_{BIG} We apply Lemma 3.9.2 to the function P, with

$$\begin{aligned}\hat{w} = \left\lceil \frac{z+1+\log(4/\alpha)}{\lceil \ell/2 \rceil} n \frac{2\ln 2}{1-\ln 2} \right\rceil &\leq 1 + \left\lfloor \frac{z+1+\log(4/\alpha)}{\lceil \ell/2 \rceil} n \frac{2\ln 2}{1-\ln 2} \right\rfloor \\ &\leq 1 + \left\lfloor \frac{8\ln 2}{1-\ln 2} \frac{z+1+\log(4/\alpha)}{\ell} n \right\rfloor \\ &\leq 1 + \left\lfloor 19 \frac{z+\log(8/\alpha)}{\ell} n \right\rfloor \leq 1 + \left\lfloor 32 \frac{zn}{\ell} \right\rfloor = \frac{1}{2}w + 1\,.\end{aligned}$$

since $z \geq 2\log(8/\alpha)$. As $\frac{1}{2}w$ is an integer,

$$\Pr\left[|H_\rho| > \frac{1}{2}w\right] = \Pr\left[|H_\rho| \geq \frac{1}{2}w + 1\right] \leq \Pr\left[|H_\rho| \geq \hat{w}\right] \leq \alpha/4\,.$$

$B_{\text{INCOMPLETE}}$ There are two ways in which the *adversary* can abort before reaching Line 18: if (Line 20) either $h_{\max}$ loop iterations are performed without failing to find a splitting adversary, or if (Line 11) at some point $Q_h = \emptyset$. If B_{REPEAT} does not hold, then all outputs will be contained by H_ρ. Furthermore, if B_{BIG} does not hold, then ρ is in Q_0; and because all outputs are contained in H_ρ, the state ρ will not be filtered out of Q_h on Line 10. Thus $\rho \in Q_h$ and the abort on Line 11 will not be used.

We now bound the probability that the algorithm will abort using Line 20. For this to happen, it must have picked $h_{\max}$ splitting adversaries, but fewer than $z+1$ of them must have produced a divisive output. (If there is a divisive output in round h, then $|Q_h| \leq \frac{1}{2}|Q_{h-1}|$; and if not, then $|Q_h| \le |Q_{h_1}|$. Thus with $z+1$ divisive outputs, $|Q_{h_{\max}}| \le |Q_0|/2^{z+1} \le |\Sigma|/2^{z+1} \le \frac{1}{2} < 1$, in which case Line 11 would have been used instead.)

For each $h \in [h_{\max}]$, let X_h be the $\{0,1\}$ indicator random variable for the event that a divisive output is found in the hth step. (If the hth step did not occur or no splitting adversary was found, set $X_h = 1$.) Since in the hth step, a splitting adversary for the distribution for the current state of the algorithm, conditioned on the transcript so far, is chosen, then $\mathbb{E}[X_h | X_1, \ldots, X_{h-1}] \ge \alpha/2$.[21] If not, then $\mathbb{E}[X_h|X_1,\ldots,X_{h-1}] = 1$. Applying Lemma 2.3.1 gives:

$$\begin{aligned}
\Pr[\sum_{h\in[h_{\max}]} X_h < z+1] &= \Pr\left[\sum_{h\in[h_{\max}]} X_h \le \left(1 - \left(1 - \frac{2z}{\alpha h_{\max}}\right)\right)\frac{\alpha h_{\max}}{2}\right] \\
&\le \exp\left(-\frac{1}{2}\left(1-\frac{2z}{\alpha h_{\max}}\right)^2 \frac{\alpha h_{\max}}{2}\right) \\
&\le \exp\left(-\frac{1}{8}\frac{\alpha h_{\max}}{2}\right) \qquad \text{since } h_{\max} = \lceil 8z/\alpha \rceil \ge 4z/\alpha \\
&= \exp\left(-\frac{\alpha}{8}\left\lceil\frac{8z}{\alpha}\right\rceil\frac{1}{2}\right) \le \exp\left(-\frac{z}{2}\right) \le \frac{\alpha}{8} \qquad \text{since } z \ge 2\log(8/\alpha)\,.
\end{aligned}$$

Thus, $\Pr[B_{\text{INCOMPLETE}} \setminus (B_{\text{REPEAT}} \cup B_{\text{BIG}})] \le \frac{\alpha}{8} \le \frac{\alpha}{4}$. (Note: if the algorithm aborts, the adversary does not, so the event B_{ABORT} is negatively correlated with $B_{\text{INCOMPLETE}}$.)

B_{ABORT} This occurs with probability $\le \gamma$.

Now, say that the adversary *does* reach a point where it will execute Line 18. First, we prove that $|W| \le w$. By Line 5, the set Q_0 only contains states $\sigma \in \Sigma$ with $|H_\sigma| \le \frac{1}{2}w$. When W is chosen on Line 13, the value of Q_{h-1} is a subset of Q_0. By the definition of W, we have:

$$\begin{aligned}
|W| &= \left|\left\{ i \in [n]\} : \frac{|\{\sigma \in Q_{h-1} : i \in H_\sigma\}|}{|Q_{h-1}|} \ge \frac{1}{2}\right\}\right| \\
&\le \sum_{i\in[n]} 2\frac{|\{\sigma \in Q_{h-1} : i \in H_\sigma\}|}{|Q_{h-1}|} \\
&= 2\frac{\sum_{\sigma\in Q_{h-1}}|H_\sigma|}{|Q_{h-1}|} \le 2\frac{1}{2}w = w\,.
\end{aligned}$$

Next, we bound the probability that the algorithm in the next t steps will produce an output outside W (or abort). The algorithm aborting is covered by event R_{ABORT}. When there is no *deterministic* splitting adversary, there also is no *randomized* splitting adversary, since each randomized adversary can be implemented by randomly picking a deterministic adversary from some distribu-

[21] While the adversary uses the state distribution conditioned on the transcript so far, this may actually be a bit more fine grained than necessary. We suspect it is enough, in this part, for the adversary to condition on $X_1, \ldots, X_{h-1}$.

tion. Now, if there is no splitting adversary, for the distribution over current states $\mathcal{D}$ as defined in Adversary 3.11.1, and for all t-step adversaries Υ, where $G(\cdot,\cdot)$ is as in Definition 3.11.4:

$$\Pr_{\hat{\sigma}\sim\mathcal{D}}[G(\hat{\sigma},\Upsilon) \text{ is divisive for } Q_{h-1}] \leq \frac{\alpha}{2}\,.$$

If a given output $y \in [n]^t$ is not divisive, then for each $i \in y$,

$$|\{\sigma \in Q_{h-1} : i \in H_\sigma\}| \geq |\{\sigma \in Q_{h-1} : y \subseteq H_\sigma\}| \geq \frac{1}{2}|Q_{h-1}|\,.$$

which implies that $i \in W$. Thus in fact $y \subseteq W$. It follows that no matter which adversary is used for the next t steps, the probability that the algorithm will produce an output outside W is $\leq \frac{\alpha}{2}$.

The chosen adversary Ξ uses t steps and provides inputs on a set of size w. Let E be the event that Line 18 is reached; so $E = \neg B_{\text{REPEAT}} \wedge \neg B_{\text{INCOMPLETE}} \wedge \neg B_{\text{BIG}} \wedge \neg B_{\text{ABORT}}$. Let G_1 be the random variable for the probability that $\mathcal{A}$ aborts, and G_2 the random variable for the probability that $\mathcal{A}$ makes an output outside of W'; these are both only nonzero when E holds. (The value of G_1 depends on the transcript of the algorithm up to Line 18, while $G_2 \leq \alpha/2$ always.) As noted above, for any transcript reaching Line 18, the probability of $\mathcal{A}$ producing an output outside W' is $\leq \frac{\alpha}{2}$. By Definition 3.11.1, the probability that $\mathcal{A}$, when run against Ξ, will make an error is at least $\Delta(w,t,G_1+G_2,z)$, because the algorithm $\mathcal{A}$, conditioned on its transcripts of inputs and outputs up to this point, will solve $\textsc{mif}(w,t)$ with a worst-case probability $\leq G_1+G_2$ of aborting.

The total probability that the algorithm aborts is:

$$\gamma \geq \Pr[B_{\text{ABORT}}] + \Pr[R_{\text{ABORT}}] = \Pr[B_{\text{ABORT}}] + \Pr[E]\mathbb{E}[G_1|E]\,.$$

Let $\hat{G}$ be a random variable distributed according to $\mu = (G_1 + G_2|E)$.

$$\begin{aligned}
\mathbb{E}[\hat{G}] = \mathbb{E}[G_1|E] + \mathbb{E}[G_2|E] &\leq \frac{\alpha}{2} + \frac{\gamma - \Pr[B_{\text{ABORT}}]}{1 - \Pr[B_{\text{ABORT}}] - \Pr[\neg E \wedge \neg B_{\text{ABORT}}]} \\
&\leq \frac{\alpha}{2} + \frac{\gamma}{1 - \Pr[\neg E \wedge \neg B_{\text{ABORT}}]} \\
&= \frac{\alpha}{2} + \frac{\gamma}{1 - \Pr[B_{\text{REPEAT}} \vee B_{\text{INCOMPLETE}} \vee B_{\text{BIG}}]} \\
&\leq \frac{\alpha}{2} + \frac{\gamma}{1 - 3\frac{\alpha}{4}} \\
&\leq \gamma + \frac{\alpha}{2} + \gamma\alpha \leq \gamma + \alpha \qquad \text{since } \gamma \leq 1/2\,.
\end{aligned}$$

The probability that the algorithm makes an error in Line 18 is:

$$\begin{aligned}
\Pr[R_{\text{ERROR}}] &\geq \Pr[E] \cdot \mathbb{E}[\Delta(w,t,G,z)|E] \\
&\geq (1 - \Pr[B_{\text{ABORT}}] - \Pr[B_{\text{REPEAT}} \vee B_{\text{INCOMPLETE}} \vee B_{\text{BIG}})\, \mathbb{E}_{\hat{G}\sim\mu}[\Delta(w,t,\hat{G},z)]
\end{aligned}$$

$$\geq \left(1 - \gamma - \frac{3\alpha}{4}\right) \mathbb{E}_{\hat{G}\sim\mu}[\Delta(w,t,\hat{G},z)]$$
$$\geq \left(\frac{1}{2} - \alpha\right) \mathbb{E}_{\hat{G}\sim\mu}[\Delta(w,t,\hat{G},z)] \,.$$

Combining this lower bound with the lower bound for the case where $\Pr[B_{\text{REPEAT}}] > \frac{\alpha}{4}$, we obtain:

$$\Delta(n,\ell,\gamma,z) \geq \min\left(\frac{\alpha\ell}{32n}, \left(\frac{1}{2} - \alpha\right)\mathbb{E}_{\hat{G}\sim\mu}[\Delta(w,t,\hat{G},z)]\right) \,. \qquad \square$$

3.11.2 Calculating the lower bound

Lemma 3.11.5. *Let $1 \leq \ell < n$. For any integer $k \geq 1$, say that z is an integer satisfying $z \leq \frac{1}{256k}\ell^{1/k}$. Then:*

$$\Delta(n,\ell,0,z) \geq \min\left(\frac{\ell}{2^{12}n}, \frac{1}{2^{k+5}}\mathbb{1}_{z\leq L}\right) \qquad \text{where} \qquad L = \frac{1}{256k}\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}} \,. \tag{3.40}$$

Consequently, algorithms for MIF with $\leq \min(\frac{\ell}{2^{12}n}, 2^{-k-5})$ error require $\geq L$ bits of space.

Proof of Lemma 3.11.5. If $k = 1$, then Eq. 3.40 follows immediately from Lemma 3.11.2, so for the rest of the proof we assume $k \geq 2$.

For $i = 1, \ldots, k-1$, define $\alpha_i = \max(\frac{1}{8\cdot 2^i}, \frac{1}{8k})$. This sequence is chosen shrink gradually and have $\alpha_1 \geq 1/16$, while $\sum_{i=1}^{k-1} \alpha_i \leq \frac{1}{4}$ and all α_i are $\geq 1/(8k)$.

Let $n_1 = n$ and $\ell_1 = \ell$, and for $i = 2, \ldots, k$, set $n_i = 2\left\lfloor 32\frac{zn_{i-1}}{\ell_{i-1}}\right\rfloor$ and $\ell_i = \left\lfloor \frac{\ell_{i-1}}{2\lceil 8z/\alpha_{i-1}\rceil}\right\rfloor$. This matches the definitions used in Lemma 3.11.3. Note that:

$$\ell_i \geq \left\lfloor \frac{\ell_{i-1}}{4\cdot 8z/\alpha_{i-1}} \right\rfloor \geq \left\lfloor \frac{\ell_{i-1}\alpha_{i-1}}{4\cdot 8z} \right\rfloor \geq \left\lfloor \frac{\ell_{i-1}}{256zk} \right\rfloor,$$

so $\ell_1 \geq \ldots \geq \ell_k$, and in particular:

$$\ell_k = \left\lfloor \cdots \left\lfloor \ell\frac{1}{256zk} \right\rfloor \cdots \frac{1}{256zk} \right\rfloor = \left\lfloor \frac{\ell}{(256zk)^{k-1}} \right\rfloor \geq 256zk \,. \qquad \text{because } z \leq \frac{1}{256k}\ell^{1/k}$$

We will later use the fact that $\ell_{k-1} \geq (256zk)^2$. By Lemma 3.1.2, we can assume $z \geq \log(\ell+1)$, as otherwise $\Delta(n,\ell,0,z) = 1$. Thus, since $k \geq 2$ and $\alpha_i \geq \frac{1}{8k}$, we have:

$$z \geq \log(\ell+1) \geq k\log(64zk) \geq k\log(64k) \geq 2\log(64k) \geq 2\log\frac{8}{\alpha_i} \,.$$

for all $i \in [k-1]$.

We will lower bound $\Delta(n,\ell,0,z)$ by recursively applying Lemma 3.11.3 $k-1$ times, and then applying Lemma 3.11.2. When Lemma 3.11.3 is used, let $\mu_{g,i}$ be the distribution maximizing the right hand side of Eq. 3.39 (with parameters n_i, ℓ_i, and abort probability g). Define random variables $G_1, \ldots, G_k$ so that $G_1 = 0$ and for $i \geq 2$,

$$G_i \sim \begin{cases} \mu_{G_{i-1},i} & \text{if } G_{i-1} \leq 1/2 \\ 1/2 & \text{otherwise} \end{cases} .$$

Then:

$$\begin{aligned}
\Delta(n, \ell, 0, z) &\geq \mathbb{1}_{G_1 \leq 1/2} \min\left(\frac{\ell_1 \alpha_1}{32 n_1}, (\frac{1}{2} - \alpha_1)\mathbb{E}\Delta(n, \ell, G_2, z)\right) \\
&\geq \mathbb{1}_{G_1 \leq 1/2} \min\Big(\frac{\ell_1 \alpha_1}{32 n_1}, \\
&\qquad (\frac{1}{2} - \alpha_1) \cdot \mathbb{E}\mathbb{1}_{G_2 \leq 1/2} \min\Big(\frac{\ell_2 \alpha_2}{32 n_2}, \\
&\qquad (\frac{1}{2} - \alpha_2) \cdot \mathbb{E}\mathbb{1}_{G_3 \leq 1/2} \min\Big(\frac{\ell_3 \alpha_3}{32 n_3}, \\
&\qquad\qquad \ldots \quad \mathbb{E}\mathbb{1}_{G_k \leq 1/2} \cdot \frac{1}{4}\mathbb{1}_{z \leq \ell_k^2/(16 n_k \ln 2)} \\
&\Big) \ldots \Big)\Big)\Big) \\
&\geq \min\Big(\Pr[G_1 \leq 1/2]\frac{\ell_1 \alpha_1}{32 n_1}, \\
&\qquad \Pr[G_2 \leq 1/2](\frac{1}{2} - \alpha_1)\frac{\ell_2 \alpha_2}{32 n_2}, \quad \ldots, \\
&\qquad \Pr[G_{k-1} \leq 1/2]\prod_{i=1}^{k-2}(\frac{1}{2} - \alpha_i)\frac{\ell_{k-1}\alpha_{k-1}}{32 n_{k-1}}, \\
&\qquad \Pr[G_k \leq 1/2]\prod_{i=1}^{k-1}(\frac{1}{2} - \alpha_i)\frac{1}{4}\mathbb{1}_{\leq \ell_k^2/(16 n_k \ln 2)}\Big) .
\end{aligned}$$

The worst-case distribution $\mu_{g,i}$ for Lemma 3.11.3 is constrained by the property that $\mathbb{E}_{G \sim \mu_{g,i}} G \leq g + \alpha_i$. Consequently, for $i \geq 2$, $\mathbb{E}[G_i] \leq \mathbb{E}[G_{i-1}] + \alpha_{i-1} \leq \ldots \leq \sum_{i=1}^{k-1} \alpha_i \leq \frac{1}{4}$. Thus by Markov's inequality, for all $i = 1, \ldots, k$, $\Pr[G_k \leq 1/2] \geq 1/2$. Using the additional fact that

$$\prod_{i=1}^{k-1}(\frac{1}{2} - \alpha_i) \geq \frac{1}{2^k}\prod_{i=1}^{k-1}(1 - 2\alpha_i) \geq \frac{1}{2^k}\exp(-4\sum_{i=1}^{k-1}\alpha_i) \geq \frac{1}{2^k}e^{-1} ,$$

the lower bound simplifies to:

$$\Delta(n, \ell, 0, z) \geq \frac{1}{2e}\min\left(2^{-1}\frac{\ell_1 \alpha_1}{32 n_1}, 2^{-2}\frac{\ell_2 \alpha_2}{32 n_2}, \ldots, 2^{-(k-1)}\frac{\ell_{k-1}\alpha_{k-1}}{32 n_{k-1}}, 2^{-k}\frac{1}{4}\mathbb{1}_{z \leq \ell_k^2/(16 n_k \ln 2)}\right) .$$

Only the first and last terms here are significant, because for $i = 2, \ldots, k$,

$$\frac{\ell_i \alpha_i}{n_i} = \left\lfloor \frac{\ell_{i-1}}{2\lceil 8z/\alpha_{i-1}\rceil} \right\rfloor \cdot \alpha_i \cdot \frac{1}{2\left\lfloor 32^{\frac{z n_{i-1}}{\ell_{i-1}}} \right\rfloor} \geq \left\lfloor \frac{\ell_{i-1}\alpha_{i-1}}{32 z} \right\rfloor \cdot \alpha_i \cdot \frac{1}{4 \cdot 32^{\frac{z n_{i-1}}{\ell_{i-1}}}}$$

$$
\begin{aligned}
&\geq \frac{\ell_{i-1}\alpha_{i-1}}{64z}\alpha_i \frac{\ell_{i-1}}{128zn_{i-1}} \geq \frac{\ell_{i-1}\alpha_i}{2^{13}z^2}\frac{\ell_{i-1}\alpha_{i-1}}{n_{i-1}} \geq \frac{\ell_{i-1}}{2^{16}z^2k}\cdot\frac{k}{k}\cdot\frac{\ell_{i-1}\alpha_{i-1}}{n_{i-1}} \\
&\geq 2\frac{\ell_{i-1}\alpha_{i-1}}{n_{i-1}},
\end{aligned}
$$

where in the last step, we used the facts that $k \geq 2$ and $\ell_{i-1} \geq \ell_{k-1} \geq 2^{16}z^2k^2$. Thus, since $\alpha_1 = 1/16$,

$$
\Delta(n,\ell,0,z) \geq \frac{1}{2e}\min\left(\frac{\ell\alpha_1}{32n}, \frac{1}{4\cdot 2^k}\mathbb{1}_{z\leq \ell_k^2/(16n_k\ln 2)}\right) \geq \min\left(\frac{\ell}{2^{12}n}, \frac{1}{2^{k+5}}\mathbb{1}_{z\leq \ell_k^2/(16n_k\ln 2)}\right).
$$

We have almost proven Eq. 3.40. It remains to relax the condition $z \leq \ell_k^2/(16n_k\ln 2)$ to $z \leq L$ for some $L \leq \ell_k^2/(16n_k\ln 2)$. As a consequence of the definitions, we have:

$$
\ell_i \geq \left\lfloor\frac{\ell_{i-1}}{256kz}\right\rfloor = \left\lfloor\frac{\left\lfloor\frac{\ell_{i-2}}{256kz}\right\rfloor}{256kz}\right\rfloor = \ldots = \left\lfloor\frac{\ell}{(256kz)^{i-1}}\right\rfloor \qquad \text{and} \qquad n_i \leq 256\frac{zn_{i-1}}{\ell_{i-1}}.
$$

If the condition does not hold, then:

$$
\begin{aligned}
z &> \frac{\ell_k^2}{16\ln 2}\frac{1}{n_k} \geq \frac{\ell_k^2}{16}\frac{\prod_{k=1}^{k-1}\ell_k}{(2^8z)^{k-1}n} \\
&\geq \frac{\ell^{k+1}}{16\cdot(2^8zk)^{2(k-1)+(k-2)+(k-3)+\ldots+1+0}(2^8z)^{k-1}n} \\
&= \frac{\ell^{k+1}}{16\cdot(2^8zk)^{(k^2+k-2)/2}(2^8z)^{k-1}n}.
\end{aligned}
$$

Rearranging to put all the z terms on the left gives:

$$
2^8z\cdot(2^8z)^{\frac{k^2+3k-4}{2}} \geq 2^8\frac{\ell^{k+1}}{16k^{\frac{k^2+k-2}{2}}n},
$$

which implies

$$
z > \frac{1}{2^8}\left(\frac{2^8\ell^{k+1}}{16n}\right)^{\frac{2}{k^2+3k-2}}\frac{1}{k^{\frac{k^2+k-2}{k^2+3k-2}}} \geq \frac{1}{2^8k}\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}} =: L.
$$

In the last inequality, we used the fact that $\frac{k^2+k-2}{k^2+3k-2} \leq 1$ for $k \geq 1$. □

Now, say a random tape algorithm has error probability $\leq \frac{1}{2^{13}}\frac{\ell}{n}$ against any adaptive adversary, and uses z bits of space. If $\ell \geq n^{8/9}$, it is easy to show that the optimal value of k with which to apply Lemma 3.11.5 is 1. Assume $\ell < n^{8/9}$. Then for all k, we have two cases. If $z \leq \frac{1}{256k}\ell^{1/k}$, then we can apply Lemma 3.11.5. By Lemma 3.1.2, $z \geq \log(\ell+1) \geq 1$, which implies $(256k)^k \leq \ell$

and thus that $k \leq \left\lfloor \frac{1}{8} \log \ell \right\rfloor$. For all $\ell < n^{8/9}$, the algorithm has error:

$$\leq \frac{\ell}{2^{13} n} < \min(\frac{\ell}{2^{12} n}, \frac{1}{2^5 \ell^{1/8}}) \leq \min(\frac{\ell}{2^{12} n}, \frac{1}{2^{k+5}}),$$

so by Lemma 3.11.5, the algorithm space must satisfy:

$$z \geq \frac{1}{256k} \left(\frac{\ell^{k+1}}{n} \right)^{\frac{2}{k^2+3k-2}} .$$

Since we either have this lower bound on z, or $z \geq \frac{1}{256k} \ell^{1/k}$, it follows that for any integer $k \geq 1$,

$$z \geq \min \left(\frac{1}{256k} \ell^{1/k}, \frac{1}{256k} \left(\frac{\ell^{k+1}}{n} \right)^{\frac{2}{k^2+3k-2}} \right) .$$

We can take the maximum over all values of k to obtain:

$$z \geq \max_{k \in \{1,\ldots,\ell\}} \frac{1}{256k} \min \left(\ell^{1/k}, \left(\frac{\ell^{k+1}}{n} \right)^{\frac{2}{k^2+3k-2}} \right) .$$

The right hand side can be simplified with the following lemma:

Lemma 3.11.6.

$$\max_{k \in \mathbb{N}} \frac{1}{k} \min \left(\ell^{1/k}, \left(\frac{\ell^{k+1}}{n} \right)^{\frac{2}{k^2+3k-2}} \right) \geq \frac{1}{k} \max_{k \in \mathbb{N}} \left(\frac{\ell^{k+1}}{n} \right)^{\frac{2}{k^2+3k-2}} \geq \frac{\log \ell}{2 \log n} \ell^{\frac{15}{32} \frac{\log \ell}{\log n}} . \tag{3.41}$$

Proof of Lemma 3.11.6. The left branch of the $\min(\cdot, \cdot)$ terms in Eq. 3.41 is actually unnecessary. For any integer $\lambda \geq 2$, say that

$$\frac{1}{\lambda} \left(\frac{\ell^{\lambda+1}}{n} \right)^{\frac{2}{\lambda^2+3\lambda-2}} = \max_{k \in \mathbb{N}} \frac{1}{k} \left(\frac{\ell^{k+1}}{n} \right)^{\frac{2}{k^2+3k-2}} .$$

Then in particular,

$$\frac{1}{\lambda} \left(\frac{\ell^{\lambda+1}}{n} \right)^{\frac{2}{\lambda^2+3\lambda-2}} \geq \frac{1}{\lambda - 1} \left(\frac{\ell^{(\lambda-1)+1}}{n} \right)^{\frac{2}{(\lambda-1)^2+3(\lambda-1)-2}} \geq \frac{1}{\lambda} \left(\frac{\ell^{\lambda}}{n} \right)^{\frac{2}{\lambda^2+\lambda-4}} ,$$

which implies

$$n^{2\lambda+2} = n^{(\lambda^2+3\lambda-2)-(\lambda^2+\lambda-4)} \geq \ell^{\lambda \cdot (\lambda^2+3\lambda-2)-(\lambda+1)\cdot(\lambda^2+\lambda-4)} = \ell^{\lambda^2+\lambda+4},$$

hence we have $\ell \leq n^{\frac{2\lambda+2}{\lambda^2+\lambda+4}}$.

On the other hand, we have

$$\frac{1}{\lambda}\ell^{1/\lambda} \geq \frac{1}{\lambda}\left(\frac{\ell^{\lambda+1}}{n}\right)^{\frac{2}{\lambda^2+3\lambda-2}} \qquad \iff \qquad n^{\lambda} \leq \ell^{(\lambda+1)\lambda-\frac{\lambda^2+3\lambda-2}{2}} = \ell^{\frac{\lambda^2-\lambda+2}{2}} ,$$

so the left branch of the $\min(\cdot,\cdot)$ in Eq. 3.41 is only smaller when $\ell \geq n^{\frac{2\lambda}{\lambda^2-\lambda+2}}$. As

$$\frac{2\lambda+2}{\lambda^2+\lambda+4} \leq \frac{2\lambda}{\lambda^2-\lambda+2} ,$$

for all $\lambda \geq 1$, it follows that the left branch of the $\min(\cdot,\cdot)$ in Eq. 3.41 is only smaller than the right when the entire term is not the maximum. Thus the left hand side of Eq. (3.41) is:

$$\geq \max_{k\in\mathbb{N}} \frac{1}{k}\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}} . \tag{3.42}$$

To get a looser but more easily comprehensible lower bound, we first observe that we can pull the $1/k$ factor out of the max, with only minor losses, because:

$$\max_{k\in\mathbb{N}} \frac{1}{k}\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}} \geq \frac{1}{k_\star}\left(\frac{\ell^{k_\star+1}}{n}\right)^{\frac{2}{k_\star^2+3k_\star-2}} \qquad \text{where} \qquad k_\star = \arg\max_{k\in\mathbb{N}}\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}} . \tag{3.43}$$

Next, we note that $\max_{k\in\mathbb{N}} \log\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}}$ is piecewise linear and convex in $\log \ell$. Consequently, we can lower bound it using the convex function $C\frac{(\log \ell)^2}{\log n}$, where C is the maximum value which satisfies the inequality at all "corner points" of $\max_{k\in\mathbb{N}} \log\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}}$. These corner points occur precisely at values of $\log \ell$ where, for some $k \geq 2$, we have:

$$\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}} = \left(\frac{\ell^{(k-1)+1}}{n}\right)^{\frac{2}{(k-1)^2+3(k-1)-2}} .$$

Rearranging this gives:

$$\frac{\log n}{\log \ell} = \frac{(k+1)((k-1)^2+3(k-1)-2)-(k-1+1)(k^2+3k-2)}{((k-1)^2+3(k-1)-2)-(k^2+3k-2)} = \frac{k^2+k+4}{2k+2} .$$

so the corners occur at $\ell = n^{\frac{2k+2}{k^2+k+4}}$; and at such ℓ, we have

$$\begin{aligned}\log\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}} &= \left(\frac{2}{k^2+3k-2}\cdot((k+1)\frac{2k+2}{k^2+k+4}-1)\right)\log n = \frac{2}{k^2+k+4}\log n \\ &= \frac{(\log \ell)^2}{\log n}\frac{2}{k^2+k+4}\left(\frac{\log n}{\log \ell}\right)^2 = \frac{(\log \ell)^2}{\log n}\frac{2}{k^2+k+4}\left(\frac{k^2+k+4}{2k+2}\right)^2\end{aligned}$$

$$= \frac{1}{2}\frac{(\log \ell)^2}{\log n}\frac{k^2+k+4}{(k+1)^2} \geq \frac{15}{32}\frac{(\log \ell)^2}{\log n}\,.$$

The function $\frac{k^2+k+4}{(k+1)^2}$ has derivative $\frac{k-7}{(k+1)^3}$ and is minimized exactly at $k = 7$, where it has value $\frac{15}{16}$. Consequently, the value $C = \frac{15}{32}$ is the best possible.

We still need to lower bound the $1/k_\star$ term of Eq. 3.43. From the calculation of the corner points, it follows that a given integer k only maximizes $\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}}$ when $\ell \in [n^{\frac{2k+4}{k^2+3k+6}}, n^{\frac{2k+2}{k^2+k+4}}]$. Therefore, the optimal value $k_\star$ must satisfy:

$$\log \ell \leq \frac{2k_\star + 2}{k_\star^2 + k_\star + 4}\log n \leq \frac{2}{k_\star}\log n\,,$$

which implies: $\frac{1}{k_\star} \geq \frac{1}{2}\frac{\log \ell}{\log n}$. With this inequality, and the value of C, we obtain:

$$x \geq \frac{1}{k_\star}\left(\frac{\ell^{k_\star+1}}{n}\right)^{\frac{2}{k_\star^2+3k_\star-2}} \geq \frac{\log \ell}{2\log n}\ell^{\frac{15}{32}\frac{\log \ell}{\log n}}\,. \qquad \square$$

Summarizing, we have the following theorem:

Theorem 3.11.7. *Random tape δ-error adversarially robust algorithms for* MIF(n, ℓ) *require*

$$\Omega\left(\max_{k\in\mathbb{N}} \frac{1}{k}\left(\frac{\ell^{k+1}}{n}\right)^{\frac{2}{k^2+3k-2}}\right) = \Omega\left(\frac{\log \ell}{\log n}\ell^{\frac{15}{32}\log_n \ell}\right)$$

bits of space, for $\delta \leq \frac{\ell}{2^{13}n}$.

Remark 3.11.8. To prove a lower bound, we required $\delta \leq \frac{\ell}{2^{13}n}$. For large values of δ, random tape algorithms can be much more efficient. For example, there is a $(\log t)$-space algorithm for MIF(n, ℓ) with $\Theta(\ell^2/t)$ error probability, which on each step randomly picks a new state (and output value) from $[t]$.

Remark 3.11.9. The lower bound of Theorem 3.11.7 is not particularly tight, and we suspect it can be improved to match the upper bound within polylog(ℓ, n) factors. It may be the case that actual algorithms must fall in one of the following two cases, or in some interpolation between them:

- If a constant fraction of the next $\ell/2$ outputs are contained in W, we could (essentially) reduce to MIF$(w, \ell/2)$.
- If on each search step of length $\Theta(\ell/z)$, the outputs of the algorithm are concentrated in a new set of size $\Theta(w/z)$, then we could (essentially) reduce to MIF$(\Theta(w/z), \Theta(\ell/z))$.

Remark 3.11.10. The adversary of Adversary 3.11.1 runs in doubly exponential time, and requires knowledge of the algorithm. The former condition cannot be improved by too much: if one-way functions exist, one could implement the random oracle algorithm for $\textsc{mif}(n, \ell)$ from Theorem 3.3.6 using a pseudo-random generator that fools all polynomial-time adversaries. One can also prove by minimax theorem that universal adversaries for random tape $\textsc{mif}(n, \ell)$ algorithms can not be used to prove any stronger lower bounds than the one for random oracle algorithms.

Remark 3.11.11. For a zero-mistake (Definition 2.2.2) algorithm with $\leq \delta$ probability of aborting or returning $\perp$, we have $0 = \Delta(n, \ell, \delta, z)$. For $\delta = O(1)$, one can obtain a lower bound on z matching Theorem 3.11.7.

A different and weaker approach is, given a zero-mistake random tape algorithm $\mathcal{A}$ with error $\leq \delta \leq \frac{1}{3}$, to construct a new algorithm $\mathcal{B}$ with error $O(\frac{\ell}{n})$ by running $\Theta(\frac{\log \frac{n}{\ell}}{\log 1/\delta})$ parallel copies of $\mathcal{A}$ and reporting outputs from any copy that has not yet aborted. Proving a space lower bound for $\mathcal{B}$ then implies a slightly weaker one for $\mathcal{A}$.

3.12 Conclusion

In this chapter, we proved upper and lower bounds for $\textsc{mif}(n, \ell)$ in a variety of models. Some of these results are closely tied to the details of MissingItemFinding, but others use more general principles.

For algorithms using $\textsc{mif}$, there appear to be two main ways in which randomness can productively be used. First, one can blindly pick a random value. A single random value will fail with $\Omega(\ell/n)$ probability in the static setting, and picking a random value for each output will typically fail with $\Omega(\min(1, \ell^2/n))$ probability in the adversarial setting. White-box-robust random tape algorithms, which essentially can only use this technique, can not do much better than deterministic algorithms when the error level is $O(\min(1, \ell^2/n))$. Second, one can pick a random subset of $[n]$, and maintain a bit to track whether the input ever overlaps with the region. This second technique is the basis for the random oracle, seed, and tape algorithms in the adversarial setting.

Because algorithms for $\textsc{mif}$ can generally be modified to be zero mistake (either give correct outputs or explicitly abort), designing zero-error algorithms is not too difficult: one need only add on a helper algorithm that is activated when the original randomized algorithm fails. For $\textsc{mif}$, this happens to use $O(\ell^2/n \operatorname{polylog}(n))$ space in expectation.

Almost all of the lower bounds in this chapter used a reduction from the two-player communication problem $\textsc{avoid}$ (or an equivalent lemma, like Lemma 3.9.2). They often used a simple iterative extraction trick: if an adversary echos back the last output of the algorithm for $\textsc{mif}$, it can force the algorithm to produce a new output and reveal more information.

The random seed (and possibly random-tape) lower bounds in the adversarial setting may use a more generally applicable strategy. Algorithms with these types of randomness are fundamentally limited by the amount of randomness they can fit into their state. For the random-seed and random-tape cases, we design adversaries that try to learn the past random decisions of the algorithm. For random seed algorithms, the adversary is rather generic, and uses a win-win strategy in which it can either learn more information or it reaches a point where on the next few inputs, the algorithm will behave pseudo-deterministically, letting one apply pseudo-deterministic lower bounds. For random tape algorithms, the adversary we design uses a specific feature of MIF: that as a consequence of the AVOID reduction, states of the algorithm can be associated with a limited set of possible future outputs. Using this, the adversary (by learning the earlier state, or at least a likely distribution over earlier states) can learn a constraint on the future behavior of the algorithm.

We note that in many aspects, the random tape lower bounds and algorithms are more complicated forms of the random seed lower bounds and algorithms. In general, when proving a random tape lower bound, it may be better to try to design a random seed lower bound first. This recommendation comes from a sample size of one; perhaps one should instead try a white-box lower bound, in order to understand what information exactly the states of the algorithm can and cannot retain about the past.

In this chapter, we defined MIF(n, ℓ) so that the input stream could contain repeated elements. Requiring that the input stream never repeats an element does not appear to change space complexity much in the regime $n = \ell^{\Theta(1)}$ that we were most interested in, but makes the proofs a bit more complicated.

We did not in general try to optimize the update time of the algorithms presented here. That being said, using a construction for random permutations described in Lemma 5.7.1, one can speed up the common task of searching a random list for a specific element.

The MissingItemFinding problem shares a few behaviors with the graph coloring task described in the next chapter. Both are fundamentally subtractive tasks: every new input strictly reduces the space of possible outputs. The bounds known for graph coloring for randomized algorithms in the static setting, random oracle algorithms in the adversarial setting, and deterministic algorithms all use very similar techniques to those of this chapter, and we expect that our white-box adversarial and pseudo-deterministic lower bounds, in particular, have easy analogues.

This chapter resolves the most pressing question about MissingItemFinding(n, ℓ), proving that the type of randomness the algorithm has access to significantly affects the space required in the adversarial setting. Only a few gaps in space complexity bounds remain – most notably, for random tape algorithms in the adversarial setting – but it is unlikely that further quantitative improvements here will teach us anything major about streaming algorithms in general.

Chapter 4

Robust and multipass deterministic streaming algorithms for graph coloring

4.1 Introduction

A graph coloring is an assignment of colors from some set to the vertices of a graph, so that no two adjacent vertices are given the same color. For a given graph G on n vertices, the minimum number of colors is denoted $\chi(G)$. Unfortunately, even computing $\chi(G)$, let alone finding a color assignment, is NP-hard, so we often study easier tasks, like finding a graph coloring where the number of colors used is a function of some easily computed graph quantity, like the maximum degree Δ of the vertices in a graph. Because the standard greedy algorithm for graph coloring uses $\leq \Delta + 1$ colors one can always obtain a "$(\Delta + 1)$-coloring".[1]

It is natural to consider the problem of graph coloring in the streaming setting. In one of the more general graph streaming models, as input one is given a graph edge insertion stream (for short: graph stream) consisting of a sequence of pairs of distinct integers in $[n]$. This sequence is interpreted as giving the edges of a graph on the vertex set $[n]$. One may forbid duplicate edges to ensure the graph is not a multigraph. A common space target for algorithms in the graph streaming setting is to use "semi-streaming" space—that is, $\widetilde{O}(n)$ bits of space, where $\widetilde{O}(\cdot)$ hides $\operatorname{polylog}(n)$ factors—and have failure probability $\leq 1/\operatorname{poly}(n)$. In the last few years, [ACK19] found a streaming algorithm which performs a $(\Delta + 1)$-coloring of an input stream encoding a graph of maximum degree Δ, using semi-streaming space. That result gives a randomized algorithm for the

[1]While a $\Delta + 1$-coloring is optimal for graphs containing cliques of size $\Delta + 1$, one can for example efficiently find Δ-colorings of Δ-colorable graphs; and even down a bit further; in particular, if Δ is *constant*, [MR14] identifies a threshold close to $\Delta - \sqrt{\Delta}$ where one can c-color c-colorable graphs in polynomial time if c is larger than the threshold, but c-coloring c-colorable graphs is NP-hard below the threshold.

static setting (see Section 2.2.2). In this chapter, we try to determine what the tradeoff between space and the number of colors used looks like for graph coloring algorithms in the *adversarial* setting. Building on parallel work by [ACS22] that proves lower bounds for 1-pass and finds multi-pass deterministic graph coloring algorithms, we also design a multi-pass deterministic algorithm which obtains a $(\Delta + 1)$-coloring in semi-streaming space.

4.1.1 Results

Our results in the adversarial setting and for multi-pass deterministic streaming algorithms are briefly compared with related work in Tables 4.1 and 4.2. In all cases we consider graph edge insertion streams. See Section 2.2 for the definitions of the models. We will give a more detailed discussion of the techniques behind these results at the start of each of the corresponding sections.

Model	**Colors**	**Space**	**Reference**
Static, random seed	$\Delta + 1$	$\widetilde{O}(n)$	[ACK19]
Static, random seed	Δ when possible	$\widetilde{O}(n)$	[AKM22]
Static, random seed	$\kappa(1 + o(1))$	$\widetilde{O}(n)$	[BCG20]
Static, random oracle	$\kappa + x$	$\Omega(n\kappa/x^2)$	[BCG20][2]
Static, random oracle	χ	$\Omega(n^2)$	[ACKP19]
Static, random seed	deg +1 list coloring	$\widetilde{O}(n)$	[HKNT22]
Adversarial	$O(\Delta)$	$\Omega(n\Delta)$	Theorem 4.3.3
Adversarial	$O(\Delta^2)$	$\Omega(n)$	Theorem 4.3.3
Adversarial, random seed	$O(\Delta^3)$	$\widetilde{O}(n)$	Theorem 4.4.1
Adversarial, random oracle	$O(\Delta^{2.5})$	$\widetilde{O}(n)$	Theorem 4.5.7
Deterministic	$\Delta + 1$	$O(n\Delta)$	greedy algorithm
Deterministic	$\leq n/2$	$\Omega(n\Delta/(\log n)^2)$	[ACS22] (with Cor. 4.7.1)

Table 4.1: A few major results for the space usage of graph coloring on single-pass edge insertion streams. In all cases, we assume the final value of the graph parameter is known in advance. Δ is the maximum degree of the graph formed by the stream, κ the maximum degeneracy of the graph formed by the stream, and χ the optimal number of colors needed, and x an integer parameter ≥ 1. Randomized algorithms must fail with (tracking) error probability $O(1/\operatorname{poly}(n))$. We assume Δ is between $\Omega((\log n)^2)$ and $n/2$.

Adversarial setting. First, we give results for graph coloring in the adversarial setting. By a reduction to (a variant of) the AVOID one-way communication task, we prove a color-space tradeoff. By "tracking error probability" we mean the probability of error over the course of the entire stream produced by the adversary; recall Definition 2.2.1.

[2][BCG20] claims an $\Omega(n^2/x^2)$ lower bound for the task of "$\kappa + x$"-coloring, but this only applies when the degeneracy $\kappa = \Omega(n)$. Adjusting the proof of [BCG20, Theorem 21] to, for a given κ, split a graph into chunks of size $5(\kappa - 2)$, similarly to the way we split the vertex set for Theorem 4.3.3, yields a more usable lower bound of $\Omega(n\kappa/x^2)$. The argument will use the easily proven communication lower bound for the direct-sum form of the INDEX communication problem.

Colors	Passes	Space	Reference
$O(\text{poly}(\Delta))$	need ≥ 2	$\widetilde{O}(n)$	[ACS22]
$O(\Delta)$	$O(\log \Delta)$	$\widetilde{O}(n)$	[ACS22]
$\Delta + 1$	1	$O(n\Delta)$	greedy algorithm
$\Delta + 1$	$O(\log \Delta \log \log \Delta)$	$\widetilde{O}(n)$	Theorem 4.6.11
$\deg + 1$	$O(\log \Delta \log \log \Delta)$	$\widetilde{O}(n)$	Theorem 4.6.15

Table 4.2: Space vs. passes vs. colors for deterministic multi-pass streams.

Theorem 4.3.3. *Let L, n, K be integers satisfying $12 \ln(4n) \leq L < K \leq \frac{n}{2}$.*

Any adversarially robust coloring algorithm $\mathcal{A}$ for graph edge insertion streams of n-vertex graphs of maximum degree $\leq L$, which maintains a coloring with $\leq K$ colors with tracking error probability $\leq \frac{3}{4}$, requires space

$$\geq \frac{1}{30} \frac{nL^2}{K} .$$

Corollary 4.1.1. *For any real value $c \geq 1$, and $\Delta \in [\Omega(\log n), O(n^{1/c})]$, a robust $O(\Delta^c)$-coloring algorithm on graph edge insertion streams which succeeds $\geq 1/4$ of the time requires $\Omega(n\Delta^{2-c})$ space. In particular, $O(\Delta)$-coloring requires $\Omega(n\Delta)$ space, $O(\Delta^2)$ coloring requires $\Omega(n)$ space, and $O(\Delta^3)$ coloring requires $\Omega(n/\Delta)$ space.*

This rules out the possibility of finding a $(\Delta + 1)$-coloring algorithm in the adversarial setting that works for all Δ and uses semi-streaming space. (We note that concurrent and independent work by [ACS22] finds an equivalent result.)

Because deterministic algorithms are also adversarially robust, Theorem 4.3.3 alone gives a tight $\Omega(n\Delta)$ lower bound for the space usage of deterministic $(\Delta + 1)$-coloring algorithms. For comparison, a slight modification of [ACS22]'s proof gives a much stronger result:

Corollary 4.7.1. *[Modification of [ACS22], Theorem 1] For any integer $L \geq 8 \log n \ln(2n)$, a deterministic algorithm which maintains an $n/2$-coloring (or better) of a graph edge insertion stream of maximum degree $\leq L$ requires space:*

$$\geq \frac{nL}{16 \ln 2 (\log n)^2} . \tag{4.16}$$

We also present two robust algorithms for graph coloring which use semi-streaming space.

Theorem 4.4.1. *Algorithm 4.4.1 is an adversarially robust $O(\Delta^3)$ coloring algorithm, which uses $\widetilde{O}(n)$ bits of space (including random bits used by the algorithm) and succeeds with high probability.*

Corollary 4.4.5. *By adjusting parameters of Algorithm 4.4.1, we obtain for any $\beta \in [0, 1]$ a robust, random-seed algorithm for $O(\Delta^{3-2\beta})$-coloring using $\widetilde{O}(n\Delta^{\beta})$ space.*

Theorem 4.5.7. *There is an $O(\Delta^{5/2})$-coloring algorithm which is robust (with total error probability $\le \delta$) against adaptive adversaries, and runs in $O(n \operatorname{polylog} n \cdot \log \frac{1}{\delta})$ bits of space, given oracle access to $O(n\Delta)$ bits of randomness.*

Corollary 4.5.8. *By adjusting parameters of Algorithm 4.5.1, we obtain for any $\beta \in [0,1]$ a robust $O(\Delta^{(5-3\beta)/2})$-coloring algorithm using $\widetilde{O}(n\Delta^{\beta})$ space, assuming oracle access to $O(n\Delta)$ bits of randomness.*

A weakness of Algorithm 4.5.1 is that it requires the algorithm be able to access all $\widetilde{O}(n\Delta)$ random bits in advance. If we assume that the adversary is limited in some fashion, then it may be possible to store $\le \widetilde{O}(n)$ true random bits, and use a pseudorandom number generator to produce the $\widetilde{O}(n\Delta)$ bits that the algorithm uses, on demand. For example, if the adversary only can use $O(n/\log n)$ bits of space, using Nisan's PRG [Nis90] on $\Omega(n)$ true random bits will fool the adversary. Alternatively, assuming one-way functions exist, there is a classic construction [HILL99] to produce a pseudorandom number generator using $O(n)$ true random bits, which in polynomial time generates poly(n) pseudorandom bits that any adversary limited to using polynomial time cannot distinguish with non-negligible probability from truly random bits.

Together, the classic streaming algorithm of [ACK19], our results on adversarially robust graph coloring, and the deterministic lower bound of [ACS22] establish a three-way separation between the number of colors achievable by algorithms using semi-streaming space. In particular, in the static setting $(\Delta + 1)$-coloring is possible in semi-streaming space; in the adversarial setting one must use between $\widetilde{\Omega}(\Delta^2)$ and $\widetilde{O}(\Delta^3)$ colors (with a tighter $\widetilde{O}(\Delta^{2.5})$ bound if one has a random oracle), and deterministic algorithms using semi-streaming space cannot achieve a coloring using $O(\Delta^c)$-colors at all, for any $c \ge 1$, if $\Delta \in [\Omega(\operatorname{polylog}(n)), O(n^{1/c})]$.

If we instead consider the space needed to achieve $\Delta + 1$ coloring, we have proven that $\Omega(n\Delta)$ space is needed in the adversarial setting for $\Delta = \Omega(\log n)$, while by [ACK19], $\widetilde{O}(n)$ space works in the static setting. This separation is quadratic when $\Delta = n/4$, say. This resolves a recent open question:[3] whether there exists a strong separation between algorithms in the static and adversarial settings for *natural* problems. [KMNS21] gave such a separation, exhibiting a function estimation problem for which the ratio between the adversarial and standard streaming complexities was as large as $\widetilde{\Omega}\left(\sqrt{\lambda_{\varepsilon,m}}\right)$, which is exponential upon setting parameters appropriately; but their function was highly artificial.

[3]This open question was explicitly raised in the STOC 2021 workshop *Robust Streaming, Sketching, and Sampling* [Ste21]. [KMNS21, Question 6.1] also asks whether there are "natural" streaming problems with strong separations between space complexities in adversarial and static settings; graph coloring, while arguably "natural", has the caveat that its outputs, like those of MissingItemFinding, are combinatorial, not numerical. We suspect that the task of estimating the number of distinct elements for a turnstile stream, which is "symmetric" and has a numerical output, has a static-vs-adversarial separation; but finding a lower bound here is still an open problem – the most recent robust algorithm is [BEEO22] and uses $\widetilde{O}(m^{1/3})$ space, where m is the stream length, and no separation from the static setting is known.

Multi-pass deterministic streaming. Here we find a multi-pass algorithm for $(\Delta+1)$-coloring that runs in semi-streaming space and a small number of rounds.

Theorem 4.6.11. *There is an efficient deterministic semi-streaming algorithm to* $(\Delta + 1)$*-color an n-vertex graph, given a stream of its edges. The algorithm uses* $O(n(\log n)^2)$ *bits of space and runs in* $O(\log \Delta \log \log \Delta)$ *passes.*

The above result shows that we can improve the $O(\Delta)$-coloring result of [ACS22] to the combinatorially optimal $(\Delta+1)$-coloring by paying only an additional $O(\log\log\Delta)$ factor in the number of passes. It is worth pointing out here that in the streaming model, as well as several other cases, 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 [BG18, ACK19], whereas the only known streaming $(\Delta + 1)$-coloring algorithm, due to [ACK19], uses sophisticated tools and a combinatorially involved analysis.[4]

Our algorithm in Theorem 4.6.11 is inspired by [ACS22]'s deterministic $O(\Delta)$-coloring algorithm, and by the distributed algorithm of [GK21] that solves $(\Delta + 1)$-coloring in the CONGEST model. (That algorithm was in turn inspired by earlier algorithms of [BKM20] and [Kuh20].) The key limitation of the streaming model is that there is not enough space for a typical vertex to "know" what exactly its neighborhood is; on the other hand, distributed computing models (including CONGEST) often have difficulty coordinating distant vertices. Fortunately, the color selection procedure used by [GK21] and preceding work does not fundamentally require precise knowledge of a vertex's neighborhood, and so requires little adaptation to work in the streaming setting.

As a by-product of the technology developed for establishing Theorem 4.6.11, we also obtain a similarly efficient algorithm for the more general problem of (degree + 1)-list-coloring. In this problem, the input 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. In a streaming setting, the input is a sequence of tokens, each either an edge of G or a pair (x, color) for some vertex x; these tokens may be interleaved arbitrarily. We obtain the following algorithmic result.

Theorem 4.6.15. *Let C be a set of colors. There is a deterministic semi-streaming algorithm for* $(\textit{degree} + 1)$*-list-coloring a graph G given a stream consisting of, in any order, the edges of G, and* (x, c) *pairs specifying that the color $c \in C$ is allowed for the vertex x. We assume no (vertex,color) pair is repeated, and that the list L_x of colors that vertex x receives has length* $\deg(x) + 1$*. The algorithm uses* $O(n(\log n)^2(\log |C|)^2)$ *bits of space and runs in* $O(\log \Delta \log\log \Delta)$ *passes.*

[4] Similar examples of this difference appear in the (randomized) LOCAL algorithms [SW10, CLP18], (deterministic) dynamic graph algorithms [BCHN18], or even provable separations for the "palette sparsification" technique [ACK19, AA20]. Yet another example is the closely related problem of O(degeneracy)-coloring versus (degeneracy + 1)-coloring studied by [BCG20] who proved that the former admits a (randomized) single-pass semi-streaming algorithm while the latter does not.

Applying Theorem 4.6.11 also gives a communication protocol for two-player graph coloring, using a small number of rounds. While it is not hard to obtain an $O(n \cdot \operatorname{polylog}(n))$ communication protocol for $(\Delta+1)$ coloring by simulating the greedy algorithm (and running binary search between Alice and Bob to find an available color for each vertex), the interesting part of Corollary 4.1.2 is that we can achieve a similar communication guarantee in a much smaller number of rounds of communication.

Corollary 4.1.2. *There is a deterministic communication protocol for finding a $(\Delta+1)$-coloring of any graph of maximum degree Δ whose edges are partitioned between two players, using $O(n(\log n)^4)$ bits of communication and $O(\log \Delta \log\log \Delta)$ rounds.*

Proof of Corollary 4.1.2. This follows by a standard conversion of a streaming algorithm to a communication protocol.[5] □

4.1.2 Related work

In this section, we describe related work *on the topic of graph coloring and graph streaming*; related work for *adversarially robust streaming* specifically is described in Section 2.2.3 and as needed in this chapter.

Graph streaming and robustness. Graph streaming has become a widely popular topic to research [McG14], especially since the advent of large and evolving networks including social media, web graphs, and transaction networks. As these large graphs are regularly mined for knowledge, and can be affected by real world news and events, such knowledge often informs their future evolution. It is thus important to consider streaming algorithms for graph problems in the adversarial setting. For practical examples where robust algorithms for graph streaming might be needed, consider a user continuously interacting with a database and choosing future queries based on past answers received; or think of an online streaming or marketing service looking at a customer's transaction history and recommending them products based on it.

However, past work on such adversarially robust streaming algorithms has focused on statistical estimation problems and on sampling problems. With the exception of [BHM+21], there has not been much study of graph theoretic problems. In many cases this is not necessary: for problems calling for Boolean answers, such as testing connectivity or bipartiteness, achieving low error in the static setting automatically implies doing so against an adaptive adversary as well,

[5]Specifically: let Alice and Bob be the two players, who receive disjoint sets of edges A and B, respectively. They will run Algorithm 4.6.1 on the stream whose first half contains the edges of A, and whose second half contains the edges of B. To do this, Alice initializes the streaming algorithm, and runs it on the first half of the stream. She then sends a message encoding the state of the algorithm to Bob, who decodes the message and runs the algorithm on the second half of the stream. Bob then sends the updated state of the streaming algorithm back to Alice. This process is repeated once for each pass of the streaming algorithm; since the algorithm uses $O(n(\log n)^2)$ bits of space, uses $O(\log \Delta \log\log \Delta) = O((\log n)^2)$ passes, the total number of bits sent by this protocol is $O(n(\log n)^4)$.

since a sequence of correct outputs from the algorithm gives away no information to the adversary. For insertion-only graph streams, a number of well-studied problems such as triangle counting, maximum matching size, and maximum subgraph density can be handled by the sketch-switching framework of [BJWY20], because the underlying functions are monotone, and their value does not change often. Thanks to efficient adversarially robust sampling [BY20, BHM$^+$21], many sampling-based graph algorithms should yield corresponding robust solutions without much overhead.

On the other hand, some problems are easily proven hard in the adversarial setting. Consider the problem of finding a spanning forest in a graph undergoing edge insertions and deletions. The celebrated Ahn–Guha–McGregor sketch [AGM12] solves this in $\widetilde{O}(n)$ space, but this sketch is not adversarially robust. In fact, any adversarially robust algorithm $\mathcal{A}$ in a dynamic graph stream which can be used to identify a single edge in the graph, must use $\Omega(n^2)$ space. Suppose we have such an algorithm $\mathcal{A}$: then we can argue that the memory state of $\mathcal{A}$ upon processing an unknown graph G must contain enough information to recover G entirely: an adversary can repeatedly ask $\mathcal{A}$ for an edge, delete it, and recurse until the evolving graph becomes empty. Thus, for basic information theoretic reasons, $\mathcal{A}$ must use $\Omega(n^2)$ bits of space. For the task of finding a spanning forest, this argument implies a quadratic gap between robust and standard streaming space complexities. Arguably, this separation is not very satisfactory, since the hardness arises from the turnstile nature of the stream, allowing the adversary to delete edges.

Graph coloring. Graph coloring is, of course, a heavily-studied problem in theoretical computer science. For this discussion, we stick to streaming algorithms for this problem, which already has a significant literature [BG18, ACKP19, ACK19, BCG20, AA20, BBMU21]. We only consider graph *vertex* coloring; graph *edge* coloring has very different behavior and will be discussed in Chapter 5.

The study of graph coloring in the classical streaming model was initiated parallelly and independently by Bera and Ghosh [BG18] and Assadi, Chen, and Khanna [ACK19]. 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 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.[6] This framework implying semi-streaming coloring algorithms was then explored by Alon and Assadi [AA20] 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.

[6]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.

Very recently, Assadi, Chen, and Sun [ACS22] 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 $\Delta \leq n^{\alpha}$ 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 [AKM22], 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 [ACKP19] 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 $3 \leq c < n$ (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 [BCG20], 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 [BBMU21] 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 $(1 \pm \varepsilon)$-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 [HKNT22] 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 [AA20] whose algorithm works only when the color-list of each vertex v is $\{1, \ldots, \deg(v)+1\}$. Note that all the works mentioned above are in the "static" streaming model and all their algorithms, except those in [ACS22], are randomized and non-robust.

4.2 Preliminaries

Notation. A graph $G = (V, E)$ typically has $n = |V|$ vertices. We may identify G with its set of edges, and write $\{u, v\} \in G$ to mean that $\{u, v\}$ is an edge in G. 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.

Colorings. A *partial coloring* of a graph $G = (V, E)$ using a palette $\mathcal{C}$ (any nonempty finite set) is a tuple (U, χ) where $U \subseteq V$ is the set of uncolored vertices and $\chi \colon V \to \mathcal{C} \cup \{\perp\}$ is a function such that $\chi(x) = \perp \Leftrightarrow x \in U$. (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 *coloring* of G is a partial coloring where $U = \emptyset$.

Coloring edge insertion streams. A graph edge insertion stream is a sequence of edges[7] $e_1, \ldots, e_m$ between pairs of vertices from some fixed set, like $[n]$. For any $i \in [m]$, the first i edges of the graph stream encode an intermediate graph G_i on vertex set $[n]$ with edge set $\{e_1, \ldots, e_i\}$. There are other models of graph streams (which handle deletions, or vertex changes) but in this chapter we use "graph stream" without qualification to refer to graph edge insertion streams.

Given a graph-theoretic parameter ψ and function f, in the $f(\psi)$-coloring (algorithmic) problem one is given a value L and a graph edge insertion stream where the final graph G containing all stream edges satisfies $\psi(G) \leq L$, and wants to maintain a coloring of the vertices using $f(L)$ colors which is compatible with the current intermediate graph. This chapter will focus mainly on the case where $\psi = \Delta$, for Δ the maximum degree of the graph formed by the input stream; we will often try to maintain an $O(\Delta^c)$-coloring for some c. (Note: we do not address the problem of maintaining a coloring using a number of colors that varies with time, depending on the current maximum degree of the intermediate graph encoded by the stream. In exchange for using a constant factor more colors, one can easily do this by running $O(\log n)$ parallel instances of fixed color streaming algorithms which handle graphs of maximum degrees $1, 2, 4, 8, \ldots$, and switching between them as necessary. Since in the adversarial setting our results give space/color tradeoffs, one can always trade back the constant factor color increase for a corresponding increase in space usage.)

For deterministic multi-pass streaming algorithms for edge insertion streams, we also consider, the *list coloring* problem, wherein each $x \in V$ has an associated list (really a set) $L_x \subseteq \mathcal{C}$ 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.

The output of an algorithm for graph coloring with K colors is a vector in $[K]^n$, specifying the color for each vertex. While naively encoding this output uses $\Theta(n \log K)$ bits of space, we note that it is sometimes possible to do better. For example, when $K = n$, there exists a graph coloring which works for *all* possible graphs, which just assigns to vertex $i \in [n]$ the color i; an algorithm that just produces this output will require 0 bits of information about the graph.

[7]We assume edges must be distinct, although the graph coloring algorithms we present also can be made to work with multigraphs, if one interprets the "maximum degree" of the graph to count the multiplicity of edges.

4.3 Hardness of adversarially robust graph coloring

As might be expected, our lower bounds are best formalized through communication complexity. Recall that a typical communication-to-streaming reduction for proving a one-pass streaming space lower bound works as follows. We set up a communication game for Alice and Bob to solve, using one message from Alice to Bob. Suppose that Alice and Bob have inputs x and y in this game. The players simulate a purported efficient streaming algorithm $\mathcal{A}$ (for P, the problem of interest) by having Alice feed some tokens into $\mathcal{A}$ based on x, communicating the resulting memory state of $\mathcal{A}$ to Bob, having Bob continue feeding tokens into $\mathcal{A}$ based on y, and finally querying $\mathcal{A}$ for an answer to P, based on which Bob can give a good output in the communication game. When this works, it follows that the space used by $\mathcal{A}$ must be at least the one-way randomized communication complexity of the game. Note, however, that this style of argument where it is possible to solve the game by querying the algorithm only once, is also applicable to the static setting. Therefore, it cannot prove a lower bound any higher than the standard streaming complexity of P.

The way to obtain stronger lower bounds by using the purported adversarial robustness of $\mathcal{A}$ is to design communication protocols where Bob, after receiving Alice's message, proceeds to query $\mathcal{A}$ repeatedly, feeding tokens into $\mathcal{A}$ based on answers to such queries. In fact, in the communication games we shall use for our reductions, Bob will not have any input at all and the goal of the game will be for Bob to recover information about Alice's input, perhaps indirectly. For our main lower bound (Theorem 4.3.3), we use a communication game that is also at the core of some of the lower bounds for MissingItemFinding (see Chapter 3).

The Subset-Avoidance Problem. Recall the SUBSET-AVOIDANCE problem introduced in Section 3.3.1 and denote it AVOID(t, a, b). To restate Definition 3.3.1, AVOID(t, a, b) denotes the following one-way Alice-to-Bob communication game.

- Alice is given $S \subseteq [t]$ with $|S| = a$;
- Bob must produce $T \subseteq [t]$ with $|T| = b$ for which T is disjoint from S.

The one-way communication complexity of this game can be lower bounded from first principles. Since each output of Bob is compatible with only $\binom{t-b}{a}$ possible input sets of Alice, she cannot send the same message on more than that many inputs. Therefore, she must be able to send roughly $\binom{t}{a}/\binom{t-b}{a}$ distinct messages for a protocol to succeed with high probability. The number of bits she must communicate in the worst case is roughly the logarithm of this ratio, which we show is $\Omega(ab/t)$. This lower bound is tight and (up to an additive $O(\log n)$ term) is matched by a deterministic protocol, as shown in Lemma 3.3.4.

In this section, we shall need to consider a direct sum version of this problem that we call AVOID$^k(t, a, b)$, where Alice has a list of k subsets and Bob must produce his own list of subsets, with his ith avoiding the ith subset of Alice. We extend our lower bound argument to show that the one-way complexity of $\textsc{avoid}^k(t, a, b)$ is $\Omega(kab/t)$.

Using Graph Coloring to Solve Subset-Avoidance. To explain how we reduce the $\textsc{avoid}^k$ problem to graph coloring, we focus on a special case of Theorem 4.3.3 first. Suppose we have an adversarially robust $(\Delta + 1)$-coloring streaming algorithm $\mathcal{A}$. We describe a protocol for solving $\textsc{avoid}(t, a, b)$. Let us set $t = \binom{n}{2}$ to have the universe correspond to all possible edges of an n-vertex graph. Suppose Alice's set A has size $a = n^2/8$. We show that, given a set of n vertices, Alice can use public randomness to randomly map her elements to the set of vertex-pairs so that the corresponding edges induce a graph G that, w.h.p., has max-degree $\Delta \approx n/4$. Alice proceeds to feed the edges of G into $\mathcal{A}$ and then sends Bob the state of $\mathcal{A}$.

Bob now queries $\mathcal{A}$ to obtain a $(\Delta+1)$-coloring of G. Then, he pairs up like-colored vertices to obtain a maximal pairing. Observe that he can pair up all but at most one vertex from each color class. Thus, he obtains at least $(n - \Delta - 1)/2$ such pairs. Since each pair is monochromatic, the two vertices don't share an edge, and hence, Bob has retrieved $(n - \Delta - 1)/2$ missing edges that correspond to elements absent in Alice's set. Since Alice used public randomness for the mapping, Bob knows exactly which elements these are. He now forms a matching with these pairs and inserts the edges to $\mathcal{A}$. Once again, he queries $\mathcal{A}$ to find a coloring of the modified graph. Observe that the matching can increase the max-degree of the original graph by at most 1. Therefore, this new coloring uses at most $\Delta + 2$ colors. Thus, Bob would retrieve at least $(n - \Delta - 2)/2$ new missing edges. He again adds to the graph the matching formed by those edges and queries $\mathcal{A}$. It is crucial to note here that he can repeatedly do this and expect $\mathcal{A}$ to output a correct coloring because of its adversarial robustness. Bob stops once the max-degree reaches $n - 1$, since now the algorithm can color each vertex with a distinct color, preventing him from finding a missing edge.

Summing up the sizes of all the matchings added by Bob, we see that he has found $\Theta((n-\Delta)^2)$ elements missing from Alice's set. Since $\Delta \approx n/4$, this is $\Theta(n^2)$. Thus, Alice and Bob have solved the $\textsc{avoid}(t, a, b)$ problem where $t = \binom{n}{2}$ and $a, b = \Theta(n^2)$. As outlined above, this requires $\Omega(ab/t) = \Omega(n^2)$ communication. Hence, $\mathcal{A}$ must use at least $\Omega(n^2) = \Omega(n\Delta)$ space.

With some further work, we can generalize the above argument to work for any value of Δ with $1 \leq \Delta \leq n/2$. For this generalization, we use the communication complexity of $\textsc{avoid}^k(t, a, b)$ for suitable parameter settings. With more rigorous analysis, we can further generalize the result to apply not only to $(\Delta+1)$-coloring algorithms but to any $f(\Delta)$-coloring algorithm. That is, we can prove Theorem 4.3.3.

4.3.1 The k-fold subset avoidance problem

Let $\textsc{avoid}^k(t, a, b)$ be the problem of simultaneously solving k instances of $\textsc{avoid}(t, a, b)$.

Lemma 4.3.1. *The public-coin δ-error communication complexity of* $\textsc{avoid}^k(t,a,b)$ *is bounded thus:*

$$\mathrm{R}^{\rightarrow}_{\delta}(\textsc{avoid}^k(t,a,b)) \geq \log(1-\delta) + k\log\left(\binom{t}{a}\Big/\binom{t-b}{a}\right) \tag{4.1}$$

$$\geq \log(1-\delta) + kab/(t\ln 2)\,. \tag{4.2}$$

Proof of Lemma 4.3.1. The proof of this lemma is almost the same as the lower bound proof for $\textsc{avoid}(t,a,b)$ from Lemma 3.3.2. As before, let Π be a δ-error one-way communication protocol for $\textsc{avoid}^k(t,a,b)$ and let $d = \mathrm{cost}(\Pi)$. Since, for each input $(S_1,\ldots,S_k) \in \binom{[t]}{a}^k$, the error probability of Π on that input is at most δ, there must exist a fixing of the randomness of Π so that the resulting deterministic protocol Π' is correct on all inputs in a set

$$\mathcal{C} \subseteq \binom{[t]}{a}^k, \quad \text{with } |\mathcal{C}| \geq (1-\delta)\binom{t}{a}^k .$$

The protocol Π' is equivalent to a function $\phi\colon \mathcal{C} \to \binom{[t]}{b}^k$ where

- the range size $|\mathrm{image}(\phi)| \leq 2^d$, because $\mathrm{cost}(\Pi) \leq d$, and
- for each $(S_1,\ldots,S_k) \in \mathcal{C}$, the tuple $(T_1,\ldots,T_k) := \phi((S_1,\ldots,S_k))$ is a correct output for Bob, i.e., $S_i \cap T_i = \varnothing$ for each i.

For any fixed $(T_1,\ldots,T_k) \in \binom{[t]}{b}^k$, the set of all $(S_1,\ldots,S_k) \in \binom{[t]}{a}^k$ for which each coordinate S_i is disjoint from the corresponding T_i is precisely the set $\binom{[t]\setminus T_1}{a} \times \cdots \times \binom{[t]\setminus T_k}{a}$. The cardinality of this set is exactly $\binom{t-b}{a}^k$. Thus, for any subset $\mathcal{D}$ of $\binom{[t]}{b}^k$, it holds that $\left|\mathcal{C} \cap \phi^{-1}(\mathcal{D})\right| \leq \binom{t-b}{a}^k |\mathcal{D}|$. Consequently,

$$(1-\delta)\binom{t}{a}^k \leq |\mathcal{C}| = |\phi^{-1}(\mathrm{image}(\phi))| \leq \binom{t-b}{a}^k |\mathrm{image}(\phi)| \leq \binom{t-b}{a}^k 2^d ,$$

which, on solving to lower bound d, gives Eq. 4.1.

The weakened lower bound Eq. 4.2 follows immediately from Eq. 3.6 in the proof of Lemma 3.3.2. □

4.3.2 Reducing k-AVOID to graph coloring

Having introduced and analyzed the k-$\textsc{avoid}$ communication game, we are almost ready to prove our main lower bound result, on the hardness of adversarially robust graph coloring. We first establish the following basic lemma on the maximum degree of a random graph.

Lemma 4.3.2. *Let G be a graph with M edges and n vertices, drawn uniformly at random. Define Δ_G to be its maximum degree. Then for $0 \le \varepsilon \le 1$:*

$$\Pr\left[\Delta_G \ge \frac{2M}{n}(1+\varepsilon)\right] \le 2n \exp\left(-\frac{\varepsilon^2}{3} \cdot \frac{2M}{n}\right). \tag{4.3}$$

Proof of Lemma 4.3.2. Let $G(n,m)$ be the uniform distribution over graphs with m edges and n vertices. Observe the monotonicity property that for all $m \in \mathbb{N}$, $\Pr_{G\sim G(n,m)}[\Delta_G \ge C] \le \Pr_{G\sim G(n,m+1)}[\Delta_G \ge C]$. Next, let $H(n,p)$ be the distribution over graphs on n vertices in which each edge is included with probability p, independently of any others, and let $e(G)$ be the number of edges of a given graph G. Then with $p = M/\binom{n}{2}$,

$$\begin{aligned}
\Pr_{G\sim G(n,M)}[\Delta_G \ge C] &= \Pr_{G\sim H(n,p)}[\Delta_G \ge C \mid e(G) = M] \\
&\le \Pr_{G\sim H(n,p)}[\Delta_G \ge C \mid e(G) \ge M] && \triangleleft \text{ by monotonicity} \\
&\le \frac{\Pr_{G\sim H(n,p)}[\Delta_G \ge C]}{\Pr_{G\sim H(n,p)}[e(G) \ge M]} \le 2 \Pr_{G\sim H(n,p)}[\Delta_G \ge C].
\end{aligned}$$

The last step follows from the well-known fact that the median of a binomial distribution equals its expectation when the latter is integral; hence $\Pr_{G\sim H(n,p)}[e(G) \ge M] \ge 1/2$.

Taking $C = (2M/n)(1+\varepsilon)$ and using a union bound and Chernoff's inequality,

$$\begin{aligned}
\Pr_{G\sim H(n,p)}\left[\Delta_G \ge \frac{2M}{n}(1+\varepsilon)\right] &\le \sum_{x\in V(G)} \Pr_{G\sim H(n,p)}\left[\deg_G(x) \ge \frac{2M}{n}(1+\varepsilon)\right] \\
&\le n \exp\left(-\frac{\varepsilon^2}{3} \cdot \frac{2M}{n}\right).
\end{aligned}$$

□

Theorem 4.3.3. *Let L, n, K be integers satisfying $12\ln(4n) \le L < K \le \frac{n}{2}$.*

Any adversarially robust coloring algorithm $\mathcal{A}$ for graph edge insertion streams of n-vertex graphs of maximum degree $\le L$, which maintains a coloring with $\le K$ colors with tracking error probability $\le \frac{3}{4}$, requires space

$$\ge \frac{1}{30}\frac{nL^2}{K}.$$

Proof of Theorem 4.3.3. Given an algorithm $\mathcal{A}$ as specified, we can construct a public-coin protocol to solve the communication problem $\textsc{avoid}^{\lfloor n/(2K)\rfloor}(\binom{2K}{2}, \lfloor LK/4\rfloor, \lfloor L/2\rfloor\lceil K/2\rceil)$ using exactly as much communication as $\mathcal{A}$ requires storage space. The protocol for the more basic problem $\textsc{avoid}(\binom{2K}{2}, \lfloor LK/4\rfloor, \lfloor L/2\rfloor\lceil K/2\rceil)$ is described in Protocol 4.3.1.

To use $\mathcal{A}$ to solve $s := \lfloor n/2K\rfloor$ instances of $\textsc{avoid}$, we pick s disjoint subsets $V_1, \ldots, V_s$ of the vertex set $[n]$, each of size $2K$. A streaming coloring algorithm on the vertex set $[2K]$ with degree

Protocol 4.3.1 Protocol for $\textsc{avoid}(\binom{2K}{2}, \lfloor LK/4 \rfloor, \lfloor L/2 \rfloor \lceil K/2 \rceil)$

```
    Require: Algorithm $\mathcal{A}$ that colors graphs up to maximum degree $L$, always using $\leq K$ colors
1:  $R \leftarrow$ publicly random bits to be used by $\mathcal{A}$
2:  $\pi \leftarrow$ publicly random permutation of $\{1, \ldots, \binom{2K}{2}\}$, drawn uniformly
3:  $e_1, \ldots, e_{\binom{2K}{2}} \leftarrow$ an enumeration of the edges of the complete graph on $2K$ vertices

4:  function ALICE(S):
5:      $Z \leftarrow \mathcal{A}$::INIT($R$), the initial state of $\mathcal{A}$
6:      for $i$ from 1 to $\binom{2K}{2}$ do
7:          if $\pi_i \in S$ then
8:              $Z \leftarrow \mathcal{A}$::INSERT(Z, $R$, $e_i$)
9:      return $Z$

10: function BOB($Z$):
11:     $J \leftarrow$ empty list
12:     for $i$ from 1 to $\lfloor L/2 \rfloor$ do
13:         CLR $\leftarrow \mathcal{A}$::QUERY($Z$, $R$)
14:         $M \leftarrow$ maximal pairing of like-colored vertices, according to CLR
15:         for each pair $\{u, v\} \in M$ do
16:             $Z \leftarrow \mathcal{A}$::INSERT($Z$, $R$, $\{u, v\}$)        ▷ $M$ is turned into a matching and inserted
17:         $J \leftarrow J \cup M$
18:     if length($J$) $\leq \lfloor L/2 \rfloor \lceil K/2 \rceil$ then
19:         return fail
20:     else
21:         $T \leftarrow \{\pi_i : e_i \in$ first $\lfloor L/2 \rfloor \lceil K/2 \rceil$ edges of $J\}$
22:         return $T$
```

limit L and using at most K colors can be implemented by relabeling the vertices in $[2K]$ to the vertices in some set V_i and using $\mathcal{A}$. This can be done s times in parallel, as the sets $(V_i)_{i=1}^{s}$ are disjoint. Note that a coloring of the entire graph on vertex set $[n]$ using $\leq K$ colors is also a K-coloring of the s subgraphs supported on $V_1, \ldots, V_s$. To minimize the number of color queries made, Protocol 4.3.1 can be implemented by alternating between adding elements from the matching M in each instance (for Line 16), and making single color queries to the n-vertex graph (for Line 13).

The guarantee that $\mathcal{A}$ uses fewer than K colors depends on the input graph stream having maximum degree at most L. In Bob's part of the protocol, adding a matching to the graph only increases the maximum degree of the graph represented by Z by at most one; since he does this $\lfloor L/2 \rfloor$ times, in order for the maximum degree of the graph represented by Z to remain at most L, we would like the random graph Alice inserts into the algorithm to have maximum degree $\leq L/2 \leq L - \lfloor L/2 \rfloor$. By Lemma 4.3.2, the probability that, given some i, this random graph on V_i

has maximum degree $\Delta_i \geq L/2$ is

$$\Pr\left[\Delta_i \geq \frac{L}{4}(1+1)\right] \leq 4Ke^{-L/12}\,.$$

Taking a union bound over all s graphs, we find that

$$\Pr\left[\max_{i\in[s]} \Delta_i \geq L/2\right] \leq 4K\left\lfloor \frac{n}{2K} \right\rfloor e^{-L/12} \leq 2ne^{-L/12}\,.$$

We can ensure that this happens with probability at most $1/2$ by requiring $L \geq 12\ln(4n)$.

If all the random graphs produced by Alice have maximum degree $\leq L/2$, and the $\lfloor L/2 \rfloor$ colorings requested by the protocol are all correct, then we will show that Bob's part of the protocol recovers at least $\lfloor L/2\rfloor\lceil K/2\rceil$ edges for each instance. Since the algorithm $\mathcal{A}$'s random bits R and permutation random bits π are independent, the probability that the maximum degree is low and the algorithm gives correct colorings on graphs of maximum degree at most L is $\geq (1/2)\cdot(1/4) = 1/8$.

The edges that Bob inserts (Line 16) are fixed functions of the query output of $\mathcal{A}$ on its state Z and random bits R. None of the edges can already have been inserted by Alice or Bob, since each edge connects two vertices which have the same color. Because these edges only depend on the query output of $\mathcal{A}$, conditioned on this query output they are independent of Z and R. This ensures that $\mathcal{A}$'s correctness guarantee against an adversary applies here, and thus the colorings reported on Line 13 are correct.

Assuming all queries succeed, and the initial graph that Alice added has maximum degree $\leq L/2$, for each $i \in [\lfloor L/2\rfloor]$, the coloring produced will have at most K colors. Let B be the set of vertices covered by the matching M, so that $[2K]\setminus B$ are the unmatched vertices. Since no pair of unmatched vertices can have the same color, $|[2K]\setminus B| \leq K$. This implies $|B| \geq K$, and since $|M| = |B|/2$ is an integer, we have $|M| \geq \lceil K/2\rceil$. Thus each **for** loop iteration will add at least $\lceil K/2\rceil$ new edges to J. The final value of the list J will contain at least $\lfloor L/2\rfloor\lceil K/2\rceil$ edges that were not added by Alice; Line 21 converts the first $\lfloor L/2\rfloor\lceil K/2\rceil$ of these to elements of $\{1,\ldots,\binom{2K}{2}\}$ not in the set S given to Alice.

Finally, by applying Lemma 4.3.1, we find that the communication C needed to solve s independent copies of $\textsc{avoid}(\binom{2K}{2}, \lfloor LK/4\rfloor, \lfloor L/2\rfloor\lceil K/2\rceil)$ with failure probability $\leq 7/8$ satisfies

$$C \geq \log\left(1-\frac{7}{8}\right) + \left\lfloor \frac{n}{2K}\right\rfloor \frac{\lfloor LK/4\rfloor \cdot \lfloor L/2\rfloor\lceil K/2\rceil}{\binom{2K}{2}\ln 2}\,.$$

Because $K > L \geq 12\ln(4n) \geq 12\ln 4$, $\lfloor LK/4 \rfloor \lfloor L/2 \rfloor \lceil K/2 \rceil \geq (LK)^2/20$, so this is:

$$\geq \frac{n}{4K} \frac{L^2K^2/20}{\frac{1}{2}(2K)^2 \ln 2} - 3 = \frac{nL^2}{40K \ln 2} - 3 \geq \frac{nL^2}{30K},$$

where the last step follows because $\frac{nL^2}{K} \geq 2L^2 \geq 2(12\ln 4)^2$, so the -3 term can be dropped in exchange for a slight increase in the denominator.

□

Remark 4.3.4. The proof of Theorem 4.3.3 used the fact that an algorithm is adversarially robust to *iteratively* extract $\lfloor L/2 \rfloor \lceil K/2 \rceil$ edges which were not in the graph, for each group of $2K$ vertices. In the static setting, the best we can do (without requiring very low error probability of the algorithm) is extract just *one* set of $\lceil K/2 \rceil$ edges, for each group of $2K$ vertices. This would give us an $\Omega(\frac{n}{2K}\frac{LK/4 \cdot K/2}{2K^2}) = \Omega(nL/K)$ lower bound on the space usage for any constant-error streaming algorithm for graph coloring in the static setting. (Assuming, of course, that $n/2 \geq K > L = \Omega(\log n)$.)

In particular, this lower bound implies that $\Omega(n)$ space is required for $(\Delta+1)$-coloring, matching (up to polylog(n) factors) the upper bound of [ACK19]. While [ACK19]'s algorithm in particular does not let us reduce space in exchange for increasing the number of colors, we note that a variation of [BG18]'s algorithm does. For any integers $K \geq 2L = \Omega(\log n)$, consider the streaming algorithm which uses an $O(\log n)$-wise independent hash family to partition the set $[n]$ of vertices into $O(\frac{K}{\log n})$ parts; and which records only the edges for which both endpoints lie in the same parts. The algorithm will be able to color the graphs formed by the edges inside each part using (with high probability) $O(\log n)$ colors; it does this for each part, using a fresh set of colors each time. The total number of colors used will be $O(\frac{K}{\log n}\log n) = K$, and the algorithm will use only $O(\frac{nL}{K}(\log n)^2)$ space. This matches, up to polylog(n) factors, the lower bound for the static setting.

4.4 A robust random-seed algorithm

In this section, we describe a simple algorithm for adversarially robust $O(\Delta^3)$ coloring, using semi-streaming space, and only random-seed access to randomness. A more complicated algorithm using $O(\Delta^{2.5})$ colors, but requiring access to a random oracle, will be given in Section 4.5.

The $O(\Delta^3)$ coloring algorithm in this section uses a form of the sketch-switching framework of [BJWY20]. In that framework, a robust algorithm runs multiple parallel copies of a classic streaming algorithm, and at any given time presents outputs from just one of the instances. When the current instance can no longer provide the required performance guarantee,[8] the robust algorithm

[8] In one example of [BJWY20], the objective is to approximate a real valued function, and each instance is switched out as soon as its output value no longer is within the approximation distance of the first output value it made that

discards the current instance and switches to presenting outputs from a different instance of the classic algorithm. We will do almost the same thing: for Algorithm 4.4.1, we maintain $O(\Delta)$ copies of an specific $O(\Delta^3)$ coloring algorithm. Before these copies are used to output colorings, they use $\widetilde{O}(n/\Delta)$ space each; the copy which is currently being used to output a coloring will use $\widetilde{O}(n)$ space. The nested algorithm uses the similar design to [BCG20]: it partitions the set of vertices into $O(\Delta^2)$ parts, and uses a fresh palette of $O(\Delta)$-colors to color each part. As long as the input to the nested algorithm is not adaptively generated, only an $O(1/\Delta^2)$ fraction of the edges in the input stream will have both endpoints in the same part, and might influence the coloring. All other edges can be discarded. Of course, when the input to this nested algorithm is adaptively generated, almost *every* input edge may have both endpoints in the same part, so once a copy of the nested algorithm is revealed, it can only process $\widetilde{O}(n)$ adaptively generated edges before it risks running out of space.

Theorem 4.4.1. *Algorithm 4.4.1 is an adversarially robust $O(\Delta^3)$ coloring algorithm, which uses $\widetilde{O}(n)$ bits of space (including random bits used by the algorithm) and succeeds with high probability.*

Proof of Theorem 4.4.1. Let $\ell = \Delta$ be the driving parameter for Algorithm 4.4.1. Since `curr` only increases every $n\Delta/\ell$ inputs, and the stream length is $\le n\Delta/2$, we will always have `curr` $\le \ell$, so the algorithm will only ever try to access $h_{i,j}$ or $D_{i,j}$ for $i \in [\ell]$. The only step of Algorithm 4.4.1 that an adversary could make fail is Line 16.

By Lemma 4.4.2, this happens with $1/\operatorname{poly}(n)$ probability. Assuming Line 16 does not fail, Lemma 4.4.3 proves that the output of the algorithm is a valid $(\Delta+1)\ell^2 = (\Delta+1)\Delta^2$ coloring. Finally, Lemma 4.4.4 verifies that Algorithm 4.4.1 uses at most $\widetilde{O}(n)$ bits of space, in total. □

Lemma 4.4.2. *Line 16 of Algorithm 4.4.1 will execute successfully, with high probability, on input streams provided by an adaptive adversary.*

Proof of Lemma 4.4.2. We first remark that the time range in which Algorithm 4.4.1 updates a given set $D_{i,j}$ is disjoint from and happens before Algorithm 4.4.1 first uses the set $D_{i,j}$. The set $D_{i,j}$ is only updated when `curr` $< i$; and only used in the query routine when `curr` $= i$. Consequently, looking at the outputs of the algorithm does not help an adversary ensure any property of $D_{i,j}$. It suffices, then, to prove that for a given i, that Line 16 succeeds with high probability on any fixed input stream.

Let G be the graph encoded by the first $\frac{n\Delta}{\ell}(i-1)$ edges of the input stream. Consider any $j \in [p]$. Because $h_{i,j}$ is drawn from a 2-universal family, for any $\{u,v\} \in G$ we have $\Pr[h_{i,j}(u) =$

was revealed. As graph coloring is a combinatorial, not an approximation problem, and a single input edge can change the coloring, we rely more on designing algorithms for the instances that will work for a limited time after being revealed.

Algorithm 4.4.1 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(parameter $\ell$):                         ▷ For $O(\Delta^3)$-coloring, set $\ell = \Delta$
    Define $p := \lceil 10 \log n \rceil$
 1: ▷ One such family, by [CW79], is $H_1 = \{(x \mapsto (ax + b) \bmod p \bmod b) : a \in \mathbb{Z}_p \setminus \{0\}, b \in \mathbb{Z}_p\}$,
    where $p$ is a prime $\geq n$
    Let $\mathcal{U}$ be a 2-universal family of hash functions from $V$ to $[\ell^2]$, of size $O(\text{poly}(n))$
 2: for $i \in [\ell], j \in [p]$ do
 3:     $h_{i,j}$ be a uniformly random function from $\mathcal{U}$ mapping $V$ to $[\ell^2]$
 4:     $D_{i,j} \leftarrow \emptyset$        ▷ Either a set of $h_{i,j}$-monochromatic edges, or $\perp$ after invalidation
 5: $B \leftarrow \emptyset$                                       ▷ buffer of edges from this epoch
 6: curr $\leftarrow 1$                                           ▷ current epoch number

    Process(edge $\{u, v\}$):
 7: if $|B| \geq n\Delta/\ell$ then
 8:     $B \leftarrow \emptyset$; curr $\leftarrow$ curr $+ 1$              ▷ End current epoch, switch to next
 9: $B \leftarrow B \cup \{\{u, v\}\}$;                               ▷ Update current buffer
10: for $i$ from curr $+ 1$ to $\ell$, and $j \in [p]$ do
11:     if $h_{i,j}(u) = h_{i,j}(v)$ then                      ▷ For $h_{i,j}$-monochromatic edges...
12:         if $D_{i,j} \neq \perp \wedge |D_{i,j}| < \frac{n\Delta}{\ell^2}$ then
13:             $D_{i,j} \leftarrow D_{i,j} \cup \{\{u, v\}\}$             ▷ Record edge in $D_{i,j}$ if there is space
14:         else
15:             $D_{i,j} \leftarrow \perp$                          ▷ Wipe buffer $D_{i,j}$ if it gets too large

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

$h_{i,j}(v)] \leq 1/\ell^2$, so

$$\mathbb{E}|D_{i,j}| = \sum_{\{u,v\} \in G} \Pr[h_{i,j}(u) = h_{i,j}(v)] \leq \frac{1}{\ell^2}|G| \leq \frac{n\Delta}{2\ell^2}\,.$$

By Markov's inequality,

$$\Pr\left[|D_{i,j}| \geq \frac{n\Delta}{\ell^2}\right] \leq \frac{\mathbb{E}|D_{i,j}|}{n\Delta/\ell^2} \leq \frac{1}{2}\,.$$

Since the $h_{i,j}$ are chosen independently, the events $\{|D_{i,j}| \geq \frac{n\Delta}{\ell^2}\}_{j \in [p]}$ are independent, and the probability that all of them hold is $\leq (1/2)^p \leq 1/n^{10}$; thus Line 16 succeeds with high probability.

□

Lemma 4.4.3. *If Line 16 does not fail, then Algorithm 4.4.1 outputs a valid $(\Delta+1)(\Delta^2)$ coloring of the input graph.*

Proof of Lemma 4.4.3. We need to prove that after receiving each edge $\{u, v\}$ in the graph, the algorithm assigns different values to u and to v. Let k be the value of the variable k chosen at Line 16, and let c be the current value of `curr`. Since $D_{\texttt{curr},k} \neq \bot$, the set $D_{c,k}$ contains all edges $\{a, b\}$ in the graph for which $h_{c,k}(a) = h_{c,k}(b)$, and, at the time the edge was added, $\texttt{curr} < c$. All edges for which $\texttt{curr} = c$ held at the time the edge was added are stored in B. If $h_{\texttt{curr},k}(u) \neq h_{\texttt{curr},k}(v)$, then the colors $(\chi(u), h_{\texttt{curr},k}(u))$ and $(\chi(v), h_{\texttt{curr},k}(v))$ assigned to u and v differ in the second coordinate. Otherwise, the edge $\{u, v\} \in D_{\texttt{curr},k} \cup B$, so the greedy coloring of $D_{\texttt{curr},k} \cup B$ will assign different values to $\chi(u)$ and $\chi(v)$. This ensures the colors assigned to u and v differ in the first coordinate.

Finally, the output color space $[(\Delta + 1)] \times [\ell^2]$ has size $\leq (\Delta + 1)\Delta^2 = O(\Delta^3)$. □

Lemma 4.4.4. *Algorithm 4.4.1 requires only $\widetilde{O}(n)$ bits of space; this includes random bits.*

Proof of Lemma 4.4.4. Because $|\mathcal{U}| = O(\operatorname{poly} n)$, picking a random hash function from $\mathcal{U}$ requires only $O(\log n)$ random bits. As the algorithm stores $\ell p = O(\ell \log n)$ of these hash functions for the variables $(h_{i,j})_{i\in[\ell],j\in[p]}$, the total space needed by these function is $O(\ell(\log n)^2)$.

Next, for each of the sets of edges $D_{i,j}$, for $i \in [\ell], j \in [p]$, Lines 12 through 15 ensure that $|D_{i,j}|$ is always $\leq \frac{n\Delta}{\ell^2} + 1$; sets that grow too large are replaced by $\bot$. Since edges can be stored using $O(\log n)$ bits, the total space usage of all the $D_{i,j}$ is $O(\frac{n\Delta}{\ell^2} \cdot \ell \cdot p \cdot \log n) = O\left(\frac{n\Delta}{\ell}(\log n)^2\right)$. Similarly, the buffer B never contains more than $\frac{n\Delta}{\ell}$ edges, since it is reset when the condition of Line 7 is true; thus B can be stored with $O(\frac{n\Delta}{\ell} \log n)$ bits. The counter `curr` is negligible. In total, the algorithm uses $O\left(\frac{n\Delta}{\ell}(\log n)^2\right) = O\left(n(\log n)^2\right)$ bits of space.[9] □

Corollary 4.4.5. *By adjusting parameters of Algorithm 4.4.1, we obtain for any $\beta \in [0, 1]$ a robust, random-seed algorithm for $O(\Delta^{3-2\beta})$-coloring using $\widetilde{O}(n\Delta^\beta)$ space.*

Proof. Use Algorithm 4.4.1 with the parameter $\ell = \lceil \Delta^{1-\beta} \rceil$. The proofs on which Theorem 4.4.1 relies all go through as is, except that the number of colors used is now $\ell^2(\Delta+1) = O(\Delta^{3-2\beta})$, and the algorithm uses space $O\left(\frac{n\Delta}{\ell}(\log n)^2\right) = O(n\Delta^\beta(\log n)^2)$. □

Remark 4.4.6. For $O(\Delta)$-coloring algorithms in the static setting, $O((\log n)^2)$ random bits suffice. (See remark at end of Section 4.3.2). However, Algorithm 4.4.1 uses $O(\Delta(\log n)^2)$ random bits in total. Is this optimal? We suspect – but have not proven – that robust algorithms for graph

[9]It is possible to reduce the space usage to $O\left(\frac{n\Delta}{\ell^2}(\log n) \max(\log n, \ell)\right)$ space, by adjusting the algorithm to share a pool of $O(\max(\log n, \ell))$ hash functions between all epochs.

coloring with $n/2$ colors, which use semi-streaming space in fact require $\widetilde{\Omega}(\Delta)$ bits of randomness, if $\Delta = \Omega(\log n)$, and we allow the graph stream to repeat edges.

Say that we could prove that pseudo-deterministic algorithms for $(n/2)$-coloring, when $\Delta \leq n/4$, require $\Omega(n\Delta/\operatorname{polylog} n)$ space. (Given the commonalities between the pseudo-deterministic and deterministic lower bounds for MissingItemFinding from Sections 3.5 and 3.6, and the fact that [ACS22]'s lower bound for deterministic graph coloring uses fundamentally the same approach as we used for MissingItemFinding, this may well be possible.) Let r be the number of random bits used by a robust random-seed graph coloring algorithm, for some $r \leq \Delta/4$. Let $\mathcal{U}$ be the set of all streams which are split into $O(r)$ equally sized epochs, where within each epoch the stream edges form a graph of maximum degree $O(\Delta/r)$. Then, using a similar argument to Lemma 3.7.1, one can show that, when the streaming algorithm must be correct on all streams in $\mathcal{U}$, it will require space at least as large as that for a pseudo-deterministic algorithm on graphs of maximum degree $O(\Delta/r)$—which may be $\Omega(n\Delta/(r \operatorname{polylog} n))$.

4.5 A robust random-oracle algorithm

In this section, we give an adversarially robust $O(\Delta^{5/2})$-coloring algorithm using semi-streaming space *and* assuming access to a random oracle. This algorithm can be seen as an awkward hybrid of two different $O(\Delta^3)$-coloring algorithms; one which is most effective for graph streams in which vertex degrees change slowly and steadily, and one which is most effective for graph streams in which vertex degrees change sporadically but by large amounts.

We assume that $\sqrt{\Delta}$ is an integer (if not, we can work with $\lceil \sqrt{\Delta} \rceil$ which will not affect the asymptotic color or space bounds). We also assume that $\Delta = \Omega((\log n)^2)$; if Δ is smaller, we can store the entire graph in semi-streaming space and then color it optimally.

The following graph-theoretic concept plays a crucial role in our algorithm.

Definition 4.5.1 (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 refer to this as a (degeneracy + 1)-coloring.

4.5.1 High-level description and techniques

We first 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 chunk 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 $((\ell-1)\sqrt{\Delta}, \ell\sqrt{\Delta}]$.
- **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 if $\deg_B(v) > \sqrt{\Delta}$; otherwise, we say that it is in the *slow* zone. We also use the terms *slow vertex* and *fast vertex*, respectively.
- **Block.** We have multiple coloring functions, denoted by h_i and g_i, that assign each node a color uniformly at random from a palette of suitable size (not to be confused with the final proper coloring; these colorings are improper). As a result, we obtain a partition of the nodes into monochromatic classes that we call "blocks." A 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 called 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 receives edges of the graph and stores it only if it is f-monochromatic, as an f-sketch.

Next, we describe how to color the slow vertices using $O(\Delta^{5/2})$ colors in semi-streaming space. Then we do the same for the fast vertices.

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 and moves on to the next chunk. We initialize Δ many coloring functions $h_1, \ldots, h_\Delta$ that run in parallel. For each i, the function h_i assigns each node a color from $[\Delta^2]$ uniformly at random. An h_i-sketch (see definition above) processes the prefix of the stream until the end of chunk i. Recall that "processing" means it stores a received edge (u, v) in the set A_i if it is h_i-monochromatic.

Suppose a query arrives in the current epoch $\texttt{curr}$. Fix a subgraph induced by *only* the slow vertices in an arbitrary $h_{\texttt{curr}}$-block on the edge set $A_{\texttt{curr}-1} \cup B$ (set $A_0 := \emptyset$). Recolor this subgraph using an offline $\Delta' + 1$-coloring algorithm where Δ' is its max-degree. Now do this for each $h_{\texttt{curr}}$-block, using fresh palettes for the distinct blocks. We then return the resultant coloring (for the slow nodes). We now argue that the number of edges stored in $(\cup_i A_i)$ 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 epoch i. But note that, until that point, we only base our output on A_js for $j < i$, which are independent on

h_i in particular. Therefore, we ensure that each h_i-sketch processes a part of the stream independent of their randomness. Hence, 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 it properly colors the graph induced by the slow nodes. Observe that we indeed stored each edge of the input graph, which is contained in any $h_{\texttt{curr}}$ block of slow vertices, in $A_{\texttt{curr}-1} \cup B$. This is because if it is in B, we have definitely stored it, and otherwise, it was in an epoch $\leq \texttt{curr} - 1$. Therefore, the $h_{\texttt{curr}-1}$-sketch received it and must have stored it in $A_{\texttt{curr}-1}$. This means each intra-block edge is properly colored by the offline algorithm, and 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 $A_{\texttt{curr}-1} \cup B$ is $O(\sqrt{\Delta} + \log n) = O(\sqrt{\Delta})$ since Δ is assumed to be $\Omega((\log n)^2)$. Hence, each $h_{\texttt{curr}}$-block of slow nodes induced on $A_{\texttt{curr}-1} \cup B$ is colored with a fresh palette of $O(\sqrt{\Delta})$ colors by the offline algorithm. There are Δ^2 many $h_{\texttt{curr}}$-blocks, and therefore, we use $O(\Delta^2 \cdot \sqrt{\Delta}) = O(\Delta^{5/2})$ colors.

Coloring fast vertices. To handle these, we use another $\sqrt{\Delta}$ 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, i.e., if $g_i(u) = g_i(v)$.

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. Thus, 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 that each edge is 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. Thus, the total number of edges stored by $C_1, \ldots, C_{\sqrt{\Delta}}$ is $O(n/\sqrt{\Delta} \cdot \sqrt{\Delta}) = O(n)$.

When a query arrives, for each level i, we consider the fast vertices in each g_i-block. Then consider the subgraph induced by these vertices on the edge set $C_i \cup B$. 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_i-block of fast vertices is stored in $C_i \cup B$. This follows from the "fastness" property of the nodes: if any such edge $\{u, v\}$ is not in the buffer B, then, since the degrees of u and v increased by at least $\sqrt{\Delta}$ in the buffer, the nodes u and v must have been at levels lower than i when $\{u, v\}$ was inserted. Therefore, it was fed to the g_i-sketch, which stored it since it is g_i-monochromatic. Hence, each intra-block edge of fast vertices is properly colored by the offline algorithm, and each inter-block edge is also properly colored since we use distinct palettes for distinct blocks.

4.5.2 The robust algorithm and its analysis

We now present the pseudocode of our algorithm in Algorithm 4.5.1. The analysis of correctness, robustness, space usage, and color bound is given below.

Lemma 4.5.2. *With high probability, for all vertices $x \in V$, we have $\sum_{i \in [\sqrt{\Delta}]} d_{C_i}(v) = O(\log n)$.*

Proof of Lemma 4.5.2. For any $x \in V$, let D be the random variable for the degree of x at the end of the stream, and let $\{x, Y_1\}$, $\{x, Y_2\}$, ... $\{x, Y_D\}$ be the edges added adjacent to x by the adversary, in order. For all $k \in [\Delta]$ and $\ell \in \sqrt{\Delta}$, let $Z_{k,\ell}$ be the random variable which is 1 if $k \le D$ and the algorithm stores the edge $\{x, Y_k\}$ in the set C_ℓ, and zero otherwise. The edge $\{x, Y_k\}$, assuming it exists, will be stored in C_i only if $g_i(x) = g_i(Y)$ and $i \ge \left\lceil \frac{\max(d(x), d(Y_k))}{\sqrt{\Delta}} \right\rceil + 1$, where $d(x)$ and $d(Y_k)$ are the values of the degree counter at the time the edge was added. Now, consider the sequence of random variables,

$$Z_{1,1}, \ldots, Z_{1,\sqrt{\Delta}}, Z_{2,1}, \ldots, Z_{2,\sqrt{\Delta}}, \ldots, Z_{\Delta,1}, \ldots, Z_{\Delta,\sqrt{\Delta}}. \tag{4.4}$$

Their sum is precisely $\sum_{\ell \in [\sqrt{\Delta}]} d_{C_\ell}(x)$. In order to bound this sum with high probability, we would like to use Lemma 2.3.1, but in order for that to work we need to prove that the expectation of a given $Z_{k,\ell}$, conditional on all the earlier terms in the sequence, is bounded. Let $\prec$ indicate the lexicographic order on pairs of the form (k', ℓ'), so that $(k'', \ell'') \prec (k', \ell')$ if either $k'' < k'$, or ($k'' = k'$ and $\ell'' < \ell'$.). Define $Z_{\prec(k,\ell)}$ to be the vector $(Z_{k',\ell'})_{(k',\ell') \prec (k,\ell)}$. We want to prove an upper bound on $\mathbb{E}[Z_{k,\ell} \mid Z_{\prec(k,\ell)}]$. Intuitively, the edge $\{x, Y_k\}$ chosen by the adversary will either definitely not be stored in C_k – because e.g. one of the degrees of the endpoints is too large – or, when it is time to check whether $g_\ell(x) = g_\ell(Y_k)$, the value read from $g_\ell(Y_k)$ will not have been revealed to the adversary so far, nor will it have been read as part of any test to determine if $\{x, Y_{k'}\}$ should be stored in $C_{k'}$, for $(k', k') \prec (k, k)$; so $g_\ell(Y_k)$ will be independent of the variables in $Z_{\prec(k,\ell)}$, and will equal $g_\ell(x)$ with probability exactly $1/\Delta^{3/2}$. Either way, we will find $\mathbb{E}[Z_{k,\ell} \mid Z_{\prec(k,\ell)}] \le 1/\Delta^{3/2}$.

Algorithm 4.5.1 Adversarially robust $O(\Delta^{2.5})$-coloring in semi-streaming space

```
    Input: Stream of edge insertions of 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                                  ▷ $\Delta$ parallel copies for $\Delta$ possible epochs
 3:     Let $h_i : V \to [\Delta^2]$ be uniformly random      ▷ $h_i$ assigns each node a color from $[\Delta^2]$ u.a.r.
 4:     $A_i \leftarrow \emptyset$                                                  ▷ edges stored by $h_i$-sketch
 5: for $i$ from 1 to $\left[\sqrt{\Delta}\right]$ do                      ▷ $\sqrt{\Delta}$ parallel copies for $\sqrt{\Delta}$ possible levels
 6:     Let $g_i : V \to [\Delta^{3/2}]$ be uniformly random ▷ $g_i$ assigns each node a color from $[\Delta^{3/2}]$ u.a.r.
 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
11:     $B \leftarrow \emptyset$; curr $\leftarrow$ curr $+ 1$          ▷ Empty buffer if full and update epoch number
12: $B \leftarrow B \cup \{\{u, v\}\}$;                                  ▷ Update buffer and buffer size
13: $d(u) \leftarrow d(u) + 1$; $d(v) \leftarrow d(v) + 1$                     ▷ Increase degrees of $u$ and $v$
14: for $i$ from (curr $+ 1$) to $\Delta$ do              ▷ Consider copies corresponding to higher epochs
15:     if $h_i(u) = h_i(v)$ then $A_i \leftarrow A_i \cup \{\{u, v\}\}$          ▷ Store $h_i$-monochromatic edges in $A_i$
16: for $i$ from $\left\lceil \frac{\max\{d(u),d(v)\}}{\sqrt{\Delta}} \right\rceil + 1$ to $\Delta$ do           ▷ Consider levels higher than both $u$ and $v$
17:     if $g_i(u) = g_i(v)$ then $C_i \leftarrow C_i \cup \{\{u, v\}\}$          ▷ Store $g_i$-monochromatic edges in $C_i$

    Query():
18: $F \leftarrow \{v \in V : \deg_B(v) > \sqrt{\Delta}\}$    ▷ $F$ contains fast vertices receiving $> \sqrt{\Delta}$ edges in the buffer
19: $S \leftarrow V \setminus F$                                     ▷ $S$ contains the remaining slow vertices
20: for $c$ from 1 to $[\Delta^2]$ do
21:     $S_{\mathtt{curr}}(c) \leftarrow \{w \in S : h_{\mathtt{curr}}(w) = c\}$            ▷ Consider each $h_{\mathtt{curr}}$-block among slow vertices
22:     Using fresh colors, (degree+1)-color subgraph induced by $S_{\mathtt{curr}}(c)$ on edge set $A_{\mathtt{curr}-1} \cup B$
23: for $\ell$ from 1 to $\left[\sqrt{\Delta}\right]$ do
24:     for $c$ from 1 to $[\Delta^{3/2}]$ do
25:         $F(\ell, c) \leftarrow \left\{w \in F : \left\lceil \frac{d(w)}{\sqrt{\Delta}} \right\rceil = \ell, \text{ and } g_\ell(w) = c\right\}$      ▷ Consider each $g_\ell$-block among fast
    vertices
26:         Using fresh colors, (degeneracy+1)-color subgraph induced by $F(\ell, c)$ on edge set $C_\ell \cup B$
27: Output resultant coloring for $S \cup F = V$
```

Now, applying Lemma 2.3.1 to the sequence of random variables from Eq. 4.4, we obtain:

$$\Pr\left[\sum_{\ell \in [\sqrt{\Delta}]} d_{C_\ell}(x) \geq 5 \log n\right] \leq \Pr\left[\sum_{k \in [\Delta]} \sum_{\ell \in [\sqrt{\Delta}]} Z_{k,\ell} \geq \Delta^{3/2} \cdot \frac{1}{\Delta^{3/2}}(1 + 4\log n)\right]$$
$$\leq \exp(-(1 + 4\log n)\ln(1 + 4\log n) - 4\log n) \leq e^{-4.04 \log n} \leq \frac{1}{n^5}.$$

Then taking a union bound of this event for each $x \in V$, we conclude that $\sum_{i \in [\sqrt{\Delta}]} d_{C_i}(x) = O(\log n)$ holds for *all* x with high probability. □

Lemma 4.5.3. *With high probability, for all vertices $x \in V$, we have $\sum_{i \in [\Delta]} d_{A_i}(v) = O(\log n)$.*

Proof of Lemma 4.5.3. The argument here is essentially the same as for the proof of Lemma 4.5.2, so we will skip most of the details, and describe briefly what changes.

Instead of defining indicator random variables $Z_{k,\ell}$ for the event that Algorithm 4.5.1 stores a given edge $\{x, Y_k\}$ in C_ℓ, we define indicator random variables $Z_{k,i}$, for $i \in [\Delta]$, for the event that the algorithm stores $\{x, Y_k\}$ in A_i. With a similar lexicographically ordered sequence of the $Z_{k,i}$, one can prove that each random variable $Z_{k,i}$ has expectation $\leq \frac{1}{\Delta^2}$, even after conditioning on the values of all the earlier random variables in the sequence. This will use the observation that, if the answer to whether the edge $\{x, Y_k\}$ will be stored in the set A_i depends on the value of $h_i(Y_k)$, then the value of $h_i(Y_k)$ has not been revealed to the adversary. Applying Lemma 2.3.1, one will then find:

$$\Pr\left[\sum_{i \in [\Delta]} d_{A_i}(x) \geq 5 \log n\right] \leq \Pr\left[\sum_{k \in [\Delta]} \sum_{i \in [\Delta]} Z_{k,i} \geq \Delta^2 \cdot \frac{1}{\Delta^2}(1 + 4 \log n)\right] \leq \exp(-4.04 \log n) \leq \frac{1}{n^5} .$$

The proof is completed by taking a union bound. □

Lemma 4.5.4. *The space usage of Algorithm 4.5.1 is $\widetilde{O}(n)$ bits, with high probability.*

Proof of Lemma 4.5.4. By Lemmas 4.5.3 and 4.5.2, all vertices $x \in V$ satisfy $\sum_{i \in [\Delta]} d_{A_i}(v) = O(\log n)$, and $\sum_{i \in [\sqrt{\Delta}]} d_{C_i}(v) = O(\log n)$, with high probability. Since

$$|C_i| = \frac{1}{2} \sum_{x \in V} d_{C_i}(x) \qquad \text{and} \qquad |A_i| = \frac{1}{2} \sum_{x \in V} d_{A_i}(x) ,$$

it follows Algorithm 4.5.1 stores $O(n \log n)$ edges in total in $\bigcup_{i \in [\Delta]} A_i \cup \bigcup_{i \in [\sqrt{\Delta}]} C_i$. Additionally, it stores a buffer B of n edges. Hence, the algorithm stores $\widetilde{O}(n)$ edges in total. Further, it stores a degree counter for each node and a couple of counters for tracking the buffer size and the epoch number. These take an additional $\widetilde{O}(n)$ bits of space. Thus, the total space usage of the algorithm is $\widetilde{O}(n)$ bits. □

Lemma 4.5.5. *At any point in the stream, for each $\ell \in [\sqrt{\Delta}]$ and $c \in [\Delta^{3/2}]$, the degeneracy of the subgraph induced by the vertex set $F(\ell, c)$ on the edge set $C_\ell \cup B$ is $O(\sqrt{\Delta})$, w.h.p.*

Proof of Lemma 4.5.5. To every vertex $v \in F(\ell, c)$, define t_v to be the length of the input stream at the time that the degree counter $d(v)$ of v increased from $(\ell - 1)\sqrt{\Delta}$ to $(\ell - 1)\sqrt{\Delta} + 1$; in other words, the time that vertex v entered level ℓ. By Lemma 4.5.2, with high probability it holds that

$d_{C_\ell}(v) = O(\log n)$, so the set C_ℓ contributes at most $O(\log n) = O(\sqrt{\Delta})$ to the degeneracy of the induced subgraph of the edge set $C_\ell \cup B$ on the vertex set $F(\ell, c)$.

It thus suffices to prove that the degeneracy of the graph H on vertices of $F(\ell, c)$ formed by edges from $B \setminus C_\ell$ is $\leq \sqrt{\Delta}$. Orient each edge $\{u, v\}$ in H from u to v if $t_v \geq t_u$, and from v to u otherwise. We will prove that the out-degree of each vertex from $F(\ell, c)$ in H will be $\leq \sqrt{\Delta}$.

Fix some $x \in F(\ell, c)$; for each edge $(x, y) \in H$, let d_{xy} be the value of $d(x)$ directly after the streaming algorithm processed the edge $\{x, y\}$. Because $x \in F(\ell, c)$, we have $d_{xy} \leq \ell\sqrt{\Delta}$. Since $x, y \in F(\ell, c)$, $g_\ell(x) = g_\ell(y) = c$. Because $\{x, y\} \in B \setminus C_\ell$, $\max(d_{xy}, d_{yx})$ must have been $\geq (\ell - 1)\sqrt{\Delta} + 1$ – otherwise the algorithm would have recorded the edge $\{x, y\}$ in C_ℓ. Now the orientation of the edge applies: because $t_y \geq t_x$, the vertex x must have reached degree $(\ell-1)\sqrt{\Delta}+1$ at the same time or before y did. Thus $d_{xy} \leq (\ell - 1)\sqrt{\Delta}$ implies $d_{yx} \leq (\ell - 1)\sqrt{\Delta}$; since we know $\max(d_{xy}, d_{yx}) > (\ell-1)\sqrt{\Delta}$, it follows $d_{xy} \leq (\ell-1)\sqrt{\Delta}$. Since the variable $d(x)$ increases with each new edge adjacent to x that arrives, and $d_{xy} \in [(\ell - 1)\sqrt{\Delta} + 1, \ell\sqrt{\Delta}]$ for all out-edges (x, y) of x in H, we conclude by the pigeonhole principle that x has out-degree $\leq \sqrt{\Delta}$ in H. This completes the proof that the degeneracy of H is $\sqrt{\Delta}$, and thus of the lemma.

□

Lemma 4.5.6. *Whenever queried, Algorithm 4.5.1 outputs a proper coloring of the current graph G and uses $O(\Delta^{5/2})$ colors w.h.p.*

Proof of Lemma 4.5.6. By Lemma 4.5.3 and Lemma 4.5.2, with high probability,

$$\max_{x \in V} \left(\sum_{i \in [\sqrt{\Delta}]} d_{C_i}(v) + \sum_{i \in [\Delta]} d_{A_i}(v) \right) = O(\log n) \,. \tag{4.5}$$

We shall see that if this holds, then Algorithm 4.5.1 will produce an $O(\Delta^{2.5})$ coloring of the graph.

The total number of colors used is the sum of the number of colors used for the coloring of each of the subsets of vertices $S_{\texttt{curr}}(c)$ (for $c \in [\Delta^2]$) and $F(\ell, c)$ (for $c \in [\Delta^{3/2}], \ell \in [\sqrt{\Delta}]$). Because each of these subsets uses a fresh set of colors, and the subsets together disjointly cover the entire vertex set, the coloring output by Algorithm 4.5.1 is valid if an only if all the individual subset colorings are valid.

For each $c \in [\Delta^2]$, consider the set $S_{\texttt{curr}}(c)$. For each edge $\{x, y\}$ in the graph, both of whose endpoints are in $S_{\texttt{curr}}(c)$, we observe that either the edge $\{x, y\}$ was added while the value of `curr` was less than it was now – in which case the algorithm would have stored $\{x, y\} \in A_{\texttt{curr}}$, because $h_{\texttt{curr}}(x) = h_{\texttt{curr}}(y)$ – or edge $\{x, y\}$ was added while `curr` had its current value – in which case $\{x, y\}$ is in the set B. Thus, $A_{\texttt{curr}} \cup B$ includes all the edges of the subgraph of G induced by $S_{\texttt{curr}}(c)$, so the degree + 1 coloring of $S_{\texttt{curr}}(c)$ will be valid.

Every vertex x in $S_{\texttt{curr}}(c)$ satisfies $\deg_B(x) \le \sqrt{\Delta}$, by the definition of the set S of slow vertices. By Eq. 4.5, $\deg_{C_{\texttt{curr}}}(x) = O(\log n) = O(\sqrt{\Delta})$. Thus the maximum degree the edge set $C_{\texttt{curr}} \cup B$ for any vertex in $S_{\texttt{curr}}(c)$ will be $O(\sqrt{\Delta})$, and so a degree+1 coloring will only use $O(\sqrt{\Delta})$ colors.

Now for $c \in [\Delta^{3/2}]$ and $\ell \in [\sqrt{\Delta}]$, consider the set $F(\ell, c)$ of vertices. To prove that the coloring of this set is correct, we must show that every edge $\{x, y\}$ which is contained in G, and which has both endpoints in $F(\ell, c)$, must be recorded in either B or in C_ℓ. Let d_x and d_y be the values of $d(x)$ and $d(y)$ after the Algorithm 4.5.1 processes the edge $\{x, y\}$, i.e., after Line 13 has executed. We have two cases: either $\ell_{x,y} = \left\lceil \max(d_x, d_y)/\sqrt{\Delta} \right\rceil$ is equal to ℓ, or it must be less than ℓ. If $\ell_{x,y} < \ell$, then the edge will be recorded in C_ℓ by Line 17. Both the degree check and the check that $g_\ell(x) = g_\ell(y)$ will pass, the latter because $x, y \in F(\ell, c)$ implies $g_\ell(x) = g_\ell(y) = c$. On the other hand, if $\ell_{x,y} = \ell$, then say without loss of generality that $\left\lceil d_x/\sqrt{\Delta} \right\rceil = \ell$ – this implies the degree of x just after the edge $\{x, y\}$ was added was at least $(\ell - 1)\sqrt{\Delta} + 1$. Meanwhile, because $x \in F(\ell, c)$, the current degree of x must be at most $(\ell - 1)\sqrt{\Delta}$. As each new edge adjacent to x increases $d(x)$ by one, $\{x, y\}$ must be one of the $\sqrt{\Delta}$ most recent edges added adjacent to x. Since $x \in F$, the last $\sqrt{\Delta}$ edges adjacent to x are all stored in B, and thus $\{x, y\} \in B$. The completes the proof that the coloring of $F(\ell, c)$ will be correct.

By Lemma 4.5.5, the degeneracy of the subgraph induced by the vertex set $F(\ell, c)$ on edge set $C_\ell \cup B$ will be $O(\sqrt{\Delta})$, assuming Eq. 4.5 holds. As Algorithm 4.5.1 computes a degeneracy+1 coloring of this graph, it will use $O(\sqrt{\Delta})$ colors.

We have proven that each of the subsets of the form $S_{\texttt{curr}}(c)$ or $F(\ell, c)$ will be properly colored using $O(\sqrt{\Delta})$ fresh colors. Since there are $2\Delta^2$ such subsets in total, we conclude that algorithm Algorithm 4.5.1 produces an $O(\Delta^{5/2})$ coloring of the graph as a whole. □

Combining Lemma 4.5.6 with Lemma 4.5.4 proves the theorem:

Theorem 4.5.7. *There is an $O(\Delta^{5/2})$-coloring algorithm which is robust (with total error probability $\le \delta$) against adaptive adversaries, and runs in $O(n \operatorname{polylog} n \cdot \log \frac{1}{\delta})$ bits of space, given oracle access to $O(n\Delta)$ bits of randomness.*

Furthermore, we have:

Corollary 4.5.8. *By adjusting parameters of Algorithm 4.5.1, we obtain for any $\beta \in [0, 1]$ a robust $O(\Delta^{(5-3\beta)/2})$-coloring algorithm using $\widetilde{O}(n\Delta^\beta)$ space, assuming oracle access to $O(n\Delta)$ bits of randomness.*

Proof of Corollary 4.5.8. These parameter changes do not significantly affect the proofs of correctness for Algorithm 4.5.1.

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

- 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:

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

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

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

and by the fast zone:

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

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

4.6 A multipass deterministic algorithm

This section gives a multipass deterministic semi-streaming algorithm for $(\Delta+1)$-coloring, proving Theorem 4.6.11. As usual, let $G = (V, E)$ denote the input graph, which 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 the algorithm we discuss first, $L_x = [\Delta+1]$ for each $x \in V$.

We will later need the following lemma that establishes the existence and constructibility of a large independent set in a sparse graph.

Lemma 4.6.1 (A constructive variation on 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.*

Proof of Lemma 4.6.1. We prove that we can in deterministic polynomial time find an independent set in graph G of size $\geq \psi(G) := \sum_{x\in V} \frac{1}{\deg x + 1}$. By Jensen's inequality,

$$\psi(G) \geq \frac{|V|^2}{\sum_{x\in V}(\deg x + 1)} = \frac{n^2}{n+2m}\,.$$

The procedure is straightforward: let $U \leftarrow V$ be the set of "uncovered" vertices, and $I \leftarrow \emptyset$ the independent set, which we will progressively expand. While U is not empty, pick $x \in U$ minimizing $\sum_{y \in N[x]} \frac{1}{\deg_{G[U]}(y)+1}$, and remove the closed neighborhood $N[x]$ from U, and add x to I. To prove that this produces a set I of size $\geq \psi(G)$, we show that every time a new vertex is picked, $\psi(G[U])$ decreases by at most 1. First, note that:

$$\min_{x\in U} \sum_{y\in N[x]} \frac{1}{\deg_{G[U]}(y)+1} \leq \frac{1}{|U|}\sum_{x\in U}\sum_{y\in N[x]} \frac{1}{\deg_{G[U]}(y)+1} = \frac{1}{|U|}\sum_{z\in U} \frac{|N[z]|}{\deg_{G[U]}(z)+1} = \frac{|U|}{|U|} = 1\,.$$

Second,

$$\begin{aligned}
\psi(G[U]) - {}&\psi(G[U \setminus N[x]]) \\
&= \sum_{z\in U} \frac{1}{\deg_{G[U\setminus N[x]]} z + 1} - \sum_{z\in U\setminus N[x]} \frac{1}{\deg_{G[U\setminus N[x]]} z + 1} \\
&= \sum_{z\in N[x]} \frac{1}{\deg_{G[U\setminus N[x]]} z + 1} + \sum_{z\in U\setminus N[x]} \left(\frac{1}{\deg_{G[U]}(z)+1} - \frac{1}{\deg_{G[U\setminus N[x]]}(z)+1}\right) \\
&\leq \sum_{z\in U} \frac{1}{\deg_{G[U\setminus N[x]]} z + 1} + \sum_{z\in U\setminus N[x]} 0\,,
\end{aligned}$$

because $\deg_{G[U]}(z) \geq \deg_{G[U]\setminus N[x]}(z)$. Combining these two inequalities gives $\psi(G[U \setminus N[x]]) \geq \psi(G[U]) - 1$.

□

4.6.1 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 constant fraction of the vertices in U, thereby shrinking $|U|$ to $\alpha|U|$, for some constant $\alpha < 1$. In the beginning, $U = V$. After at most $\lceil \log_{1/\alpha} \Delta \rceil$ such epochs, we will have $|U| \le n/\Delta$: 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. See Algorithm 4.6.1 for the high level pseudocode.

Algorithm 4.6.1 Deterministic semi-streaming algorithm for $(\Delta+1)$-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$(G, U, \chi)$        ▷ See Algorithm 4.6.2; shrinks $|U|$ to at most $\alpha|U|$
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 $G$
```

Each epoch of the algorithm is divided into *stages*, where each stage whittles down a set of proposed colors for each uncolored vertex. To explain this better, the following definition is useful.

Definition 4.6.2 (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 every two 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

$$\operatorname{slack}(x \mid T) = \max\{0, |T \cap L_x| - |\{y \in \mathrm{N}(x) \setminus U : \chi(y) \in T\}|\}, \tag{4.6}$$

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

$$\Phi = \Phi(U, \chi, \mathcal{P}) = \sum_{\{x,y\} \in E} \mathbb{1}_{x \in U \wedge y \in U} \cdot \mathbb{1}_{P_x = P_y} \cdot \left(\frac{1}{s_x} + \frac{1}{s_y}\right) \tag{4.7}$$

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 slightly from the "slack" defined by [HKNT22], where the number of colors used by the neighbors is known exactly. It turns 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 using an upper bound on the number of used colors, $|\{y \in \mathrm{N}(x) \setminus U : \chi(y) \in T\}|$, instead of the exact quantity $|T \cap \{\chi(y) : y \in \mathrm{N}(x) \setminus U\}|$ – is that the former can be written as a summation $\sum_{y:\{x,y\}\in E} \mathbb{1}_{y\notin U} \mathbb{1}_{\chi(y)\in T}$ over the data stream, which can be easily computed in $O(\log n)$ space. 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 [GK21, BKM20] do not need such a modified notion of "slack".

The set $\mathrm{Free}(T, x) := (T \cap L_x) \setminus \{\chi(y) : y \in \mathrm{N}(x) \setminus U\}$ 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 $|\mathrm{Free}(T, x)| \geq \mathrm{slack}(x \mid T)$, 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 $\mathrm{Free}(P_x, x)$, the only monochromatic edges we might create are within U and the number, $m_{\mathrm{mono}}(U, \chi, \mathcal{P})$, of such edges satisfies

$$\mathbb{E} m_{\mathrm{mono}}(U, \chi, \mathcal{P}) = \sum_{\substack{\{x,y\}\in E(G[U]) \\ P_x = P_y}} \frac{|\mathrm{Free}(P_x, x) \cap \mathrm{Free}(P_y, y)|}{|\mathrm{Free}(P_x, x)| \cdot |\mathrm{Free}(P_y, y)|} \leq \sum_{\substack{\{x,y\}\in E(G[U]) \\ P_x = P_y}} \left(\frac{1}{s_x} + \frac{1}{s_y}\right) = \Phi\,. \quad (4.8)$$

4.6.2 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. (4.8), ensures that assigning each $x \in U$ the sole surviving color in P_x would not create too many monochromatic edges. Now, Lemma 4.6.1 allows us to commit to these proposed colors for at least $(1 - \alpha)|U|$ 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.[10] 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

$$w_{x,\mathbf{j}} = \frac{\mathrm{slack}(x \mid P_{x,\mathbf{j}})}{\sum_{\mathbf{i} \in \{0,1\}^k} \mathrm{slack}(x \mid P_{x,\mathbf{i}})} \,. \tag{4.9}$$

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

$$\mathbb{E}\Phi(U,\chi,\widetilde{\mathcal{P}}) = \Phi(U,\chi,\mathcal{P}) \,. \tag{4.10}$$

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 in a space-efficient manner 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 not-too-large value p. By using a suitable map $g\colon U \times [p] \to \{0,1\}^k$, we can use a uniform random value in $[p]$ to sample from a distribution *close enough* to the $(w_{x,\mathbf{j}})$ distribution. Then, for each x, we shrink P_x to $P_{x,\mathbf{j}(x)}$ where $\mathbf{j}(x) = g(x, h(x))$ and $h \in_R \mathcal{H}$. Let $\mathcal{P}_h$ denote the resulting collection of proposed color sets.

It turns out that a prime $p = \Theta(n \log n)$ suffices for the guarantees we will eventually need. Thus, by choosing (e.g.) the Carter–Wegman family of affine functions on $\mathbb{F}_p$, we can take $|\mathcal{H}| = O(n^2(\log n)^2)$. This enables us to use two streaming passes[11] with $\widetilde{O}(n)$ space to identify a specific function $h \in \mathcal{H}$ for which $\Phi(U,\chi,\mathcal{P}_h)$ is "approximately below" the average value for functions in $\mathcal{H}$. We will 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 will then allow us to shrink U by

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

[11]Can we reduce the number of passes to one, using a different hash family? We use 2-independence for Eq. 4.13 to hold for arbitrary weight pairs $w_{u,\mathbf{j}}, w_{v,\mathbf{j}}$, and the weights can be as small as $1/\Delta$, *suggesting* that $\Omega(\Delta^2)$ different functions may be needed, which is too many if e.g. $\Delta = \Omega(n^{2/3})$; but there might be ways around this.

a constant factor α, as noted earlier.

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

4.6.3 Detailed algorithm and proof of correctness

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

$$Q^{(i)}_{\mathbf{j}} := \left\{\mathbf{a} \in \{0,1\}^b : (a_{ki-k+1}, \dots, a_{ki}) = \mathbf{j}\right\}; \quad \mathcal{Q}^{(i)} := \left\{Q^{(i)}_{\mathbf{j}}\right\}_{\mathbf{j}\in\{0,1\}^k}. \tag{4.11}$$

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.

Before we proceed, we also need the following lemma:

Lemma 4.6.3. *For $\varepsilon > 0$, $p \geq n/\varepsilon$, and $\mathbf{w} = (w_{x,\mathbf{j}})_{x\in U, \mathbf{j}\in\{0,1\}^k}$ there is a function $g_{\mathbf{w}} \colon U \times [p] \to \{0,1\}^k$ satisfying:*

$$\frac{|g^{-1}_{\mathbf{w}}(x,\mathbf{j})|}{p} \leq w_{x,\mathbf{j}}\,(1+\varepsilon)\,, \quad \forall\, \mathbf{j} \in \{0,1\}^k\,.$$

Proof of Lemma 4.6.3. As $\sum_{\mathbf{j}\in\{0,1\}^k} w_{x,\mathbf{j}} = 1$, we can do this by directing the first $\lfloor p w_{x,\mathbf{0}}(1+\varepsilon)\rfloor$ entries of $g_{\mathbf{w}}(x,\cdot)$ to the pattern $\mathbf{0}$; the next $\lfloor p w_{x,\mathbf{1}}(1+\varepsilon)\rfloor$ entries to the pattern $\mathbf{1}$; and so on (where $\mathbf{0}, \mathbf{1}, \dots$ is an enumeration of $\{0,1\}^k$), stopping as soon as all p entries of $g_{\mathbf{w}}(x,\cdot)$ are filled.

We now argue that $g_{\mathbf{w}}$ is well-defined, i.e., that every entry $g_{\mathbf{w}}(x,\cdot)$ is indeed filled. Examining eq. (4.6), since every slack value is at most n, every nonzero value $w_{x,\mathbf{j}}$ is $\geq 1/n$. Recalling that $p \geq 8n\log n$, we observe that for such $\mathbf{j}$,

$$\lfloor p w_{x,\mathbf{j}}\,(1+\varepsilon)\rfloor \geq p w_{x,\mathbf{j}} + (p w_{x,\mathbf{j}}\varepsilon) - 1 \geq p w_{x,\mathbf{j}} + \frac{n(1/n)\varepsilon}{\varepsilon} - 1 = p w_{x,\mathbf{j}}\,,$$

so a total of $\geq \sum_{\mathbf{j}\in\{0,1\}^k} p w_{x,\mathbf{j}} \geq p$ entries $g_{\mathbf{w}}(x,\cdot)$ will be covered. □

The full logic for the epoch is given in Algorithm 4.6.2.

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. (4.7).

Algorithm 4.6.2 Partial coloring step for Algorithm 4.6.1

```
1: procedure COLORING-EPOCH(graph $G$, partial coloring $(U, \chi)$)
2:     $\varepsilon \leftarrow \frac{1}{8 \log n}$                    ▷ precision for hash functions and sum calculations
3:     $b \leftarrow \lceil \log(\Delta + 1) \rceil$                    ▷ each color is a b-bit vector
4:     $k \leftarrow 1 + \lfloor \log(n/|U|) \rfloor$                   ▷ number of bits fixed in each stage
5:     for all $x \in U$ do $P_x \leftarrow \{0,1\}^b$                  ▷ the initial, trivial PCC
6:     for all stage $i$, from 1 through $\lceil b/k \rceil$ do
7:         pass 1:
8:             for all $x \in U$ and $Q \in \mathcal{Q}^{(i)}$ do compute slack$(x \mid P_x \cap Q)$ by using eq. (4.6)
9:         end
10:        Determine all $w_{x,\mathbf{j}}$ values using eq. (4.9), noting that $P_{x,\mathbf{j}} = P_x \cap Q^{(i)}_{\mathbf{j}}$
11:        $p \leftarrow$ prime in $[\lceil n/\varepsilon \rceil, 2\lceil n/\varepsilon \rceil]$;  $\mathcal{H} \leftarrow \{z \mapsto az + b : a, b \in \mathbb{F}_p\}$   ▷ Carter–Wegman hashing
12:        Implicitly construct $g_{\mathbf{w}} : U \times [p] \to \{0,1\}^k$ as per Lemma 4.6.3.
13:        For each $h \in \mathcal{H}$, define $\mathcal{P}_h = \{P_{x,h}\}_{x \in U}$, where $P_{x,h} := P_x \cap Q^{(i)}_{g_{\mathbf{w}}(x,h(x))}$
14:        ▷ Identify an $h^\star \in \mathcal{H}$ for which $\Phi(U, \chi, \mathcal{P}_{h^\star})$ is not much larger than average, as follows:
15:        pass 2:
16:            Split $\mathcal{H}$ into $\sqrt{|\mathcal{H}|}$ parts
17:            Estimate $\sum_h \Phi(U, \chi, \mathcal{P}_h)$ for each part, up to $(1+\varepsilon)$ relative error
18:            Pick the part minimizing the estimated sum
19:        end
20:        pass 3:
21:            Estimate $\Phi(U, \chi, h)$ for each $h$ within the chosen part, up to $(1+\varepsilon)$ relative error
22:            Choose $h^\star$ as the (approximate) minimizer
23:        end
24:        for all $x \in U$ do $P_x \leftarrow P_{x,h^\star}$              ▷ constrain the PCC more tightly

25:    end-of-epoch pass:                                          ▷ each $P_x$ is now a singleton
26:        Collect $F \leftarrow \{\{u, v\} \in E : u \in U, v \in U$, and $P_u = P_v\}$     ▷ we will prove that
   $|F| = O(|U|)$
27:        In the graph $(V, F)$, find an independent set $I$ with $|I| \ge (1-\alpha)|U|$, using Lemma 4.6.1
28:        for all $x \in I$ do                                      ▷ extend $\chi$ by coloring $x$
29:            $U \leftarrow U \setminus \{x\}$
30:            $\chi(x) \leftarrow$ the sole element in $P_x$
31:    end
```

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

$$\mathrm{d_{conf}}(x) = \mathrm{d_{conf}}(x; U, \chi, \mathcal{P}) := |\{y \in \mathrm{N}(x) \cap U : P_y = P_x\}|, \tag{4.12}$$

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 $s_x = \mathrm{slack}(x \mid P_x)$ from Definition 4.6.2.

Lemma 4.6.4. *For every PCC, $\Phi(U, \chi, \mathcal{P}) = \sum_{x \in U} \mathrm{d_{conf}}(x)/s_x$.*

Proof of Lemma 4.6.4. From the definitions in eqs. (4.6) and (4.7), using some straightforward algebra,

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

Lemma 4.6.5. *For all x and disjoint sets T_1, T_2: $\mathrm{slack}(x \mid T_1 \sqcup T_2) \le \mathrm{slack}(x \mid T_1) + \mathrm{slack}(x \mid T_2)$.*

This lemma follows immediately from eq. (4.6) and the fact that $\max\{0, a_1 + a_2\} \le \max\{0, a_1\} + \max\{0, a_2\}$.

Lemma 4.6.6. *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 of Lemma 4.6.6. Recalling that each $L_x \cap P_x = L_x = [\Delta + 1]$ for the initial PCC, we use eqs. (4.6) and (4.12) to derive

$$s_x - \mathrm{d_{conf}}(x) = \max\{0, \Delta + 1 - |\mathrm{N}(x) \setminus U|\} - |\mathrm{N}(x) \cap U| = \Delta + 1 - \deg(x) \ge 1.$$

Thus, $\mathrm{d_{conf}}(x)/s_x \le 1$ (and is not "0/0") for all $x \in U$. Lemma 4.6.4 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 $u \ne v$,

$$\begin{aligned}
\Pr[g_{\mathbf{w}}(u, h(u)) = g_{\mathbf{w}}(v, h(v)) = \mathbf{j}] &= \Pr[g_{\mathbf{w}}(u, h(u)) = \mathbf{j}] \cdot \Pr[g_{\mathbf{w}}(v, h(v)) = \mathbf{j}] \\
&= \Pr\left[h(u) \in g_{\mathbf{w}}^{-1}(u, \mathbf{j})\right] \cdot \Pr\left[h(v) \in g_{\mathbf{w}}^{-1}(v, \mathbf{j})\right] \\
&\le w_{u,\mathbf{j}} w_{v,\mathbf{j}} (1 + \varepsilon)^2 \\
&\le e^{2\varepsilon} w_{u,\mathbf{j}} w_{v,\mathbf{j}}.
\end{aligned} \tag{4.13}$$

To keep the rest the derivation compact, let us abbreviate "slack" to "sk." The candidate PCCs $\mathcal{P}_h$ defined in line 13 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 $\mathcal{H}$:

$$\begin{aligned}
&\mathbb{E}\Phi(U,\chi,\mathcal{P}_h)\\
&= \sum_{\{u,v\}\in E} \mathbb{E}\mathbb{1}_{u\in U}\mathbb{1}_{v\in U}\cdot\mathbb{1}_{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(G[U])}\sum_{\mathbf{j}\in\{0,1\}^k}\Pr\left[P_{u,h}=P_{u,\mathbf{j}}=P_{v,\mathbf{j}}=P_{v,h}\right]\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 } 13}{=} \sum_{\{u,v\}\in E(G[U])}\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]\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. (4.13)}}{\leq} \sum_{\substack{\{u,v\}\in E(G[U])\\ P_u=P_v}}\sum_{\mathbf{j}\in\{0,1\}^k} e^{2\varepsilon}w_{u,\mathbf{j}}w_{v,\mathbf{j}}\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. (4.9)}}{=} e^{2\varepsilon}\sum_{\substack{\{u,v\}\in E(G[U])\\ 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\left(\frac{1}{\mathrm{sk}(u\mid P_{u,\mathbf{j}})}+\frac{1}{\mathrm{sk}(v\mid P_{v,\mathbf{j}})}\right)\\
&= e^{2\varepsilon}\sum_{\substack{\{u,v\}\in E(G[U])\\ 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}})}\\
&= e^{2\varepsilon}\sum_{\substack{\{u,v\}\in E(G[U])\\ 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 } 4.6.5}{\leq} e^{2\varepsilon}\sum_{\substack{\{u,v\}\in E(G[U])\\ P_u=P_v}}\left(\frac{1}{\mathrm{sk}(u\mid P_u)}+\frac{1}{\mathrm{sk}(v\mid P_u)}\right)\\
&= e^{2\varepsilon}\Phi_i\,. \qquad (4.14)
\end{aligned}$$

Thus, picking $h^\star$ with $\Phi(U,\chi,\mathcal{P}_{h^\star})$ below average would ensure $\Phi_{i+1}\leq e^{2\varepsilon}\Phi_i$. However, due to precision constraints, each of lines 17 and 21 could contribute a relative error of $(1+\varepsilon)$, so the $h^\star$ actually picked by the algorithm gives only the following weaker guarantee:

$$\Phi_{i+1}\leq(1+\varepsilon)^2 e^{2\varepsilon}\Phi_i\leq e^{4\varepsilon}\Phi_i\,.$$

Since the number of stages in the epoch is $\ell\leq\lceil b/k\rceil\leq\log(\Delta+1)\leq\log n$, and $\varepsilon=1/(8\log n)$ we have

$$\Phi_\ell\leq\left(e^{4\varepsilon}\right)^\ell\Phi_0\leq e^{1/2}|U|\leq 2|U|\,. \qquad \square$$

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

Proof of Lemma 4.6.7. At the start of the epoch, $s_x \geq |L_x| - |\operatorname{N}(x)| = (\Delta + 1) - \deg(x) \geq 1$.

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 $P_{x,\mathbf{j}} = P_x \cap Q_{\mathbf{j}}^{(i)}$ for a pattern $\mathbf{j} \in \{0,1\}^k$ satisfying $w_{x,\mathbf{j}} > 0$: the way $g_{\mathbf{w}}$ is defined (Lemma 4.6.3) ensures this. By Lemma 4.6.5,

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

so there exists $\mathbf{j} \in \{0,1\}^k$ for which $\operatorname{slack}(x \mid P_{x,\mathbf{j}}) \geq 1$. One such $\mathbf{j}$ must be picked as the chosen pattern for x, because $w_{x,\mathbf{j}} > 0$ implies $\operatorname{slack}(x \mid P_{x,\mathbf{j}}) > 0$. Consequently, the new value of P_x chosen at the end of the stage (line 24) will continue to satisfy the invariant $s_x \geq 1$.

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. It is not possible that P_x is empty, because $|P_x \cap L_x| \geq s_x \geq 1$. Thus $|P_x \cap L_x| = s_x = 1$. ☐

Lemma 4.6.8. *The set F collected at the end of an epoch satisfies $|F| \leq |U|$.*

Proof of Lemma 4.6.8. Using the terminology of Lemma 4.6.6, at the end of an epoch, we have

$$2|U| \overset{\text{lemma 4.6.6}}{\geq} \Phi_\ell \overset{\text{lemma 4.6.4}}{=} \sum_{x \in U} \frac{\operatorname{d_{conf}}(x)}{s_x} \overset{\text{lemma 4.6.7}}{=} \sum_{x \in U} \frac{|\{y \in \operatorname{N}(x) \cap U : P_x = P_y\}|}{1} = 2|F|\,. \quad ☐$$

Lemma 4.6.9. *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 $|U'| \leq \frac{2}{3}|U|$.*

Proof of Lemma 4.6.9. As noted before, 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 $\operatorname{N}(x)$ because $s_x \neq 0$ (Lemma 4.6.7). 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 4.6.1, (V, F) contains an independent set I of size

$$|I| \geq \frac{|U|^2}{2|F| + |U|} \overset{\text{lemma 4.6.8}}{\geq} \frac{|U|}{3}$$

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

4.6.4 Space and pass complexity

Lemma 4.6.10. *Algorithm 4.6.1 runs in $O(n(\log n)^2)$ bits of space and $O(\log\Delta\cdot\log\log\Delta)$ streaming passes.*

Proof of Lemma 4.6.10. For the space bound, it suffices to establish that Coloring-Epoch runs in $O(n(\log n)^2)$ space. At each stage of an epoch, the algorithm maintains the current PCC, consisting of the partial coloring (U,χ) and the collection $\mathcal{P}=\{P_x\}_{x\in U}$. The former can be stored in $O(n\log\Delta)$ bits directly; so can the latter, since the subcube structure of P_x allows for a natural $O(b)=O(\log\Delta)$-bit description.

We now turn to the space required to execute the passes. Focus on stage i within epoch r. Computing the slack values in pass 1 requires $|U|2^k$ counters, one for each pair $(x,Q_{\mathbf{j}}^{(i)})$, to determine $|\{y\in\mathrm{N}(x):\chi(y)\in P_x\cap Q_{\mathbf{j}}^{(i)}\}|$. Each such counter fits in $O(\log\Delta)$ bits. By our choice of k, the total space bound for these counters is $O(n\log\Delta)$. Moving on, identifying $h^\star$ requires keeping track of $\sqrt{|\mathcal{H}|}$ accumulators, to evaluate sums of the form given in line 13, in each of passes 2 and 3. These accumulators do not need to be stored at full precision; a relative error of $(1+\varepsilon)$ is acceptable, so $O(\log\frac{n}{\varepsilon})=O(\log n)$ bits per accumulator suffice. Since $p=\Theta(n/\varepsilon)=\Theta(n\log n)$ and $|\mathcal{H}|=p^2$ (line 11), the total space cost of all the accumulators is $O(\sqrt{|H|}\log n)=O(n(\log n)^2)$ bits.

Next, we consider the end-of-epoch pass. By Lemma 4.6.8, $|F|\le|U|=O(n)$ so this pass needs only $O(n\log n)$ bits to collect the edges in F. The rest of its computations happen offline and need no further storage. This completes the space complexity analysis.

Finally, we account for the number of passes. In epoch r, there are $\lceil b/k_r\rceil$ stages, where k_r is the value of k for the epoch; each such stage makes three streaming passes; additionally, there is one end-of-epoch pass. There is also one final pass after all epochs are done (line 6). By Lemma 4.6.9, each epoch shrinks $|U|$ to at most $\alpha=2/3$ times its previous value. Notice that the epochs stop once $|U|\le n/\Delta$, so there are at most $\lceil\log_{1/\alpha}\Delta\rceil$ epochs. Furthermore, at the start of the rth epoch, $|U|\le\alpha^{r-1}n$, implying $k_r\ge 1+\lfloor(r-1)\log\alpha^{-1}\rfloor$ for this epoch, which in turn upper-bounds the number of stages of the epoch. Putting it all together, the total number of streaming passes, across all epochs, is

$$\begin{aligned}1+\sum_{r=1}^{\lceil\log_{1/\alpha}\Delta\rceil}\left(3\left\lceil\frac{b}{k_r}\right\rceil+1\right)&=O(\log\Delta)+O(b)\cdot\sum_{r=1}^{\lceil\log_{1/\alpha}\Delta\rceil}\frac{1}{k_r}\\&=O(\log\Delta)\cdot\sum_{r=1}^{\lceil\log_{1/\alpha}\Delta\rceil}\frac{1}{r}\\&=O(\log\Delta\cdot\log\log\Delta)\,.\end{aligned}$$

□

This concludes the proof of our first major algorithmic result, which we now recap.

Theorem 4.6.11. *There is an efficient deterministic semi-streaming algorithm to* $(\Delta + 1)$*-color an* n*-vertex graph, given a stream of its edges. The algorithm uses* $O(n(\log n)^2)$ *bits of space and runs in* $O(\log \Delta \log\log \Delta)$ *passes.*

4.6.5 Extending to list coloring

We can extend Algorithm 4.6.1 to handle the more general problem of $(\deg +1)$-list-coloring. This will require a careful refinement of some of the low-level details of the previous algorithm.

The main problem that we face for $(\deg +1)$-list-coloring is that the set of all possible colors that vertices might use can now be much larger than Δ. If all vertices have disjoint color lists, then there may be $\Theta(n\Delta)$ colors in use. To work around this issue, instead of using a fixed partition refinement scheme, we choose the color partitions as a function of the color lists. Fortunately, it suffices to derive the color partitions from a small, almost universal family of hash functions. The following lemmas give such families.

Definition 4.6.12. A family $\mathcal{H}$ of hash functions $A \to B$ is δ-almost universal if for all distinct $a_1, a_2 \in A$,

$$\Pr_{h \in_R \mathcal{H}}[h(a_1) = h(a_2)] \le \delta\,.$$

Lemma 4.6.13. *For finite sets* A, B*, there is a* $3/|B|$*-almost universal hash family* $\mathcal{H}$ *of functions* $A \to B$*, of size* $|\mathcal{H}| = t = O((|B| \log |A|)^2)$*, wherein each function is computable in* $\operatorname{poly}(\log(|A|, \log |B|)$ *time.*

Proof of Lemma 4.6.13. Let $\alpha = \lceil \log |A| / \log |B| \rceil$, and let q be a prime between $|B|\alpha$ and $2|B|\alpha$; such a prime is guaranteed to exist by Bertrand's postulate. Note $q = O(|B| \log |A|)$. We will construct a $3/|B|$-almost-universal hash family by composing a hash family from A (viewed as a subset of $\mathbb{Z}_q^\alpha$) to $[q]$ with a hash family from $[q]$ to $\mathbb{Z}_{|B|} \cong B$. For the first family, we use the hash functions derived from the Reed-Solomon codes via the equivalence between hash functions and codes [Sti96]; for the second, we use the standard multiplicative family. Thus, we obtain a family $\mathcal{H}$ of q^2 functions from $\mathbb{Z}_q^\alpha$ to $\mathbb{Z}_{|B|}$, defined as:

$$\mathcal{H} := \left\{ (x_1, \ldots, x_\alpha) \mapsto \big(a \sum_{i=1}^{\alpha} x_i b^{i-1} \bmod q\big) \bmod |B| : a \in \mathbb{Z}_q, b \in \mathbb{Z}_q \right\}.$$

To prove that this is a $3/|B|$-almost universal family, we observe that given distinct vectors $(x_1, \ldots, x_\alpha)$ and $(y_1, \ldots, y_\alpha)$,

$$\Pr_{h \in_R \mathcal{H}}[h(x_1, \ldots, x_\alpha) = h(y_1, \ldots, y_\alpha)] \le \Pr_{h \in_R \mathcal{H}}\left[\sum_{i=1}^{\alpha} x_i b^i \equiv \sum_{i=1}^{\alpha} y_i b^i \pmod q\right]$$

$$+ \quad \Pr\left[(a\tilde{x} \bmod q) \not\equiv (a\tilde{y} \bmod q) \pmod{|B|} \middle| \tilde{x} \neq \tilde{y}\right],$$

where $\tilde{x} = \sum_{i=1}^{\alpha} x_i b^i \bmod q$, and $\tilde{y}$ is likewise defined. By a standard proof (see e.g. [ACS22, Lemma 5.1]), the second probability is $\leq \frac{2}{|B|}$. On the other hand, using the fundamental theorem of algebra,

$$\Pr_{b \in \mathbb{Z}_q}\left[\sum_{i=1}^{\alpha}(x_i - y_i)b^{i-1} \bmod q = 0\right] = \frac{\alpha - 1}{q} \leq \frac{1}{|B|},$$

and thus

$$\Pr_{h \in_R \mathcal{H}}[h(x_1, \dots, x_\alpha) = h(y_1, \dots, y_\alpha)] \leq \frac{1}{|B|} + \frac{2}{|B|} \leq \frac{3}{|B|}.$$

As they only require multiplications, additions, and remainders, the functions in $\mathcal{H}$ can be computed in $O(\alpha(\log q)^2) = O((1 + \frac{\log|A|}{\log|B|})(\log|B| \log\log|A|)^2)$ time. (Note: identifying the prime q deterministically by testing integers in the candidate interval for primarily may use up to $O(q \operatorname{polylog} q)$ time, but if e.g. Cramer's conjecture holds and prime gaps are small, then only $O(\operatorname{polylog} q)$ time would be needed to find q.) □

While it is theoretically possible to use a significantly smaller hash function family, we haven't yet found an explicit construction for this.

Lemma 4.6.14. *For finite sets A, B, there* exists *a $3/|B|$-almost universal hash family $\mathcal{H}$ of functions $A \to B$, of size $|\mathcal{H}| = O(|B| \log |A|)$.*

Proof of Lemma 4.6.14. Say that each function in $\mathcal{H}$ is chosen uniformly and independently at random from the set of all functions from A to B. It suffices to prove that $\mathcal{H}$ fails to be $3/|B|$-almost universal with probability < 1. Let $|\mathcal{H}| = \lceil 8|B| \log|A| \rceil$ and behold:

$$\begin{aligned}
&\Pr_{\mathcal{H}}\left[\exists x, y : \Pr_{h \in_R H}[h(x) = h(y)] \geq 3/|B|\right] \\
&\leq \sum_{\{x,y\} \in \binom{A}{2}} \Pr_{\mathcal{H}}\left[\Pr_{h \in_R H}[h(x) = h(y)] \geq 3/|B|\right] \\
&= \sum_{\{x,y\} \in \binom{A}{2}} \Pr_{\mathcal{H}}\left[\sum_{h \in H} \mathbb{1}_{h(x)=h(y)} \geq 3\mathbb{E}[\sum_{h \in H} \mathbb{1}_{h(x)=h(y)}]\right] \\
&\leq \sum_{\{x,y\} \in \binom{A}{2}} (e^2/9)^{\mathbb{E}[\sum_{h \in H} \mathbb{1}_{h(x)=h(y)}]} = \sum_{\{x,y\} \in \binom{A}{2}} (e^2/9)^{|\mathcal{H}|/|B|} \qquad \text{by Chernoff bound} \\
&< |A|^2 2^{-|\mathcal{H}|/(4|B|)} \leq 1.
\end{aligned}$$

□

Theorem 4.6.15. *Let C be a set of colors. There is a deterministic semi-streaming algorithm for $(\text{degree}+1)$-list-coloring a graph G given a stream consisting of, in any order, the edges of G, and (x, c) pairs specifying that the color $c \in C$ is allowed for the vertex x. We assume no (vertex,color) pair is repeated, and that the list L_x of colors that vertex x receives has length $\deg(x)+1$. The algorithm uses $O(n(\log n)^2(\log|C|)^2)$ 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 4.6.16. *Let $s \geq 3$, and let C be a set. There exists a family $\mathcal{F}$ of $O(s^2(\log|C|)^2)$ partitions of C so that, for every collection $L_1, \ldots, L_t$ of subsets of C:*

$$\frac{1}{|\mathcal{F}|}\sum_{\mathcal{R}\in\mathcal{F}}\sum_{i\in[t]}\max_{S\in\mathcal{R}}(|L_i \cap S| - 1) \leq \sqrt{\frac{3}{s}}\sum_{i\in[t]}(|L_i| - 1)\,, \tag{4.15}$$

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

Proof of Lemma 4.6.16. Let $\mathcal{H}$ be a $3/s$-almost universal hash family $C \to [s]$, which has size $|\mathcal{H}| = O(s^2(\log|C|)^2)$. (For example, from Lemma 4.6.13.) Let h be a randomly chosen element of $\mathcal{H}$, and let $\mathcal{R} = \{R_1, \ldots, R_s\}$ 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 $z \geq 1$, $z - 1 = f^{-1}(\binom{z}{2})$, we have for any $i \in [t]$ that:

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

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

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

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

$$\mathbb{E}\sum_{j\in[s]}\binom{L_i \cap R_j}{2} = \mathbb{E}\sum_{x,y\in L_i : x\neq y}\mathbb{1}_{h(x)=h(y)} = \sum_{x,y\in L_i : x\neq y}\Pr[h(x) = h(y)] \leq \binom{|L_i|}{2}\frac{3}{s}\,.$$

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

$$f\left(\sqrt{\frac{3}{s}}(z-1)\right) = \frac{\sqrt{\frac{3}{s}}(z-1)\cdot(\sqrt{\frac{3}{s}}(z-1)+1)}{2} = \frac{3}{s}\,\frac{(z-1)(z+\sqrt{\frac{s}{3}}-1)}{2} \geq \frac{3}{s}\binom{z}{2}\,,$$

which implies $f^{-1}(\frac{3}{s}\binom{z}{2}) \leq \sqrt{\frac{3}{s}}(z-1)$. Thus:

$$\mathbb{E} \max_{j \in [s]}(|L_i \cap R_j| - 1) \leq f^{-1}\left(\binom{|L_i|}{2}\frac{3}{s}\right) \leq \sqrt{\frac{3}{s}}(|L_i| - 1)\,.$$

By linearity of expectation, it follows

$$\mathbb{E} \sum_{i \in [t]} \max_{j \in [s]}(|L_i \cap R_j| - 1) \leq \sqrt{\frac{3}{s}} \sum_{i \in [t]}(|L_i| - 1)\,.$$

This is equivalent to Eq. 4.15, if we let $\mathcal{F}$ be the set of possible values of $\mathcal{R}$.

□

Proof of Theorem 4.6.15. There are two main changes to the algorithm in Theorem 4.6.11. First, because the color lists L_x are no longer fixed, computing $\text{slack}(x \mid P_x \cap Q)$ 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, colors) pair arrives, the total space usage from this stage is still $O(|U| 2^k \log \Delta)$.

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 setting $k = 1 + \left\lfloor \log \frac{n}{|U|} \right\rfloor$, we set $k = 1 + \left\lfloor \frac{1}{3} \log \frac{n}{|U|} \right\rfloor$. Also, the number of stages ℓ used will, instead of $\lceil \log(\Delta + 1) \rceil$, be $\ell = \lceil 2b/k \rceil + 1$ instead. For the first $\lceil 2b/k \rceil$ stages, we adaptively pick partitions from the family of Lemma 4.6.16 over the set $\mathcal{C}$ with $s = 2^{k+2}$ parts. The resulting partitions use $O(\ell \log s \log \log |\mathcal{C}|) = O((\log \Delta)^2 \log \log |\mathcal{C}|)$ space to store in total. Because there are only $O(s^2 (\log C)^2)$ candidate partitions, we can find the best partition from Lemma 4.6.16 in a single pass over the stream. We use an array of integer counters $(a_{\mathcal{R},j,x})_{\mathcal{R} \in \mathcal{F}, j \in [s], x \in U}$, each initialized to zero. Let $\mathcal{R}_j$ be the jth partition in $\mathcal{R}$. The counter $a_{\mathcal{R},j,x}$ records the number of (vertex, color) pairs (x, c) in the stream for which $c \in \mathcal{R}_j \cap P_x$. Because we are promised that no (vertex, color) pairs are repeated, at the end of the pass $a_{\mathcal{R},j,x} = |(P_x \cap \mathcal{R}_j) \cap L_x|$. We will then choose $\mathcal{Q}^{(i)}$ to be the partition $\mathcal{R} \in \mathcal{F}$ which minimizes the quantity $\sum_{x \in U} \min_{j \in [s]} a_{\mathcal{R},j,x}$. The total additional space needed to perform this pass is:

$$O(|\mathcal{F}| s |U| \log \Delta) = O(s^3 |U| \log \Delta (\log |C|)^2) = O(n \log \Delta (\log |C|)^2)\,.$$

since $s = O((n/|U|)^{1/3}$.

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 4.6.16).

$$\sum_{x\in U}(|L_x \cap P_{x,j_x}| - 1) \leq \sum_{x\in U} \max_{j\in[s]}(|L_x \cap P_x \cap Q_j^{(i)}| - 1) \leq \sqrt{\frac{3}{s}} \sum_{x\in U}(|L_x \cap P_x| - 1)$$

Each stage scales $\sum_{x\in U}(|L_x \cap P_x| - 1)$ by a factor which is $\leq \frac{3}{s} < 2^{-k/2}$, so after $\ell - 1 = \lceil 2\log(\Delta+1)/k \rceil$ stages, we have

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

In the last stage, we set $\mathcal{Q} = \{\{x\} : x \in \mathcal{C}\}$, where $\mathcal{C} = \bigcup_{x\in U} L_x$. Unlike the other stages, where $|\mathcal{Q}| \leq 2^{k+2}$, 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 $O(|U|\log|\mathcal{C}|)$ 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 $(x, Q) \in U \times \mathcal{C}$ combination, we store a $\{0,1\}$ value for each $(x, Q) \in \sqcup_{x\in U}\{(x, \{q\}) : q \in L_x \cap P_x\}$ 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 4.6.11 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 uses only a constant factor more counters ($2^{k+2}|U|$ in total). 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. Due to the additional pass per stage and the increased number of stages per epoch, the algorithm will use about eight times more passes than in Theorem 4.6.11. The precision of the estimates in the passes finding $h^\star$ will need to be improved accordingly, but we will still have $\varepsilon = \Omega(1/\log n)$.

Storing the per vertex partitions P_x requires only $\ell(k+2) + \log(|\mathcal{C}|) = O(\log|\mathcal{C}|)$ 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 almost no adjustment. $\square$

4.7 Slightly improved deterministic lower bound

In this section, we describe a few changes which strengthen the result of [ACS22, Theorem 1], to obtain:

Corollary 4.7.1. *[Modification of [ACS22], Theorem 1] For any integer $L \geq 8\log n \ln(2n)$, a* *deterministic algorithm which maintains an $n/2$-coloring (or better) of a graph edge insertion stream of maximum degree $\leq L$ requires space:*

$$\geq \frac{nL}{16 \ln 2 (\log n)^2} \,. \tag{4.16}$$

For sake of completeness, we present a proof of Corollary 4.7.1 below; it largely repeats that of [ACS22, Theorem 1], using slightly different notation. The main change is an improvement to the lower bound applied once the graph G_{k+1} has been found, in which we send the edges in G_{k+1} to the algorithm before asking the algorithm for a coloring; compare [ACS22, Lemma 4.13], which immediately evaluates the number of colors used by the algorithm. It will rely on the following lemma.

Lemma 4.7.2 (Paraphrased from [ACS22, Lemma 4.3])**.** *Let $G = (V, E)$ be an n-vertex graph, $s \geq 1$ an integer, $p \in (0, 1)$, and d a real value which is $\geq$ the maximum degree of a vertex in G, and which also satisifes $d \geq 4\ln(2n)/p$. Let $\mathcal{H}_{G,2pd}$ be the set of subgraphs of G of maximum degree $< 2pd$. Let Ψ map $\mathcal{H}_{G,2pd}$ to $\{0,1\}^s$, and define $E_{miss}(\Psi, \phi) := \{e \in G : \forall H \in \mathcal{H}_{G,2pd}, e \notin H\}$. Then there exists $\psi^\star \in \{0,1\}^s$ for which*

$$|E_{miss}(\Psi, \phi)| \leq \frac{1}{p}(s+1)\ln 2$$

Proof of Corollary 4.7.1. Say that $L \geq 8 \log n \ln(2n)$, and that deterministic graph coloring algorithm $\mathcal{A}$ uses $\leq n/2$ colors and s bits of space, and assume for sake of contradiction that $s \leq \frac{nL}{16 \ln 2 (\log n)^2}$. Identify all of $\mathcal{A}$'s states with unique strings in $\{0,1\}^s$. The first deviation from the proof of [ACS22, Theorem 1] is a slightly different choice of parameters. Define $k = \left\lfloor \log \frac{2n \log n}{L} \right\rfloor$, and for each $i \in [k]$, set:

$$d_i = \frac{2n}{2^i} \qquad \text{and} \qquad p_i = \frac{L}{4n \log n} 2^i \,.$$

(This ensures $d_i p_i = \frac{L}{2 \log n} \geq 4 \ln(2n)$. Since $i \in [k]$, we also have $p_i \leq \frac{L}{4n \log n} \frac{2n \log n}{L} \leq \frac{1}{2}$.)

We will iteratively construct a family of inputs on which $\mathcal{A}$ must (if it is to be correct) use $> n/2$ colors. Let $V_1 = V$, let G_1 be the clique on V_1, and let σ_1 be the initial state of $\mathcal{A}$. For $i = 1, \ldots, k$, we will choose:

- a collection $\mathcal{F}_i$ of subgraphs of G_i, each of maximum degree $< 2p_i d_i$
- a set of vertices $V_{i+1} \subseteq V_i$, satisfying $|V_{i+1}| \geq n - i\frac{n}{4k}$
- a graph G_{i+1} on V_{i+1}, of maximum degree $\leq d_{i+1}$, containing precisely the edges between vertices of V_{i+1} which are in G_i but not in any $F \in \mathcal{F}_i$

- a state σ_{i+1} of the algorithm, reachable from σ_1 by a graph stream formed from any sequence $(F_1, \ldots, F_{i+1}) \in \mathcal{F}_1 \times \ldots \times \mathcal{F}_i$

Since $|V_1| = n$, and $d_1 = n$, the maximum degree of G_1 is $\leq d_1$; we also have that σ_1 is reachable from σ_1 by the empty graph stream. Thus, these conditions hold trivially for G_1, V_1, σ_1.

For each $i = 1, \ldots, k$, define the set of graphs $\mathcal{H}_i$ of subgraphs of G_i of maximum degree $< 2p_i d_i$; this matches the definition of $\mathcal{H}_{G_i, 2p_i d_i}$ from Lemma 4.7.2. Since the maximum degree of G_i is $\leq d_i$, and $p_i \in (0,1)$, and $2p_i d_i \geq 4\ln(2n)$, the preconditions of Lemma 4.7.2 are satisfied. Define $\Phi_i : \mathcal{H}_i \to \{0,1\}^s$ to map each subgraph H in $\mathcal{H}_i$ to the state of the algorithm $\mathcal{A}$ that will result if, starting from state σ_i, the algorithm processes the edges of H in some canonical order. Then, applying Lemma 4.7.2, there exists a $\sigma_{i+1} \in \{0,1\}^s$ for which

$$|E_{miss}(\Phi_i, \sigma_{i+1})| \leq \frac{1}{p_i}(s+1)\ln 2\,. \tag{4.17}$$

Define $\mathcal{F}_i = \Phi_i^{-1}(\sigma_{i+1})$, so that $E_{miss}(\Phi_i, \sigma_{i+1}) = G_i \setminus \bigcup_{F \in \mathcal{F}_i} F$. Define $V_{i+1} \subseteq V_i$ to be the set of vertices with degree $\leq d_{i+1}$ in $E_{miss}(\Phi_i, \sigma_{i+1})$, and define G_{i+1} to be the graph on V_{i+1} formed by edges from $E_{miss}(\Phi_i, \sigma_{i+1})$. Then all vertices in $V_i \setminus V_{i+1}$ have degree $> d_{i+1}$ in $E_{miss}(\Phi_i, \sigma_{i+1})$, so:

$$|E_{miss}(\Phi_i, \sigma_{i+1})| \geq \frac{1}{2} d_{i+1} |V_i \setminus V_{i+1}|\ . \tag{4.18}$$

Combining Eqs. 4.17 and 4.18 implies[12]

$$\begin{aligned} |V_i \setminus V_{i+1}| &\leq \frac{(s+1)2\ln 2}{p_i d_{i+1}} \leq s \cdot \frac{4\ln 2}{p_i d_{i+1}} \\ &\leq \frac{nL}{16\ln 2(\log n)^2}(4\ln 2)\frac{\log n}{L} \leq \frac{n}{4\log n} \leq \frac{n}{4k}\,. \end{aligned}$$

Thus $|V_{i+1}| \geq |V_i| - |V_i \setminus V_{i+1}| \geq n - (i-1)\frac{n}{4k}$.

We observe that the state σ_{k+1} is reachable from σ_1 by adding edges from any sequence $(F_1, \ldots, F_k) \in \mathcal{F}_1 \times \ldots \times \mathcal{F}_k$; the graphs in each such sequence are edge disjoint and each have maximum degree $\leq \frac{L}{2\log n} \leq \frac{L}{2k}$. Consequently, each union $\bigcup_{i \in [k]} F_i$ will have maximum degree $\leq L/2$. Every edge between vertices in V_{k+1} which is not in G_{k+1} could possibly have been included by a graph in one of the $\mathcal{F}_1, \ldots, \mathcal{F}_k$.

The second deviation from the proof of [ACS22, Theorem 1] is the following. Consider the state $\sigma_\star$ which $\mathcal{A}$ will reach, starting from state σ_{k+1}, after sending every edge in G_{k+1} to the algorithm. Since the maximum degree of G_{k+1} is $d_{k+1} \leq \frac{2n}{2^{k+1}} \leq \frac{8nL}{2n\log n} \leq L/2$, every graph formed by the union of a sequence $(F_1, \ldots, F_k, G_{k+1})$ with $(F_1, \ldots, F_k) \in \mathcal{F}_1 \times \ldots \times \mathcal{F}_k$ will have maximum degree $\leq L$. The algorithm $\mathcal{A}$ is required, on $\sigma_\star$, to produce a valid coloring for each such graph; but it

[12] Applying the upper bound on s here, immediately, is another change from the proof of [ACS22, Theorem 1].

cannot color any two vertices in V_{k+1} the same, because there might have been an edge between the two. Consequently, it must use $\geq |V_{k+1}| \geq 3n/4 > n/2$ distinct colors. This contradicts the requirement that $\mathcal{A}$ use only $\leq n/2$ colors, so our initial assumption that $s \leq \frac{nL}{16 \ln 2 (\log n)^2}$ must have been wrong.

□

4.8 Conclusion

This chapter proved a $\widetilde{\Omega}(\Delta^2)$ color lower bound for adversarially robust streaming algorithms for graph coloring using semi-streaming space; described robust semi-streaming algorithms for $O(\Delta^3)$ and $O(\Delta^{2.5})$ coloring, and described a deterministic semi-streaming algorithm for $\Delta + 1$ coloring using $O(\log \Delta \log \log \Delta)$ passes over the stream.

We have not addressed the more general case of dynamic graph streams, in which each stream element corresponds to an action to either add *or delete* an edge. In general, the robust algorithms we present can easily be adapted to work in this setting, because unlike edge insertions, edge deletions do not require that the algorithm change its output coloring. The two main changes necessary are to add logic to remove deleted edges from the algorithms' buffers (but not reduce degree counters), and to adjust algorithm parameters as necessary to account for the fact that the stream length can now be longer than m. See the ArXiv version of [CGS22] for a design specifically tuned for dynamic graph streams.

We note that some of our results may be extendable to $O(\kappa^c)$ coloring, where $c \geq 1$ and κ is the maximum degeneracy of the input. The lower bound Theorem 4.3.3 of course translates, since a graph with maximum degree Δ will have degeneracy $\kappa \leq \Delta$. The $O(\Delta^3)$-coloring algorithm Theorem 4.4.1 may be easily adaptable – it should suffice to replace Δ with κ in the algorithm, and double the number of epochs to account for the fact that the graph stream will have $\leq \kappa n$ edges in total, instead of $\leq n\Delta/2$.

It is not fully clear why the color gap between robust upper and lower bounds remains. Graph coloring is more complicated than just running many instances of algorithms for MissingItemFinding from Chapter 3 in parallel. To maintain a graph coloring, for each vertex v in the graph, we must not just pick a color that avoids the colors of its neighbors at the time the edges to them were added, but ensure that whenever one of the neighbors' colors changes, v's color is still not equal any of them. It may be that this coordination requirement – that vertices' colors maintain the graph coloring invariant, when their neighbors are updated – is the barrier that has made finding a robust semi-streaming $O(\Delta^2)$-coloring algorithm so difficult, and which requires Algorithm 4.5.1's different approaches to handling vertices with quickly changing and slowly changing neighborhoods. We leave the task of finding a robust $O(\Delta^2)$-coloring algorithm in semi-streaming space as an open question.

Chapter 5

Streaming algorithms for online edge coloring

5.1 Introduction

A proper edge-coloring of a graph or a multigraph colors its edges such that no two adjacent edges share the same color. The goal is to use as few colors as possible. Any graph with maximum vertex-degree Δ trivially requires Δ colors to be properly edge-colored. A celebrated theorem of Vizing [Viz64] says that $\Delta + 1$ colors suffice for any simple graph.[1] There are constructive polynomial time algorithms that achieve a $(\Delta + 1)$-edge-coloring in the classical offline setting [MG92]. These algorithms are likely to be optimal with respect to the number of colors: distinguishing between whether the edge-chromatic number (i.e., the minimum number of colors needed to edge-color a graph) of a simple graph is Δ or $\Delta + 1$ is NP-hard [Hol81].

The edge-coloring problem has several practical applications, including in switch routing [AMSZ03], round-robin tournament scheduling [JURdW16], call scheduling [EJ01], optical networks [RU94], and link scheduling in sensor networks [GDP05]. In many of these applications, such as in switch routing, the underlying graph is built gradually by a sequence of edge insertions and the color assignments need to be done instantly and irrevocably. This is modeled by the *online* edge coloring problem. Due to its restrictions, an online algorithm cannot obtain a $(\Delta+1)$-coloring [BMN92]. Consider, however, the simple greedy algorithm that colors every edge with the first available color that is not already assigned to any of its neighbors. Since each edge can have at most $2\Delta - 2$ adjacent edges, this algorithm achieves a $(2\Delta - 1)$-coloring, i.e., a competitive ratio of $2 - o(1)$ (since the optimum is Δ or $\Delta + 1$). Bar-Noy, Motwani, and Naor [BMN92] showed that no online algorithm can perform better than this greedy algorithm. However, they proved this only for graphs with max-degree $\Delta = O(\log n)$. They conjectured that for $\Delta = \omega(\log n)$, it is

[1]For multigraphs, $3\Delta/2$ colors are sufficient and on some multigraphs necessary [Sha49].

possible to get better bounds, and that, in particular, a $(1 + o(1))\Delta$-coloring is possible. Several works [AMSZ03, BMM12, CPW19, BGW21, SW21, KLS$^+$22, NSW23] have studied online edge coloring with the aim of beating the greedy algorithm and/or resolving the said conjecture. Other variants of the problem have also been studied [FN03, Mik16, FM18]. However, all prior works assume that all graph edges are always stored in the memory along with their colors.

With the ubiquity of big data in the modern world, this assumption often seems fallacious. The graphs that motivate the study of edge coloring, such as communication and internet routing networks, turn out to be large-scale or massive graphs in today's world, making it expensive for servers to store them entirely in their memory. This has led to big graph processing models such as *graph streaming* that, similar to the online model, have sequential access to the graph edges, but can only store a small summary of the input graph so as to solve a problem related to it. There is an immediate barrier for the edge coloring problem in this setting: the output size is as large as the input, and hence an algorithm must use space linear in the input size to present the output as a whole. To remedy this, one can consider the natural extension of the model where the output is also reported in streaming fashion: in the context of edge coloring, think of the algorithm having a limited working memory to store information about both the input graph and the output coloring; it periodically streams or announces the edge colors before deleting them from its memory. This is the so called W-streaming model. Unlike the online model, here we don't need to assign a color to the incoming edge right away, and can defer it to some later time. However, due to the space restriction, we are not able to remember all the previously announced colors. Note that this makes even the greedy $(2\Delta-1)$-coloring algorithm hard (or maybe impossible) to implement in this model.

In this chapter, we aim to get the best of both worlds of the online and the streaming models: the goal is to design *streaming* online algorithms for edge coloring, using significantly less space than the $\widetilde{O}(n\Delta)$ bits needed to store all the edges in a graph of maximum degree Δ. This is motivated by modern practical scenarios that demand immediate color assignment as well as space optimization. We succeed in designing such algorithms and at the same time, the quality of our algorithms is close to optimal: we achieve an $O(1)$-competitive ratio, i.e., a color bound of $O(\Delta)$. Note that no prior work studying edge-coloring in the sublinear-space setting could attain an $O(\Delta)$-coloring W-streaming algorithm, let alone online. For adversarial edge-arrival streams, we get an online $O(\Delta)$-coloring in $O(n\sqrt{\Delta})$ space, significantly reducing the space used by prior online algorithms at the cost of only a constant factor in the number of colors. We can smoothly tradeoff space with colors to get an $O(\Delta t)$-coloring in $\widetilde{O}(n\sqrt{\Delta/t})$ space. This improves upon the state of the art [CL21, ASZZ22] which obtained the same color bound using $\widetilde{O}(n\Delta/t)$ space. Furthermore, for the natural and well-studied settings of vertex-arrival in general graphs and one-sided vertex arrival in bipartite graphs, we can improve the space usage to $O(n \text{ polylog } n)$, i.e., semi-streaming, which is the most popular memory regime for graph streaming problems. Most of our algorithms generalize to multigraphs and can be made deterministic.

5.1.1 Results

We study edge-coloring in the online model with sublinear (i.e., $o(n\Delta)$) memory as well as in the W-streaming model and improve upon the state of the art. These results are summarized in Table 5.1 and Table 5.2. They also mention the state of the art, for comparison.

Arrival	Algorithm	Colors	Space	Graph	Reference
Edge	Randomized	$\left(\frac{e}{e-1}+o(1)\right)\Delta$	$\widetilde{O}(n\Delta)$	Simple	[KLS+22]
Edge	Randomized	$O(\Delta)$	$\widetilde{O}(n\sqrt{\Delta})$	Simple	Theorem 5.5.4
Edge	Deterministic	$(2\Delta-1)t$	$O(n\Delta/t)$	Multigraph	[ASZZ22]
Edge	Deterministic	$\widetilde{O}(\Delta t)$	$\widetilde{O}(n\sqrt{\Delta/t})\star$	Multigraph	Theorem 5.5.8
Vertex	Randomized	$(1.9+o(1))\Delta$	$\widetilde{O}(n\Delta)$	Simple	[SW21]
Vertex	Randomized	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Theorem 5.4.3
Vertex	Deterministic	$2\Delta-1$	$O(n\Delta)$	Multigraph	Greedy folklore
Vertex	Deterministic	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Theorem 5.4.7
One-sided vertex	Randomized	$(1+o(1))\Delta$	$\widetilde{O}(n\Delta)$	Simple	[CPW19]
One-sided vertex	Randomized	1.533Δ	$\widetilde{O}(n\Delta)$	Multigraph	[NSW23]
One-sided vertex	Randomized	5Δ	$\widetilde{O}(n)\star$	Multigraph	Lemma 5.4.1
One-sided vertex	Deterministic	$2\Delta-1$	$O(n\Delta)$	Multigraph	Greedy folklore
One-sided vertex	Deterministic	$O(\Delta)$	$\widetilde{O}(n)\star$	Multigraph	Lemma 5.4.6

Table 5.1: Our results in the online model. Here, $t = O(\Delta)$ is any positive integer. Algorithms marked with a $\star$ require oracle randomness for randomized algorithms and exponential preprocessing time for deterministic.

Algorithm	Colors	Space	Graph	Reference
Randomized	$O(\Delta^2/s)$	$\widetilde{O}(ns)$	Simple	[CL21]
Randomized	$O(\Delta^2/s)$	$\widetilde{O}(n\sqrt{s})$	Simple	Corollary 5.1.1
Randomized	$O(\Delta^2/s)$	$\widetilde{O}(n\sqrt{s})\star$	Multigraph	Theorem 5.5.3
Deterministic	$(1-o(1))\Delta^2/s$	$O(ns)$	Simple	[ASZZ22]
Deterministic	$\widetilde{O}(\Delta^2/s)$	$\widetilde{O}(n\sqrt{s})\star$	Multigraph	Corollary 5.1.2

Table 5.2: Our results in the W-streaming model. Here, $s \leq \Delta/2$ is any positive integer. Results marked with $\star$ require oracle randomness for randomized algorithms and exponential preprocessing time for deterministic.

We consider the problem under (adversarial) edge-arrivals as well as vertex-arrivals. We give an account of our results in each of these models below.

Edge-arrival model. Here we design both online and W-streaming algorithms.

Theorem 5.5.4. *Given any adversarial edge-arrival stream of a simple graph, there is a randomized algorithm for online $O(\Delta)$-edge-coloring using $O(n\sqrt{\Delta \log n})$ bits of space and $\widetilde{O}(n\sqrt{\Delta})$ oracle random bits.*

Previously, there was no sublinear space online algorithm known for $O(\Delta)$-coloring. As observed in Table 5.1, all prior algorithms use $\Omega(n\Delta)$ space in the worst case to achieve a color bound of $O(\Delta)$.

Note that Theorem 5.5.4 immediately implies a randomized W-streaming algorithm with the same space and color bounds. Although immediate, we believe that it is important to note it as a corollary.

Corollary 5.1.1. *Given an adversarially ordered edge stream of any simple graph, there is a randomized W-streaming algorithm for $O(\Delta)$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

The above result improves upon the state of the art algorithms of [CL21, ASZZ22] which, as implied by Table 5.2, only obtain $\omega(\Delta)$-colorings for $o(n\Delta)$ space (the non-trivial memory regime in W-streaming). In fact, we improve upon them by a factor of $\Omega(\sqrt{\Delta})$ in space for $O(\Delta)$-coloring.

We show that the above W-streaming algorithm can be made to work for multigraphs and against adaptive adversaries at the cost of $\widetilde{O}(n\Delta)$ bits of oracle randomness.

Theorem 5.5.3. *There is a randomized W-streaming algorithm for $O(\Delta)$ edge coloring on edge arrival streams for multigraphs which uses $O(n\sqrt{\Delta}(\log(n\Delta))^2)$ bits of space, with error $\leq 1/\operatorname{poly}(n)$ against any adaptive adversary. The algorithm also requires $\widetilde{O}(n\Delta)$ bits of oracle randomness.*

Further, we prove that we can make the above algorithms deterministic at the cost of only a polylogarithmic factor in space. Once again, the online algorithm immediately implies a W-streaming algorithm.

Theorem 5.5.8. *There is a deterministic algorithm for online $O(\Delta(\log \Delta)^2)$ edge coloring in edge arrival streams for multigraphs, using $O(n\sqrt{\Delta}(\log n)^{2.5}(\log \Delta)^3)$ bits of space, and an advice string of $\widetilde{O}(n\sqrt{\Delta})$ bits, which works for all inputs. (By picking a uniformly random advice string, the same algorithm can alternatively be used as a robust algorithm with $1/\operatorname{poly}(n)$ error; the advice can also be verified and computed in exponential time.)*

Corollary 5.1.2. *Given an adversarially ordered edge stream of any multigraph, there is a deterministic W-streaming algorithm for $O(\Delta(\log^2 \Delta))$-edge-coloring using $\widetilde{O}(n\sqrt{\Delta})$ bits of space.*

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

Lemma 5.3.5. *Let f, g be functions from $\mathbb{N} \mapsto \mathbb{N}$. Given a streaming algorithm $\mathcal{A}$ for $g(\Delta)$-coloring over edge arrival streams on multigraphs of max degree Δ, using $f(N, \Delta)$ bits of space, for any positive integer s, there is a streaming algorithm $\mathcal{B}$ for $(g(s\Delta) + s\Delta)$-coloring edge arrival streams for multigraphs of max degree Δ, using $f(N/s, s\Delta) + O(n \log \Delta)$ bits of space.*

For the online model, the above lemma combined with Theorem 5.5.8 immediately gives the tradeoff of $\widetilde{O}(\Delta t)$ colors and $\widetilde{O}(n\sqrt{\Delta/t})$ space for any $t = O(\Delta)$, as claimed in Table 5.1. In other words, combined with Corollary 5.1.1, it implies the W-streaming bounds of $O(\Delta^2/s)$ colors and $O(n\sqrt{s})$ space for any $s = O(\Delta)$, as claimed in Table 5.2. Note that our results match the tradeoff obtained by the state of the art for $t = \Theta(\Delta)$ and $s = O(1)$, and strictly improve upon them for $t = o(\Delta)$ and $s = \omega(1)$.

Vertex-Arrival Model. We now turn to the weaker vertex-arrival model. The online edge-coloring problem has been widely studied in this setting as well (see Section 5.1.2 for a detailed discussion). Our online algorithms obtain significantly better space bounds than in the edge-arrival setting.

Theorem 5.4.3. *There is a randomized online $O(\Delta)$-edge coloring algorithm for vertex arrival streams over multigraphs using $O(n \log(n\Delta/\delta))$ bits of space, with error $\leq \delta$ against any adaptive adversary. It uses $O(n\Delta \log \Delta)$ oracle random bits.*

Thus, at the cost of only a constant factor in the number of colors, we can improve the memory usage from $\widetilde{O}(n\Delta)$ to $\widetilde{O}(n)$ for vertex-arrival streams. Since this algorithm immediately implies a W-streaming algorithm with the same bounds, we see that for vertex-arrival streams, $O(\Delta)$-coloring can be achieved in semi-streaming space, the most popular space regime for graph streaming. Behnezhad et al. [BDH+19] mentioned that "a major open question is whether [the number of colors for W-streaming edge-coloring] can be improved to $O(\Delta)$ while also keeping the memory near-linear in n." Our results answer the question in the affirmative for vertex-arrival streams, which is a widely studied model in the streaming literature as well.

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

Theorem 5.4.7. *There is a deterministic online $O(\Delta)$-edge coloring algorithm for vertex arrival streams over multigraphs using $O(n \log(n\Delta))$ bits of space, using an advice string of length $\widetilde{O}(n)$, which works for all inputs. (By picking a uniformly random advice string, the same algorithm can alternatively be used as a robust algorithm with $1/\operatorname{poly}(n)$ error; the advice can also be computed in exponential time.)*

An interesting special case of the vertex-arrival model is the one-sided vertex-arrival setting for bipartite graphs. Here, the vertices on one side of the bipartite graph are fixed, while the vertices on the other side arrive in a sequence along with their incident edges. A couple of works [CPW19, NSW23] have studied online edge-coloring specifically in this model. We design streaming online algorithms in this model (see Algorithms 5.4.1 and 5.4.2) and use them as building blocks for our algorithms in the more general settings of vertex-arrival and edge-arrival. These algorithms may be of independent interest due to practical applications of the one-sided vertex-arrival model; moreover, the randomized algorithm in this model uses only 5Δ colors (as opposed to our other algorithms, where the hidden constant in $O(\Delta)$ is rather large).

Finally, we present a lower bound on the space requirement of a deterministic online edge-coloring algorithm.

Theorem 5.6.2. *For all $\beta \in (1, 2)$, and integers n, Δ satisfying $\Delta \leq n(2-\beta)/(64\beta)$, every deterministic online streaming algorithm for edge-coloring that uses $\beta\Delta$ colors requires $\Omega((2-\beta)^3 n)$ bits of space. In particular, $(2\Delta - 1)$-edge-coloring requires $\Omega(n/\Delta^3)$ space.*

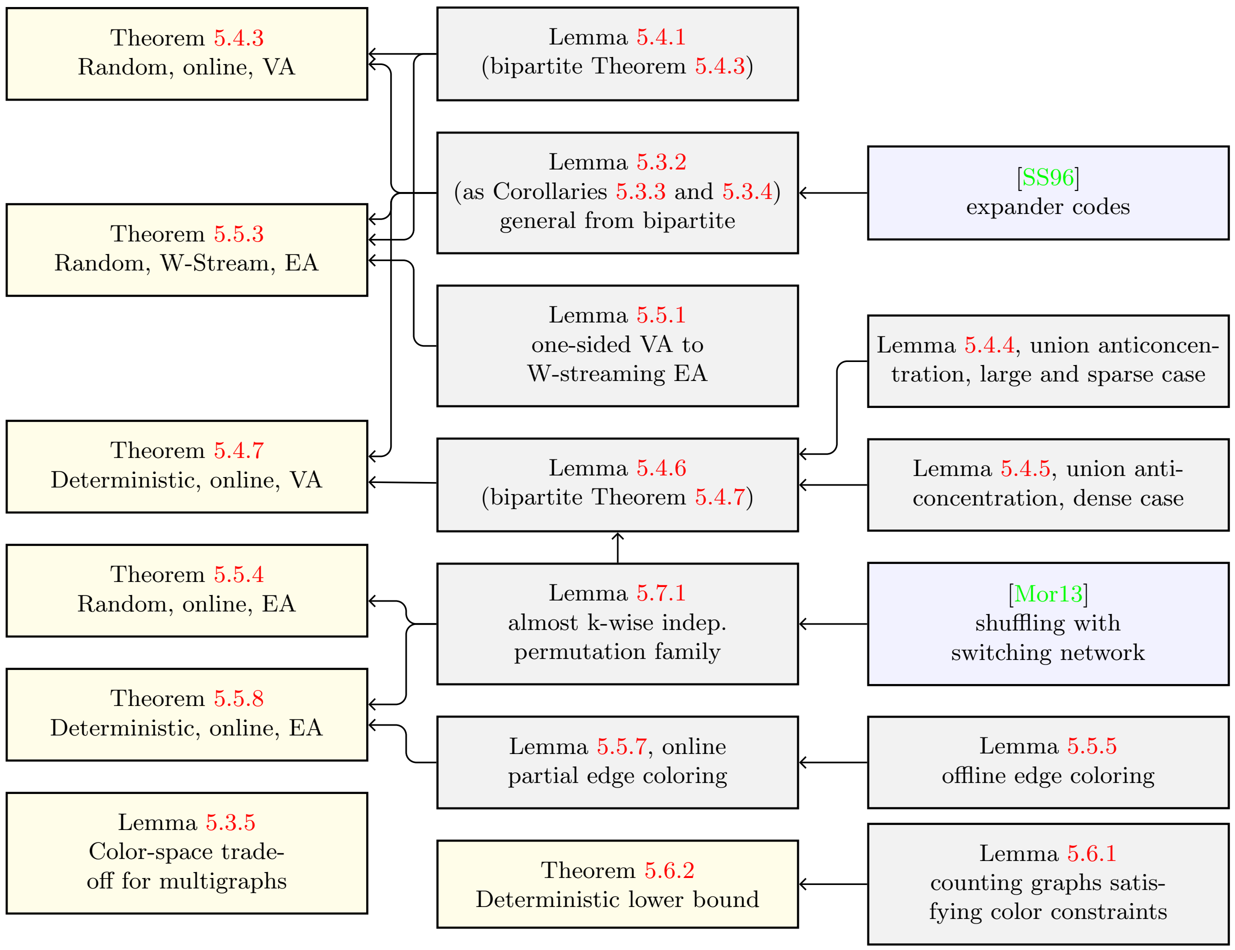
To the best of our knowledge, this is the first non-trivial space lower bound proven for an online edge-coloring algorithm.

An outline of how the several building blocks are put together to obtain the above results is given in Figure 5.1.

5.1.2 Related work

Online model. The edge-coloring problem has a rich literature in the online model [AMSZ03, ASZZ22, BMN92, BMM12, BGW21, CPW19, FM18, FN03, Mik15, Mik16, NSW23, KLS$^+$22, SW21]. The seminal work of Bar-Noy, Motwani, and Naor [BMN92] showed that no online algorithm can do better than the greedy algorithm that obtains a $(2\Delta-1)$-coloring by assigning each edge the first available color that's not already used by any of its adjacent edges. However, this lower bound applies only to graphs with $\Delta = O(\log n)$. They conjectured that for $\Delta = \omega(\log n)$, there exist online $(1 + o(1))\Delta$-coloring algorithms. Although this conjecture remains unresolved, there has been significant progress on it over the years. A number of works [AMSZ03, BMM12, BGW21] considered the problem under *random-order* edge arrivals: Aggarwal et al. [AMSZ03] showed that if $\Delta = \omega(n^2)$, then a $(1+o(1))\Delta$-coloring is possible. For $\Delta = \omega(\log n)$ (the bound in the said conjecture), Bahmani et al. [BMM12] obtained a 1.26Δ-coloring. Bhattacharya et al. [BGW21] then attained the "best of both worlds" by designing a $(1 + o(1))\Delta$-coloring algorithm for $\Delta = \omega(\log n)$, resolving the conjecture for random-order arrivals.

More relevant to our work is the setting of *adversarial-order* edge arrivals. Cohen et al. [CPW19] were the first to make progress on [BMN92]'s conjecture in this setting: they obtained a $(1+o(1))\Delta$-



Theorem 5.4.3 Random, online, VA
Lemma 5.4.1 (bipartite Theorem 5.4.3)
Lemma 5.3.2 (as Corollaries 5.3.3 and 5.3.4) general from bipartite
[SS96] expander codes
Theorem 5.5.3 Random, W-Stream, EA
Lemma 5.5.1 one-sided VA to W-streaming EA
Lemma 5.4.4, union anticoncentration, large and sparse case
Theorem 5.4.7 Deterministic, online, VA
Lemma 5.4.6 (bipartite Theorem 5.4.7)
Lemma 5.4.5, union anti-concentration, dense case
Theorem 5.5.4 Random, online, EA
Lemma 5.7.1 almost k-wise indep. permutation family
[Mor13] shuffling with switching network
Theorem 5.5.8 Deterministic, online, EA
Lemma 5.5.7, online partial edge coloring
Lemma 5.5.5 offline edge coloring
Lemma 5.3.5 Color-space trade-off for multigraphs
Theorem 5.6.2 Deterministic lower bound
Lemma 5.6.1 counting graphs satisfying color constraints

Figure 5.1: Overview of how the results in this paper fit together. Primary results are in yellow; main supporting lemmas in gray; and notable external results in blue. EA = edge arrival, VA = vertex arrival.

coloring for bipartite graphs under one-sided vertex arrivals (i.e., the nodes on one side are fixed, and the nodes on the other side arrive one by one with all incident edges). Their algorithm assumes a priori knowledge of the value of Δ. For unknown Δ, they prove that no online algorithm can achieve better than a $(e/(e-1))\Delta$-coloring, and also complement this result with a $(e/(e-1)+o(1))\Delta$-coloring algorithm for unknown Δ. For bipartite *multigraphs* with one-sided vertex arrivals, Naor et al. [NSW23] very recently prove that 1.533Δ colors suffice, while at least 1.207Δ colors are necessary even for $\Delta = 2$. Saberi and Wajc [SW21] showed that it is possible to beat the greedy algorithm for $\Delta = \omega(\log n)$ under vertex arrivals in general graphs: they design a $(1.9 + o(1))\Delta$-coloring algorithm. Recently, Kulkarni et al. [KLS+22] made the first progress on the said conjecture in the general setting of adversarial edge arrivals: they obtained a $(e/(e-1)+o(1))\Delta$-coloring in this model. Note that the focus of all these works was on resolving [BMN92]'s conjecture without any space limitations. Our focus is on designing streaming online algorithms while staying within a constant factor of the optimal number of colors. The only prior sublinear-space online edge-coloring algorithm we know was given by Ansari et al. [ASZZ22]: a (deterministic) online $2\Delta t$-coloring in $O(n\Delta/t)$ space for any $t \leq \Delta$.

A number of works [FN03, EFKM10, FM18] have studied the variant of the problem where given a fixed number of colors, the goal is to color as many edges as possible. Mikkelsen [Mik15, Mik16] considered online edge-coloring with limited advice for the future.

W-Streaming model. The W-streaming model [DFR06] is a natural extension of the classical streaming model for the study of problems where the output size is very large, possibly larger than our memory. While prior works have considered several graph problems in this model [DFR06, DEMR10, LS11, GSS22], we are only aware of three papers [BDH$^+$19, CL21, ASZZ22] that have studied edge-coloring here. Behnezhad et al. [BDH$^+$19] initiated the study of W-streaming edge-coloring algorithms. They considered the problem for both adversarial-order and random-order streams: using $\widetilde{O}(n)$ bits of working memory, they gave an $O(\Delta^2)$-coloring in the former setting, and a $(2e\Delta)$-coloring in the latter setting. Charikar and Liu [CL21] improved these results: for adversarial-order streams, for any $s = \Omega(\log n)$, they gave an $O(\Delta^2/s)$-coloring algorithm that uses $\widetilde{O}(ns)$ space; and for random-order streams, they gave a $(1 + o(1))\Delta$-coloring algorithm using $\widetilde{O}(n)$ space. Both of the aforementioned algorithms for adversarial-order streams are, however, randomized. Ansari et al. [ASZZ22] gave simple deterministic algorithms achieving the same bounds of $O(\Delta^2/s)$ colors and $\widetilde{O}(ns)$ space. Their algorithm can also be made online at the cost of a factor of 2 in the number of colors. Note that parameterizing our results in Table 5.2 appropriately, our algorithms achieve $O(\Delta^2/s)$-colorings in $\widetilde{O}(n\sqrt{s})$ space, matching the state of the art for $s = O(1)$, and strictly improving upon it for $s = \omega(1)$.

Concurrent work. In an independent and parallel work, Behnezhad and Saneian [BS23] have designed a randomized $\widetilde{O}(n\sqrt{\Delta})$-space W-streaming algorithm for $O(\Delta)$-edge-coloring for edge-arrival streams in simple general graphs. This matches our Corollary 5.1.1. Their result generalizes to give, for any $s \in [\sqrt{\Delta}]$, an $O(\Delta^{1.5}/s)$ coloring algorithm in $\widetilde{O}(ns)$ space, while we achieve an $O(\Delta^2/s)$-coloring in the same space. They also get an $O(\Delta)$-edge-coloring algorithm for vertex-arrival streams using $\widetilde{O}(n)$ space, similar to our Theorem 5.4.3. Note that some of our edge-arrival algorithms have the additional strong feature of being online, while it is not clear if their edge-arrival algorithm can also be implemented in the online setting. In terms of techniques, while both works have some high level ideas in common, e.g., using random offsets/permutations to keep track of colors, or designing a one-sided vertex-arrival algorithm first and building on it to obtain the edge-arrival algorithm, the final algorithms and analyses in the two papers are fairly different.

Another independent work by Chechik, Mukhtar, and Zhang [CMZ23] obtains a randomized W-streaming algorithm that edge-colors an edge-arrival stream on general multi-graphs using $O(\Delta^{1.5}\log\Delta)$ colors in expectation[2], and $\widetilde{O}(n)$ bits of space in expectation. Unlike us, they make no claims in the online model.

5.2 Preliminaries

5.2.1 Notation

The notation $\widetilde{O}(x)$ ignores $\operatorname{poly}(\log(n), \log(\Delta))$ factors in x. $A \sqcup B$ gives the disjoint union of A and B. $\mathbb{S}_t$ is the set of permutations over $[t]$, and for any permutation $\sigma \in \mathbb{S}_t$ and $X \subseteq [t]$, we denote $\sigma[X] := \{\sigma_i : i \in X\}$.

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

Definition 5.2.1. A random permutation σ in $\mathbb{S}_n$ is k-wise independent if, for all distinct $a_1, \ldots, a_k$ in $[n]$, and distinct $b_1, \ldots, b_k$ in $[n]$, we have:

$$\Pr\left[\bigwedge_{i\in[k]} \{\sigma(a_i) = b_i\}\right] = \frac{1}{\prod_{i\in[k]}(n-i+1)}\,.$$

A family of permutations is k-wise independent if the random variable for a uniformly randomly chosen element of that family is k-wise independent.

Per [AL12], while it is not known if there are nontrivial k-wise independent families of permutations for large k and n, one can always construct weighted distributions which have support of size $n^{O(k)}$ and provide k-wise independence.

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

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

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

5.2.2 Models

This chapter will use the following models of presenting edges to an algorithm to be colored. In all cases, the set of vertices for the graph is known in advance. For general graphs, we call the set of

[2]While [CMZ23] does not claim this, one can prove their algorithm uses $O(\Delta^{1.5}\log\Delta)$ colors with $\geq 1 - 1/\operatorname{poly}(n)$ probability.

vertices V; for bipartite graphs, V is partitioned into two disjoint sets, which we typically call A and B. Let G be the (multi-) graph formed by taking the union of all edges in the stream.

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

Definition 5.2.2. With an *edge arrival stream*, the algorithm is given a sequence of edges in the graph. Each edge is provided as an ordered pair $\{x, y\}$ of vertices in V. In this paper, online algorithms processing edge arrival streams will implement a method $\textsc{Process}(\{x, y\})$ which returns the color assigned to the edge. For example, see Algorithm 5.2.1, an implementation of the greedy edge coloring algorithm using $O(n\Delta)$ bits of space. W-streaming algorithms may assign the color for an edge at any time, although all edges must be given a color at the end of the stream.

Definition 5.2.3. In a *vertex arrival stream*, the algorithm is given a sequence of (vertex, edge-set) pairs (v, M_v), where the edge (multi-) set M_v contains all edges from v to vertices that have been seen earlier in the stream. Online algorithms should report colors for all edges in M_v when (v, M_v) is processed.

A *one-sided vertex arrival stream* on a bipartite graph with parts A, B is like a vertex arrival stream, if the vertices for one part (B) were all presented first, and then all the (vertex, edge-set) pairs for the other part (A) are given. For one-sided vertex arrival, we assume that the algorithm knows parts A and B in advance, and receives the (vertex, edge-set) pairs for B. The stream consists of pairs (v, M_v), where each $v \in A$, and M_v contains all edges from v to B.

Recall from Chapter 2 that an algorithm is said to be robust if it works with $\geq 1 - \delta$ probability even when its input streams are adaptively generated. By "adaptively generated", we mean that the input is produced by an adaptive adversary that sees all outputs of the online (or W-streaming) algorithm, and repeatedly chooses the next element of the stream based on what the algorithm has output so far.

5.3 Algorithm transformations/reductions

5.3.1 From bipartite graphs to general graphs

We show that it is essentially enough to consider bipartite graphs. Suppose that we can partition a general graph into $O(\log n)$ bipartite graphs, each of which has max-degree roughly $\Delta / \log n$,

Algorithm 5.2.1 A greedy $2\Delta - 1$ online edge-coloring algorithm using $O(n\Delta)$ bits of space

```
   Input: Stream of edges in an $n$-vertex graph $G = (V, E)$

   Initialize:
1: for $v \in V$ do
2:     $U_v \leftarrow \emptyset$ is a subset of $[2\Delta - 1]$

   Process(edge $\{x, y\}$) $\rightarrow$ color
3: Let $c$ be arbitrary color in $[2\Delta - 1] \setminus U_x \setminus U_y$
4: Add $c$ to $U_x$ and to $U_y$
5: return color $c$
```

where Δ is the max-degree of the original graph. Then, if we run our algorithm on these bipartite graphs with disjoint palettes, we use colors roughly proportional to Δ. It is known (see [CL21] or for a similar result, [SW21, Lemma 2.1]) that such a partition can be done in a randomized way, incurring a multiplicative overhead of just $1 + o(1)$ in the number of colors. We show that if we are willing to tolerate an $O(1)$ blowup in the number of colors, then this partition can be done deterministically. Since such a primitive is used in multiple edge-coloring algorithms, this deterministic version might be of independent interest. One advantage of this version is that it works against adaptive adversaries, unlike the randomized version which can be shown to be breakable by such an adversary.

We construct this partition using appropriate binary codes for the vertices: the codes are of length $O(\log n)$ and we have a bipartite graph corresponding to each bit, where the bipartition is given by whether the bit is 0 or 1. Now, we need to ensure that (a) every edge goes to "some" bipartition, and (b) the max-degree of a single bipartite graph is not much higher than $\Delta / \log n$. This can be done using codes with constant rate and relative distance, like the expander codes described by [SS96]. Now we can focus on getting $O(\Delta)$-colorings for bipartite graphs, which would give us asymptotically same number of colors for general graphs.

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

First, we will need the following lemma:

Corollary 5.3.1 (Practical high-rate-distance-product binary codes, via [SS96])**.** *There exists a constant t_0 so that, for all integers t, there exists a binary code of dimension $\geq t$, length $t' \leq 64 \max(t, t_0)$, and distance $\geq \frac{1}{400} t'$. The code can be implemented with $O(poly(t))$ initial setup time and space, and $O(t^2)$ encoding time.*

Proof of Corollary 5.3.1. [SS96] Theorem 19 proves that there exists a polynomial-time constructable family of expander graphs with constant rate and relative distance, but does not prove a usable guarantee on the lengths of codes in this family. Fortunately, only a bit more work needs to be done to do this.

Let $\varepsilon = \sqrt{2}/20$. By the Gilbert-Varshamov bound, there is a constant c_0 so that for all $y \geq c_0$, there exists a linear code B_y of length y, rate $\geq 1 - H(\varepsilon)$ and distance $\geq \varepsilon y$. Let p be a prime equivalent to $1 \pmod 4$ for which $p > c_0$ and $2\sqrt{p}/(p+1) \leq (1 - 1/\sqrt{2})\varepsilon$. Say q is a prime which is $> p$ and equivalent to $1 \pmod 4$. By [LPS88], if p is a quadratic residue of q, then there is an expander graph $G_{p,q}$ on $(q^3 - q)/2$ vertex set $V_{p,q} = PSL(2, \mathbb{Z}/q\mathbb{Z})$; and if p is not a quadratic residue of q, then there is an expander graph $G_{p,q}$ on the $q^3 - q$ vertex set $V_{p,q} = PGL(2, \mathbb{Z}/q\mathbb{Z})$. In both cases, $G_{p,q}$ is regular of degree $p + 1$ and has second eigenvalue $\leq 2\sqrt{p}$. By [SS96], one can construct a code $C(B_p, G_{p,q})$ which has length $(p+1)|V_{p,q}|/2$, and by [SS96, Lemma 15], rate $\geq 1 - 2H(\varepsilon) = 0.2628\ldots \geq 1/4$, and relative distance

$$\geq \left(\frac{\varepsilon - \frac{2\sqrt{p}}{p+1}}{1 - \varepsilon - \frac{2\sqrt{p}}{p+1}} \right) \geq \frac{\varepsilon^2}{2} \geq \frac{1}{400}\,.$$

By [Mor93], for any $r \geq 25$, there exists at least one prime $\equiv 1 \pmod 4$ in the interval $[r, 2r]$.

Now, let $t_0 = \lceil (p+1)^4/16 \rceil$. For all $t \geq t_0$, let q be the least prime $\equiv 1 \pmod 4$ which is $\geq (16t/(p+1))^{1/3}$. Then the dimension of the code $C(B_p, G_{p,q})$ is

$$\geq \frac{1}{4}\frac{(p+1)|V_{p,q}|}{2} \geq \frac{1}{8}(p+1)\frac{q^3 - q}{2} \geq \frac{p+1}{16}q^3 \geq t\,.$$

On the other hand, the length of the code is:

$$\leq \frac{(p+1)|V_{p,q}|}{2} \leq \frac{1}{2}(p+1)(q^3 - q) \leq \frac{1}{2}(p+1)q^3 \leq \frac{1}{2}(p+1)\left(2\left(\frac{16t}{p+1}\right)^{1/3}\right)^3 \leq 64t\,.$$

The expander graph $G_{p,q}$ is computable in $O(\text{poly}(p,q)) = O(\text{poly}\, t)$ time, and the expander code takes $O(t^2)$ time to encode. $\square$

Lemma 5.3.2 (Deterministic general-to-bipartite partitioning)**.** *For sufficiently large n, there is a set of $t = 4\lceil \log n \rceil$ bipartite graphs $F_1, \ldots, F_t$, and an online algorithm $\mathcal{A}$, which processes a stream of edges and assigns each edge to one of the t graphs. The algorithm ensures that at each time, for each vertex v, $\deg_{F_i}(v) \leq \frac{300}{\log n}\deg_G(x) + 1$. It uses $O(n(\log n)(\log \Delta))$ bits of space.*

Proof of Lemma 5.3.2. We claim Algorithm 5.3.1 works for sufficiently large n.

It is clear that at each point in time, for all $v \in V$ and $i \in [t]$, the algorithm will have $\deg_{F_i}(v)$ be the number of edges assigned to F_i incident on v.

Line 8 of the algorithm ensures that before edge $\{x, y\}$ is assigned, $\deg_{F_i}(x) \leq 1200\deg(x)/t$.

Algorithm 5.3.1 Algorithm to partition general graph edges into bipartite graphs

```
    Input: Stream of edges in an $n$-vertex graph $G = (V, E)$

    Initialize:
 1: for $v \in V$ do
 2:     $\deg_{F_i}(v) \leftarrow 0$
 3: Setup binary code $\mathcal{C}$ of length $t := 4\lceil \log n \rceil$ from Corollary 5.3.1
 4: for $i \in [t]$ do
 5:     Let $F_i$ be the bipartition with parts $A_i = \{v \in V : \mathcal{C}(v)_i = 0\}$ and $B_i = \{v \in V : \mathcal{C}(v)_i = 1\}$

    Process(edge $\{x, y\}$)
 6: Let $\deg(x) = \sum_{i \in [t]} \deg_{F_i}(x)$ and $\deg(y) = \sum_{i \in [t]} \deg_{F_i}(y)$.
 7: for $i \in [t]$ do
 8:     if $\mathcal{C}(x)_i \neq \mathcal{C}(y)_i$ and $\deg_{F_i}(x) \le 1200\deg(x)/t$ and $\deg_{F_i}(y) \le 1200\deg(y)/t$ then
 9:         Increase $\deg_{F_i}(x)$ and $\deg_{F_i}(y)$ by 1
10:         Assign edge $\{x, y\}$ to $F_i$
11:         return
12: unreachable
```

Consequently, after the edge is assigned, $\deg_{F_i}(x) \le 1200\deg(x)/t + 1 \le 300\deg(x)/\log(n) + 1$. Similarly, we will have $\deg_{F_i}(y) \le 300\deg(y)/\log(n) + 1$.

It remains to prove that the algorithm will always assign an edge, and that Line 12 is never reached. When processing edge an $\{x, y\}$, define $B_v := \{i \in [t] : \deg_{F_i}(v) > 1200\deg(v)/t$. By Markov's inequality, since $\sum_{i \in [t]} \deg_{F_i}(v) = \deg(v)$, $|B_v| \le t/1200$. Because the code $\mathcal{C}$ has minimum distance $t/400$, the set $K = \{i : \mathcal{C}(x)_i \neq \mathcal{C}(y)_i\}$ has size $\ge t/400$; and the set of $i \in [t]$ for which Line 8 passes has size $|K \setminus B_x \setminus B_y| \ge t/400 - t/1200 - t/1200 = t/1200$, and hence is nonempty. $\square$

Using Lemma 5.3.2 to route edges to $O(\log n)$ instances of an algorithm that handles bipartite graphs gives the following corollaries, for edge arrival and vertex arrival streams, respectively.

Corollary 5.3.3 (Of Lemma 5.3.2)**.** *Say $f : \mathbb{N} \mapsto \mathbb{N}$ is a function for which $f(x)/x$ is monotonically increasing. Then given an algorithm $\mathcal{A}$ for edge coloring with $f(\Delta)$ colors on edge arrival streams over bipartite graphs of max degree Δ, which uses $g(n, \Delta)$ bits of space, one can implement an algorithm $\mathcal{B}$ for edge coloring with $f(O(\Delta))$ colors on general graphs, using $O((g(n, \Delta) + n\log\Delta)\log n)$ space.*

Proof of Corollary 5.3.3. If $\Delta \le t_n$, implement $\mathcal{B}$ using the greedy algorithm (Algorithm 5.2.1), which will use $2\Delta - 1 = f(O(\Delta))$ colors and $O(n\Delta) = O(n\log n)$ space. Otherwise, do the following:

Let D be an instance of Algorithm 5.3.1 from Lemma 5.3.2, with bipartite graphs $F_1, \ldots, F_{t_n}$. Set up t_n instances $A_1, \ldots, A_{t_n}$ of $\mathcal{A}$, where A_i is configured to handle edge arrival streams of maximum degree $\hat{\Delta} = \lfloor 1200\Delta/t_n \rfloor + 1$ on the bipartite graph F_i. Ensure that each A_i outputs colors in $[f(\hat{\Delta})]$. Whenever an edge e arrives, use D to determine to which bipartite graph F_i the edge e should be assigned, and then assign it the color $(i, A_i.\textsc{Process}(e)) \in [t_n] \times \left[f(\hat{\Delta})\right]$. The total space used will be $O(g(n, \Delta) \log n)$ for the instances of $\mathcal{A}$ and $O(n \log \Delta \log n)$ for D. The number of colors used will be:

$$\begin{aligned} t_n \cdot f(\hat{\Delta}) &\leq t_n f(\lfloor 1200\Delta/t_n \rfloor + 1) \\ &\leq t_n f(\lfloor 1201\Delta/t_n \rfloor) \qquad \text{since } \Delta \geq t_n \\ &\leq t_n \frac{\lfloor 1201\Delta/t_n \rfloor}{2000\Delta} f(2000\Delta) \qquad \text{since } x \leq y \text{ implies } f(x) \leq \frac{x}{y} f(y) \\ &\leq f(2000\Delta) \,. \end{aligned}$$

□

Combining the previous corollary with that fact that one can convert an algorithm for one-sided vertex arrival streams on bipartite graphs to general ("two-sided") vertex arrival streams on bipartite graphs, only doubling the number of colors used, gives the following:

Corollary 5.3.4 (Of Lemma 5.3.2)**.** *Say $f : \mathbb{N} \mapsto \mathbb{N}$ is a function for which $f(x)/x$ is monotonically increasing. Then given an algorithm $\mathcal{A}$ for edge coloring with $f(\Delta)$ colors on one-sided vertex arrival streams over bipartite graphs of max degree Δ with $\leq n$ fixed vertices, which uses $g(n, \Delta)$ bits of space, one can implement an algorithm $\mathcal{B}$ for edge coloring under vertex arrivals of general graphs using $f(O(\Delta))$ colors and $O((g(n, \Delta) + n \log \Delta) \log n)$ space.*

Proof of Corollary 5.3.4. The proof that this works is similar to that of Corollary 5.3.3. As before, the case $\Delta \leq t_n$ is addressed using the greedy algorithm. Set up an instance D of Algorithm 5.3.1 from Lemma 5.3.2. For each bipartite graph F_i for $i \in [t_n]$ from Lemma 5.3.2, let $A_i \sqcup B_i$ give the two parts of the graph. Instantiate an instance of $\mathcal{A}$ called X_i with fixed vertex set A_i, and one called Y_i with fixed vertex set B_i; both instances should handle vertex degrees up to $\hat{\Delta} := \lfloor 1200\Delta/t_n \rfloor + 1$ and output colors in $[\hat{\Delta}]$. Whenever a vertex v and multi-set of edges M_v arrives, use D to assign each edge in M_v to some $i \in [t_n]$, processing the edges in arbitrary order. Let $P : M_v \to [t_n]$ be this assignment. Then for each $i \in [t_n]$, if $v \in A_i$, send input $(v, \{e \in M_v : P(e) = i\})$ to instance Y_i; otherwise, with $v \in B_i$, send input $(v, \{e \in M_v : P(e) = i\})$ to instance X_i. Let $\chi : M_v \to [f(\hat{\Delta})]$ give the colors assigned by the instances of $\mathcal{A}$. For each $e \in M_v$, assign color $(P(e), \mathbb{1}_{v \in A_{P(e)}}, \chi(e)) \in [t_n] \times \{0, 1\} \times [f(\hat{\Delta})]$; this ensures that no two edges incident on v receive the same color.

The bounds on the space and number of colors are performed similarly to Corollary 5.3.3. □

5.3.2 Color-space tradeoff

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

Lemma 5.3.5. *Let f, g be functions from $\mathbb{N} \mapsto \mathbb{N}$. Given a streaming algorithm $\mathcal{A}$ for $g(\Delta)$-coloring over edge arrival streams on multigraphs of max degree Δ, using $f(N, \Delta)$ bits of space, for any positive integer s, there is a streaming algorithm $\mathcal{B}$ for $(g(s\Delta) + s\Delta)$-coloring edge arrival streams for multigraphs of max degree Δ, using $f(N/s, s\Delta) + O(n \log \Delta)$ bits of space.*

Proof of Lemma 5.3.5. Pseudocode for algorithm $\mathcal{B}$ is given by Algorithm 5.3.2.

Algorithm 5.3.2 Adapting an edge coloring algorithm $\mathcal{A}$ to trade colors for space, with parameter s

```
Input: Stream of edge arrivals for $n$-vertex graph $G = (V, E)$
Assume $V = [n]$

Initialize:
Let $\chi : K_s \mapsto [s]$ give an $s$-edge coloring of $K_s$.[3]
1: $A \leftarrow$ instance of $\mathcal{A}(\lceil n/s \rceil, \Delta s)$.
2: for $v \in [n]$ do
3:     $d_v \leftarrow 0$
Process(edge $\{x, y\}$) $\to$ color
4: $d_x \leftarrow d_x + 1$
5: $d_y \leftarrow d_y + 1$
6: if $\lceil x/s \rceil = \lceil y/s \rceil$ then
7:     Let $c \leftarrow \Delta \cdot (\chi(\{x \bmod s, y \bmod s\}) - 1) + d_{\min(x,y)}$
8:     return color $(0, c)$
9: Let $c \leftarrow A.\textsc{Process}(\lceil x/s \rceil, \lceil y/s \rceil)$
10: return color $(1, c)$
```

This algorithm partitions the set of all vertices into sets $S_1, \ldots, S_{\lceil n/\delta \rceil}$, where set S_i contains the s vertices $\{s(i-1)+1, \ldots, si-1, si\}$. It provides the nested algorithm instance A with the (non-loop) edges in the graph H formed by contracting these sets. Edges entirely inside one of the S_i are colored using a separate set of Δs colors.

As the total number of edges incident on a set of s vertices in G is $\le \Delta s$, the maximum degree of H will also be $\le \Delta s$. Since instance A is guaranteed to correctly edge color all multigraphs on

$[\lceil n/s \rceil]$ of maximum degree $\leq \Delta s$, no two edges adjacent to a vertex in H will be assigned the same color. Consequently, the edges from each individual vertex $v \in S_i$ to vertices outside $S - I$ will all be given different colors.

Consider one of the vertex sets S_i; a given edge $\{x, y\}$ with $x, y \in S_i$ will be assigned a color which, due to the use of χ to partition edges, will differ from the colors assigned to all other edge types between vertices in S_i; and if the edge $\{x, y\}$ was processed in the past, this time will assign a different color since $d_{\min(x,y)}$ has been increased since then.

The algorithm will require $f(\lceil N/s \rceil, \Delta s)$ bits of space to store A, and $n \log \Delta$ bits of state to keep track of all vertex degrees. The total number of colors used will be $g(s\Delta) + s\Delta$; if $g(x) = O(x)$, this will be $O(s\Delta)$.

□

5.4 Edge coloring on vertex arrival streams

5.4.1 Randomized online algorithm for vertex arrivals

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

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

[3]While it is possible to implement this more efficiently, this function can also be evaluated by running the Misra-Gries algorithm [MG92] in $O(s^3)$ time.

expectation. Since we only store a counter for each vertex whose value can go up to $O(\Delta)$, the space usage is $\widetilde{O}(n)$. Thanks to the reductions discussed above, we can extend this to a semi-streaming $O(\Delta)$-coloring for the general vertex arrival case, even for general graphs.

Lemma 5.4.1. *Theorem 5.4.3 holds for one-sided vertex arrival streams on bipartite graphs.*

Proof of Lemma 5.4.1. Consider Algorithm 5.4.1. This algorithm will have the required properties if $\Delta \geq 6 \ln \frac{n}{\delta}$; if Δ is smaller, convert the vertex arrival stream to an edge arrival stream and pass it to Algorithm 5.2.1, which guarantees a $2\Delta - 1$ coloring of the graph using $O(n\Delta) = O(n \ln \frac{n}{\delta})$ bits of space.

Algorithm 5.4.1 Randomized algorithm for 5Δ edge coloring for one sided vertex arrival bipartite streams

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

    Initialize:
 1: Let $C = 5\Delta$.
 2: for $v \in B$ do
 3:     Let $\sigma_v$ be a uniformly random permutation over $[C]$    ▷ constructed on demand from random oracle bits.
 4:     $h_v \leftarrow 1$.
    Process(vertex $x$ with multiset $M_v$ of edges to $B$)
 5: Let $S \leftarrow \emptyset$                ▷ Set of colors $M_x$ will have used so far
 6: for $e = \{x, y\}$ in $M_x$, in arbitrary order do
 7:     while $h_y \leq C \wedge \sigma_y[h_y] \in S$ do
 8:         $h_y \leftarrow h_y + 1$
 9:     if $h_y > C$ then
10:         abort
11:     Assign color $\sigma_y[h_y]$ to $e$
12:     $S \leftarrow S \cup \{\sigma_y[h_y]\}$
13:     $h_y \leftarrow h_y + 1$
```

This algorithm will never assign the same color to any pair of edges adjacent to the same vertex; at worst, it will abort. The condition on Line 7 ensures that when a vertex x is processed, no two edges will be assigned the same color. On the other hand, after Line 11 assigns a color to an edge, Line 13 increases h_y; because σ_v is a permutation, this prevents the algorithm from ever assigning the same color twice to edges incident on some vertex y in B.

For the rest of the proof, we will argue that the algorithm never aborts; equivalently, that $h_y \leq C$ always holds for all $y \in B$. In fact, we shall prove the stronger claim, that $h_y \leq C - 2\Delta$ holds with probability $\geq 1 - \delta/n$ for each individual $y \in B$. Consider a specific vertex $y \in B$. For each $i \in [\Delta]$, let $V_{y,i}$ be the random variable counting the number of times that the loop starting at Line 7 ran, when the ith edge adjacent to y was processed. If there was no ith edge (or the algorithm already aborted), we set $V_{y,i} = 0$; then at the end of the stream, we will have $h_y \leq \Delta + \sum_{i \in \Delta} V_{y,i}$.

We now consider the distribution of $V_{y,i}$, conditioned on both the value of the variable S at the time the ith edge was processed, and on the parts of the permutation σ_y which the algorithm has read so far, $\sigma_y[1..h_y - 1]$.

$$\begin{aligned}
\Pr[V_{y,i} \geq k \mid S, \sigma_y[1..h_y - 1]] &= \Pr[\sigma_y[h_y, \ldots, h_y + k - 1] \subseteq S \mid S, \sigma_y[1..h_y - 1]] \\
&= \binom{|S \cap \sigma_y[h_y, ..., C]|}{k} \Big/ \binom{C - h_y - 1}{k} \\
&\leq \binom{\Delta}{k} \Big/ \binom{2\Delta}{k} \qquad\qquad \text{since } |S| \leq \Delta,\ h_y \leq C - 2\Delta \\
&\leq \frac{\Delta \cdot (\Delta - 1) \cdots (\Delta - k + 1)}{2\Delta \cdot (2\Delta - 1) \cdots (2\Delta - k + 1)} \leq \frac{1}{2^k}\,.
\end{aligned}$$

Since this bound holds for all values of S and all $\sigma_y[1..h_y - 1]$, in particular we have

$$\Pr[V_{y,i} \geq k | (V_{y,j})_{j<i}] = \mathbb{E}_{S,\, \sigma_y[1..h_y\, -\, 1] \text{ compat with } (V_{y,j})_{j<i}} \Pr[V_{y,i} \geq k | S, \sigma_y[1..h_y - 1]] \leq \frac{1}{2^k}\,.$$

By Lemma 5.4.2, for $e^t \in [1, 2)$, we have $\mathbb{E}[e^{tV_{y,i}} \mid (V_{y,j})_{j<i}] \leq 1/(2 - e^t)$, and $\mathbb{E}[V_{y,i} \mid (V_{y,j})_{j<i}] \leq 1$.

By a slight variation on the Chernoff bound:

$$\begin{aligned}
\Pr[\sum_{i=1}^{\Delta} V_{y,i} \geq 2\Delta] &\leq \inf_{t \geq 0} \Pr\left[\prod_{i=1}^{\Delta} e^{tV_{y,i}} \geq e^{2t\Delta}\right] \\
&\leq \inf_{t \geq 0} \frac{1}{e^{2t\Delta}} \mathbb{E}[e^{tV_{y,1}} \cdots \mathbb{E}\left[e^{tV_{y,\Delta}} \mid (V_{y,j})_{j<\Delta}\right] \cdots] \\
&\leq \min_{t : e^t \in [1,2)} \frac{1}{e^{2t\Delta}} \left(\frac{1}{2 - e^t}\right)^{\Delta} = \left(\min_{t : e^t \in [1,2)} \frac{1}{e^{2t}(2 - e^t)}\right)^{\Delta} \\
&= \left(\frac{1}{\max_{x \in [1,2)} x^2(2 - x)}\right)^{\Delta} = \left(\frac{27}{32}\right)^{\Delta} \leq \exp(-\Delta/6)\,.
\end{aligned}$$

Since $h_v \leq \Delta + \sum_{i \in \Delta} V_{y,i}$, this implies that

$$\Pr[h_v \geq C] \leq \Pr[h_v \geq C - 2\Delta] = \Pr[h_v \geq 3\Delta] \leq \Pr[\sum_{i=1}^{\Delta} V_{y,i} \geq 2\Delta] \leq \exp(-\Delta/6)\,.$$

Thus, by a union bound, the probability that *any* vertices $v \in B$ will have $h_v > C$ at the end of the algorithm will be $\leq n \exp(-\Delta/6)$. In particular, if $\Delta \geq 6 \ln(n/\delta)$, the algorithm will the guaranteed to abort with probability $\leq \delta$. □

Lemma 5.4.2. *Let W be a nonnegative integral random variable where, for all $k \in \mathbb{N}$, $\Pr[W \geq k] \leq 1/2^k$. Then $\mathbb{E}[W] \leq 1$; and furthermore, for all t for which $e^t \in [1, 2)$:*

$$\mathbb{E}[e^{tW}] \leq \frac{1}{2 - e^t}\,.$$

Proof of Lemma 5.4.2. First,

$$\mathbb{E}[W] = \sum_{k=0}^{\infty} \Pr[W \geq k] \leq \sum_{k=0} 1/2^{k+1} = 1\,.$$

Next,

$$\mathbb{E}[e^{tW}] = \sum_{k=0}^{\infty} e^{tk} \Pr[W = k] \leq \sum_{k=0}^{\infty} e^{tk} \frac{1}{2^{k+1}} = \frac{1}{2} \sum_{k=0}^{\infty} \left(\frac{e^t}{2}\right)^k = \frac{1}{2} \cdot \frac{1}{1 - \frac{1}{2}e^t} = \frac{1}{2 - e^t}\,.$$

The inequality step follows because:

$$\begin{aligned}
\sum_{k=0}^{\infty} e^{tk} \left(\frac{1}{2^{k+1}} - \Pr[W = k]\right) &= \sum_{k=0}^{\infty} e^{tk} \left((\frac{1}{2^k} - \frac{1}{2^{k+1}}) - (\Pr[W \geq k] - \Pr[W \geq k-1]) \right) \\
&= \sum_{k=0}^{\infty} e^{tk} \left(\frac{1}{2^k} - \Pr[W \geq k]\right) - \sum_{k=0}^{\infty} e^{tk} \left(\frac{1}{2^{k+1}} - \Pr[W \geq k-1]\right) \\
&= \sum_{k=0}^{\infty} e^{tk} \left(\frac{1}{2^k} - \Pr[W \geq k]\right) - \sum_{k=1}^{\infty} e^{t(k-1)} \left(\frac{1}{2^k} - \Pr[W \geq k]\right) \\
&= \left(\frac{1}{2^0} - \Pr[W \geq 0]\right) + \sum_{k=1}^{\infty} (e^t - e^{t(k-1)}) \left(\frac{1}{2^k} - \Pr[W \geq k]\right) \\
&= 0 + \sum_{k=1}^{\infty} e^{t(k-1)} (e^t - 1) \left(\frac{1}{2^k} - \Pr[W \geq k]\right) \geq 0\,.
\end{aligned}$$

□

Observe that Corollary 5.3.4 and Lemma 5.4.1 collectively imply Theorem 5.4.3.

Theorem 5.4.3. *There is a randomized online $O(\Delta)$-edge coloring algorithm for vertex arrival streams over multigraphs using $O(n \log(n\Delta/\delta))$ bits of space, with error $\leq \delta$ against any adaptive adversary. It uses $O(n\Delta \log \Delta)$ oracle random bits.*

5.4.2 Deterministic online algorithm for vertex arrivals

Let us now turn to the deterministic version of the problem. We describe a deterministic algorithm which uses a advice string, and show that if the advice string is chosen randomly, the algorithm will with high probability work on every input stream. This algorithm can also be used as a randomized algorithm, if we choose the advice uniformly at random, and we use a somewhat complicated analysis to show that this can be achieved using a (random) advice string of length only $\widetilde{O}(n)$.

When a vertex a and its neighboring edges are processed, the randomized online vertex arrival algorithm is greedily finding a coloring (i.e, a matching between edges incident on a, and the next few available colors for the "fixed" vertices at the endpoints of these edges.) While this greedy selection works acceptably in the *average* case, it does not provide strong worst-case guarantees: with

$\Omega(1/\operatorname{poly}(n))$ probability, *some* vertex adjacent to a will increase its counter by $\Omega(\log n)$, thereby marking ("consuming") $\Omega(\log n)$ colors from its random permutation as unavailable. Consequently, there is small but nonzero risk that a vertex will use up too many colors from its random permutation and run out. In contrast, our deterministic algorithm for online vertex arrival edge coloring is designed to ensure that, for any fixed vertex, the *amortized* number of colors consumed per edge incident on a fixed vertex is always bounded by a constant.

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

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

A notable problem is that storing each permutation would require $\widetilde{O}(\Delta)$ bits each.[4] To avoid this, we select the permutations from small, almost-k-wise independent families of permutations, instead. This works, but requires a more careful analysis to prove correct, and a large fraction of our proof is spent dealing with the interaction of this and support for multigraphs.

We now prove two somewhat technical anti-concentration lemmas; intuitively, they say that,

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

even after removing some large fixed set, it is unlikely that the union of many independent random subsets will be much smaller than expected. They will be used by the proof of Lemma 5.4.6.

Lemma 5.4.4. *Let C, t, w be integers, with $C \geq t \geq 512$, and $8|C$. Let $\varepsilon \leq C^{-t-1}$. Say that $F_1, \ldots, F_w$ are subsets of $[C]$, and define $s_i = |F_i|$ for all $i \in [w]$. We furthermore require $\min_{i\in[w]} s_i \geq \frac{1}{2}t$, and $\min_{i\in[w]} s_i \geq \frac{1}{2}\max_{i\in[w]} s_i$. Let $X \subseteq [C]$ satisfy $|X| \leq \frac{1}{8}C$, and let $\sum_{i\in[w]} s_i \leq \frac{1}{2}C$. Then if $\sigma_1, \ldots, \sigma_w$ are (ε, t)-wise independent random permutations over $[C]$,*

$$\Pr\left[\left|\bigcup_{i\in[w]} \sigma_i[F_i] \setminus X\right| < \frac{1}{8}\sum_{i\in[w]} s_i\right] \leq \exp(-\frac{1}{2^9}tw)\,.$$

Proof of Lemma 5.4.4. Since the $(\sigma_i)_{i\in[w]}$ are (ε, t)-wise independent, in particular we have for any $i \in [w]$,$j \in [C]$, that $\Pr[j \in \sigma_i[F_i]] \leq s_i/C + \varepsilon$, and for any $Q \subseteq [C]$ with $|Q| \leq t$, that

$$\Pr[Q \subseteq \sigma_i[F_i]] = \frac{s_i \cdot (s_i - 1)\cdots(s_i - |Q| + 1)}{C \cdot (C-1)\cdots(C - |Q| + 1)} + \varepsilon \leq \left(\frac{s_i}{C}\right)^{|Q|}\,. \tag{5.1}$$

Let U_i be a random subset of $[C]$ in which each element in C is included independently with probability s_i/C. Eq. 5.1 thus implies $\Pr[Q \subseteq \sigma_i[F_i]] \leq \Pr[Q \subseteq U_i]$. Now, for any fixed set $H \subseteq [C]$, let $Y_{H,i} := |\sigma_i[F_i] \cap H|$, and $W_{H,i} = |U_i \cap H|$. Then $\mathbb{E}[Y_{H,i}] \leq s_i|H|/C + \varepsilon C = \mathbb{E}[W_{H,i}] + \varepsilon C$, and as a consequence of Eq. 5.1, we have for all $k \leq t$, that $\mathbb{E}[Y_{H,i}^k] \leq \mathbb{E}[W_{H,i}^k] + \varepsilon C$. This lets us bound the moment generating function of $Y_{H,i}$, for nonnegative z:

$$\begin{aligned}
\mathbb{E}e^{zY_{H,i}} &\leq \sum_{k=0}^{\infty} \frac{1}{k!}(\mathbb{E}(zY_{H,i})^k) \\
&\leq \sum_{k=0}^{t} \frac{1}{k!}(\mathbb{E}(zY_{H,i})^k) + \sum_{k=t+1}^{\infty} \frac{1}{k!}(\mathbb{E}(zY_{H,i})^k) \\
&\leq \sum_{k=0}^{t} \frac{1}{k!}(\mathbb{E}(zW_{H,i})^k) + \sum_{k=t+1}^{\infty} \frac{1}{(k-t-1)!(t+1)!}((zs_i)^k) + \varepsilon tC \\
&\leq \mathbb{E}e^{zW_{H,i}} + \frac{(zs_i)^{t+1}}{(t+1)!}e^{zs_i} + \varepsilon tC \\
&\leq \exp\left(\frac{s_i|H|}{C}(e^z - 1)\right) + \frac{(zs_i)^{t+1}}{(t+1)!}e^{zs_i} + \varepsilon tC\,.
\end{aligned} \tag{5.2}$$

We will use this to upper bound the probability that a supermartingale, which sums how many elements in each $\sigma_i[F_i]$ were present in $\bigcup_{j<i} \sigma_j[F_j]$, grows too large. Let B_1 be an arbitrary set of size $\frac{1}{8}C$ which contains X. (This definition will make the following analysis simpler than if we had set $B_1 = X$). For each $i \in \{2, \ldots, w\}$, define $B_i = B_{i-1} \cup \sigma_{i-1}[F_{i-1}]$. Note that $|B_w| \leq \frac{5}{8}C$. We have $\mathbb{E}[\sum_{i\in[w]} Y_{B_i,i}] \leq \sum_{i\in[w]} \left(\frac{|B_i|}{C}s_i + \varepsilon C\right) \leq \frac{|B_w|}{C}\sum_{i\in[w]} s_i \leq \frac{5}{8}\sum_{i\in[w]} s_i$.

Note that

$$\Pr\left[\left|\bigcup_{i\in[w]}\sigma_i[F_i]\setminus X\right| < \frac{1}{8}\sum_{i\in[w]} s_i\right] \le \Pr\left[\sum_{i\in[w]} Y_{B_i,i} \ge \frac{7}{8}\sum_{i\in[w]} s_i\right]. \tag{5.3}$$

Let $\gamma = (\frac{7}{8}\sum_{i\in[w]} s_i)/\mathbb{E}[\sum_{i\in[w]} Y_{B_i,i}]$; this is $\ge 7/5$. Applying a modified proof of the Chernoff bound/Azuma's inequality to the right hand side of Eq. 5.3 gives:

$$\begin{aligned}
&= \inf_{z>0} \Pr\left[\exp\left(z\sum_{i\in[w]} Y_{B_i,i}\right) \ge \exp\left(\gamma z\mathbb{E}[\sum_{i\in[w]} Y_{B_i,i}]\right)\right] \\
&\le \inf_{z>0} \frac{\exp\left(z\sum_{i\in[w]} Y_{B_i,i}\right)}{\exp\left(\gamma z\mathbb{E}[\sum_{i\in[w]} Y_{B_i,i}]\right)} \qquad \text{by Markov's inequality} \\
&= \inf_{z>0} \exp\left(z\sum_{i\in[w]} (Y_{B_i,i} - \gamma\mathbb{E}[Y_{B_i,i}])\right) \\
&= \inf_{z>0} \mathbb{E}\left[\exp\left(z(Y_{B_1,1} - \gamma\mathbb{E}[Y_{B_1,1}])\right)\cdots\mathbb{E}\left[\exp\left(z(Y_{B_w,w} - \gamma\mathbb{E}[Y_{B_w,w}])\right)\middle| Y_{B_1,1},\ldots,Y_{B_{w-1},w-1}\right]\cdots\right]
\end{aligned}$$

(In this step, we upper bound terms from the inside out, replacing each B_i with the B that maximizes the term.)

$$\begin{aligned}
&\le \inf_{z>0} \prod_{i\in[w]} \max_{B:\frac{1}{8}C\le|B|\le\frac{5}{8}C} \mathbb{E}\exp\left(z(Y_{B,i} - \gamma\mathbb{E}[Y_{B,i}])\right) \\
&= \inf_{z>0} \prod_{i\in[w]} \max_{B:\frac{1}{8}C\le|B|\le\frac{5}{8}C} \frac{\mathbb{E}\exp\left(zY_{B,i}\right)}{\exp\left(\gamma z\mathbb{E}[Y_{B,i}]\right)} \\
&\le \inf_{z>0} \prod_{i\in[w]} \max_{B:\frac{1}{8}C\le|B|\le\frac{5}{8}C} \frac{\exp\left(\frac{s_i|B|}{C}(e^z-1)\right) + \frac{(zs_i)^{t+1}}{(t+1)!}e^{zs_i} + \varepsilon tC}{\exp\left(\gamma z\mathbb{E}[Y_{B,i}]\right)}. \qquad \text{by Eq. 5.2}
\end{aligned}$$

Now set $z = \frac{t}{8\frac{1}{w}\sum_{i\in[w]} s_i}$. Since $t \le 2\min_{i\in[w]} s_i$, it follows $z \le \frac{2\min_{i\in[w]} s_i}{8\min_{i\in[w]} s_i} \le \frac{1}{4}$. Since $\max_{i\in[w]} s_i \le 2\min_{i\in[w]} s_i$, we have $z \le t/(4s_i)$ for all $i \in [w]$. This implies $(zs_i)^{t+1}/(t+1)!e^{zs_i} \le (t/4)^{t+1}e^{t/4}/(t+1)! \le \frac{1}{2}$. Since $\varepsilon \le C^{-t-1}$, we also have $\varepsilon tC \le \frac{1}{2}$. Continuing the upper bound of Eq. 5.3:

$$\begin{aligned}
&\le \prod_{i\in[w]} \max_{B:\frac{1}{8}C\le|B|\le\frac{5}{8}C} \frac{\exp\left(\frac{s_i|B|}{C}(e^z-1)\right)+1}{\exp\left(\gamma((zs_i|B|/C)-\varepsilon)\right)} \\
&\le e^{\gamma w\varepsilon} \prod_{i\in[w]} \frac{\exp\left(\frac{s_i}{8}(e^z-1)\right)+1}{\exp\left(\gamma zs_i/8\right)} \qquad \text{maximum occurs at } |B| = \frac{1}{8}C \\
&\le e^{w(\ln(2)+\gamma\varepsilon)} \prod_{i\in[w]} \exp((e^z - 1 - \gamma z)s_i/8) \qquad \text{since } (x+1)/x^\gamma \le 2/x^{\gamma-1}
\end{aligned}$$

$$\begin{aligned}
&\le e^{w(\ln(2)+\gamma\varepsilon)} \prod_{i\in[w]} \exp\left(-\frac{1}{4}z\frac{s_i}{8}\right) && \text{since } z \le \frac{1}{4} \text{ implies } \frac{1}{4}z \le \frac{7}{5}z - (e^z - 1)\\
&= \exp\left(-\frac{1}{4}\frac{tw}{8\sum_{i\in[w]} s_i}\sum_{i\in[w]}\frac{s_i}{8} + w(\ln 2 + \gamma\varepsilon)\right)\\
&\le \exp(-\frac{tw}{2^8} + w(\ln 2 + \gamma\varepsilon))\\
&\le \exp(-\frac{tw}{2^9}) && \text{since } \frac{t}{2^9} \ge 1 \ge \ln 2 + \gamma\varepsilon\,.
\end{aligned}$$

This completes the proof of the lemma. □

Lemma 5.4.5. *Let C, s, w be integers, with $s \ge 4$. Let $\varepsilon \le C^{-s-1}$. Say that $F_1, \ldots, F_w$ are subsets of $[C]$, with each $|F_i| \le s$, $|F_i| \ge 2$ and $\sum_{i\in[w]} |F_i| \ge \frac{1}{2}C$. Furthermore, let $X \subseteq [C]$ satisfy $|X| \le \frac{1}{8}C$. Then if $\sigma_1, \ldots, \sigma_w$ are (ε, s)-wise independent random permutations over $[C]$,*

$$\Pr\left[\left|\bigcup_{i\in[w]} \sigma_i[F_i] \setminus X\right| < \frac{1}{16}C\right] \le \exp(-\frac{1}{2}\sum_{i\in[w]} |F_i|)\,.$$

This lemma differs from Lemma 5.4.4 in that the sets $(\sigma_i[F_i])_{i\in[w]}$ are now smaller, and $\sum_{i\in[w]} |F_i|$ is $\Omega(C)$.

Proof of Lemma 5.4.5. Because the $(\sigma_i)_{i\in[w]}$ are (ε, s)-wise independent, for each $i \in [w]$, the random variable $\sigma_i[F_i]$ has total variation distance ε from being a uniform random subset of $[C]$ of size $|F_i|$. Let $\tau = \left\lfloor \frac{1}{16}C \right\rfloor$. We will bound the probability that there exists any set $T \subseteq [C]$ of size τ for which $\bigcup_{i\in[w]} \sigma_i[F_i] \subseteq T \cup X$. Observe:

$$\begin{aligned}
&\Pr\left[\exists T \in \binom{[C]}{\tau} : \bigcup_{i\in[w]} \sigma_i[F_i] \subseteq T \cup X\right]\\
&\le \sum_{T\in\binom{[C]}{\tau}} \Pr\left[\bigcup_{i\in[w]} \sigma_i[F_i] \subseteq T \cup X\right]\\
&= \sum_{T\in\binom{[C]}{\tau}} \prod_{i\in[w]} \Pr\left[\sigma_i[F_i] \subseteq T \cup X\right] && \text{since } \sigma_i \text{ are independent}\\
&\le \sum_{T\in\binom{[C]}{\tau}} \prod_{i\in[w]} \left(\frac{\binom{|T\cup X|}{|F_i|}}{\binom{C}{|F_i|}} + \varepsilon\right) && \text{since } F_i \text{ approximately uniform}\\
&= \sum_{T\in\binom{[C]}{\tau}} \prod_{i\in[w]} \left(\frac{|T\cup X|\cdots(|T\cup X| - |F_i| + 1)}{C\cdots(C - |F_i| + 1)} + \varepsilon\right)
\end{aligned}$$

$$
\begin{aligned}
&\le \sum_{T\in\binom{[C]}{\tau}} \prod_{i\in[w]} \left(\frac{|T\cup X|}{C}\right)^{|F_i|} && \text{since } 2\le |F_i|\le s \text{ and } \varepsilon \le C^{-s-1} \\
&\le \binom{C}{\tau} \max_{T\in\binom{[C]}{\tau}} \left(\frac{|T\cup X|}{C}\right)^{\sum_{i\in[w]}|F_i|} \\
&\le 2^{0.338C}\left(\frac{3}{16}\right)^{\sum_{i\in[w]}|F_i|} && \text{using } |T|+|X| \le \left(\frac{1}{16}+\frac{1}{8}\right)C \text{ and } \tau \le \frac{C}{16} \\
&\le \exp(0.235C - \ln(16/3)\sum_{i\in[w]}|F_i|) \\
&\le \exp(-\frac{1}{2}\sum_{i\in[w]}|F_i|)\,. && \text{since } \sum_{i\in[w]}|F_i| \ge C/2
\end{aligned}
$$

This completes the proof. □

Lemma 5.4.6. *Theorem 5.4.7 holds for one-sided vertex arrival streams on bipartite graphs.*

Proof of Lemma 5.4.6. $\delta \in (0,1)$ is a parameter governing the probability that Algorithm 5.4.2 will fail to be a correct deterministic algorithm, if its advice is chosen randomly. If one just wants a deterministic algorithm, setting $\delta = 1/2$ suffices. If $\Delta \le \log\frac{n\Delta}{\delta}$, we use the simple greedy algorithm (Algorithm 5.2.1). For $\Delta \ge \log\frac{n\Delta}{\delta}$, we use Algorithm 5.4.2.

This algorithm maintains, for each vertex $v \in B$, two variables b_v and Q_v that indicate which colors in $[C]$ are certainly available for that vertex. It ensures that none of the colors in the set $\Xi_v = \{\sigma_v[(b_v-1)s)] : i \in Q_v\} \sqcup \{\sigma_v[j] : j > b_v s\}$ have been used. When a new vertex x in A arrives, along with the multiset M_x of edges adjacent to it, the algorithm selects a set F_y indicating candidate colors $\sigma_y[F_y]$ for each y adjacent to x, and computes a matching between the edges in M_x and the set of all colors, allowing each edge in M_x only the colors corresponding to the edge's endpoint in B. This matching ensures that all edges incident to x receive different colors; and for all $y \in B$, the use of the set F_y to constrain the set of candidate colors to a subset of Ξ_v ensures that all incident to y receive different colors.

For a given vertex y, as the algorithm runs, b_y will be increased, by either Line 18 or 22. Line 18 only triggers when $|Q_y| \le s - \frac{1}{2^{17}}s$, which requires that vertex y has received $\ge \frac{1}{2^{17}}s$ incident edges since the last time b_y was increased. Since there will be at most Δ edges incident to y, the total increase to b_y from this line over the course of the algorithm will be $\le \Delta/(\frac{1}{2^{17}}s) = 2^{17}\Delta/s$. On the other hand, Line 18 only triggers when $d_{x,y} \ge \frac{1}{16}s$, and then increases b_y by $\lceil 64d_{x,y}/s\rceil + 1$. Since $\sum_{x\in A} d_{x,y} \le \Delta$,

$$
\sum_{x\in A: d_{x,y} > \frac{1}{16}s} \left(\left\lceil\frac{64d_{x,y}}{s}\right\rceil + 1\right) \le \sum_{x\in A: d_{x,y} > \frac{1}{16}s} \left(\frac{64d_{x,y}}{s} + 2\right) \le \sum_{x\in A: d_{x,y} > \frac{1}{16}s} \frac{96d_{x,y}}{s} \le \frac{96\Delta}{s}\,.
$$

Thus, the total increase in b_y will be $\le (2^{17}+96)\Delta/s$, and b_y will always be $\le (2^{17}+97)\Delta/s$.

Algorithm 5.4.2 Deterministic algorithm for $O(\Delta)$ edge coloring for one sided vertex arrival bipartite streams

```
**Input**: Stream of vertex arrivals for $n$-vertex graph $G = (A \sqcup B, E)$ of max degree $\Delta$

$\delta \in (0, 1)$ is a parameter so that, if the advice is chosen randomly, it will work for all inputs
with probability $\geq 1 - \delta$

Initialize:                                                    ▷ Requires: $\Delta \geq \log \frac{n\Delta}{\delta}$
Let $C = 2^{18}\Delta$.
Let $s = \lceil 2^{18} \log \frac{n\Delta}{\delta} \rceil$.
Advice: $(\sigma_v)_{v\in B}$, where each $\sigma_v$ is a permutation over $[C]$. If chosen randomly, each is $(\varepsilon, s)$-wise
independent for $\varepsilon \leq C^{-s-1}$
1: for $v \in B$ do
2:     $b_v \leftarrow 1$.
3:     $Q_v \leftarrow [s]$.
Process(vertex $x$ with multiset $M_x$ of edges to $B$)
Let $d_{x,y}$ be the number of times edge $\{x, y\}$ is in $M_x$
4: for each $y \in B$ with $d_{x,y} > 0$ do
5:     if $d_{x,y} < \frac{1}{16}s$ then
6:         Let $F_y = (b_y - 1)s + Q_y$
7:     else
8:         Let $F_y = ((b_y - 1)s + Q_y) \sqcup [b_y s, b_y + \lceil \frac{64 d_{a,b}}{s} \rceil s]$
9: Construct bipartite graph $H$ from $M_x$ to $[C]$, where each edge $e \in M_x$ connects to all $c \in \sigma_y[F_y]$.
10: Compute an $M_x$-saturating matching $P$ of $H$.
11: for each $e \in M_x$ do
12:     Assign color $P(e)$ to $e$
13:     if $d_{x,y} < \frac{1}{16}s$ then
14:         Remove $\sigma_y^{-1} - (b_y - 1)s$ from $Q_y$
15: for each $y \in B$ with $d_{x,y} > 0$ do
16:     if $d_{x,y} < \frac{1}{16}s$ then
17:         if $|Q_y| \leq s - \frac{1}{2^{17}}s$ then
18:             $b_y \leftarrow b_y + 1$
19:             $Q_y \leftarrow [s]$
20:     else
21:         $Q_y \leftarrow [s]$
22:         $b_y \leftarrow b_y + \lceil \frac{2 d_{x,y}}{s} \rceil + 1$
```

Looking at the construction of the set F_y on Lines 6 and 8, we see that it will only ever contain elements which are $\leq (2^{17} + 98)\Delta$. Since $C = 2^{18}\Delta \geq (2^{17} + 98)\Delta$, it follows that computing $\sigma_y[F_y]$ will never index out of range.

The only remaining way this algorithm could fail is if Line 10 were to report that no M_x-saturating matching exists. We will show that, if the $(\sigma_v)_{v\in B}$ are drawn from (ε, s)-wise independent distributions, then with probability $\geq 1-\delta$, for all possible sets M_x and combinations of "free slots",

$(F_y)_{y:d_{x,y}>0}$, Hall's condition will hold on the graph H constructed on Line 9.

Since whether the constructed graph H has a matching does not depend on the value of x, only on the number of edges arriving at a given $y \in B$, we do not need to take a union bound over all possible sets M_x. Instead, define a configuration by a tuple $(S, (d_y)_{y\in S}, (b_y)_{y\in S}, (Q_y)_{y\in S})$. The set S gives the neighborhood of x, and for each $y \in S$ we set $d_y = d_{x,y}$. The values b_y, Q_y match the values from the algorithm at the time M_x arrives. Note that the set F_y is a function of b_y, Q_y, and d_y, and for fixed b_y, Q_y is monotone increasing as a function of d_y. We do *not* need any extra cases to handle Hall's condition for subsets of M_x; consider any subset $M'_x \subseteq M_x$, and let S', d'_y, F'_y correspond to M'_x. Then because $F'_y \subseteq F_y$ for each $y \in S'$, if Hall's condition holds for the configuration $(S', (d'_y)_{y\in S'}, (b_y)_{y\in S'}, (Q_y)_{y\in S'})$, then

$$\left|\bigcup_{y\in S'} F_y\right| \geq \left|\bigcup_{y\in S'} F'_y\right| \geq \sum_{y\in S'} d'_y\,,$$

which implies that Hall's condition also holds for the subset M'_x within the bipartite graph H constructed for M_x.

Let $d = \sum_{y\in S} d_y$. If the permutations $(\sigma_y)_{y\in B}$ were each chosen uniformly at random, it would be straightforward to prove that Hall's condition fails for each configuration $(S,\ (d_y)_{y\in S},\ (b_y)_{y\in S},\ (Q_y)_{y\in S})$ with probability $\leq e^d (d/C)^{\Theta(|S|s+d)}$, after which a union bound over configurations gives a $\leq \delta$ total failure probability. However, because we assume the $(\sigma_y)_{y\in B}$ are only (ε, s)-wise independent, we will need a more precise argument.

Each configuration $(S, (d_y)_{y\in S}, (b_y)_{y\in S}, (Q_y)_{y\in S})$ can be split into $O(\log \Delta)$ different "level configurations". Let $\tau = s/16$; then for each vertex y, if $d_y < \tau$, the algorithm will choose F_y using Line 6 to be a subset of size $\leq s$ and $\geq s(1 - 2^{-17}) \geq s/2$; and if $d_y \geq \tau$, the algorithm will choose F_y using Line 8, to be a subset of size $\geq 64 d_y$. Define $L_0 = \{y \in S : d_y < \tau\}$. For each $\ell \in \{1, \ldots, \lambda\}$, for $\lambda = \lceil \log \Delta \rceil$ let $L_\ell = \{y \in S : 2^{\ell-1}\tau \leq d_y < 2^\ell \tau\}$. Also write $L_{>\ell} = \bigcup_{j>\ell} L_j$. Within each "level", the values of d_y are either all small ($< \tau$), or all within a factor 2 of each other. We will show that with high probability, the following two conditions hold:

$$\forall \ell \geq 1, \forall (L_\lambda, \ldots, L_{\ell+1}) \text{ where } \max_{i>\ell} (3/2)^{i-\ell} |L_i| \leq |L_\ell|, \forall d_y, b_y, Q_y \text{ for } y \in \bigsqcup_{j\geq\ell} L_\ell :$$

$$\left|\left(\bigcup_{y\in L_\ell} \sigma_y[F_y]\right) \setminus \left(\bigcup_{y\in L_{>\ell}} \sigma_y[F_y]\right)\right| \geq 8 \sum_{y\in L_0} d_y \tag{5.4}$$

$$\forall (L_\lambda, \ldots, L_1) \text{ where } \max_{i>0} (3/2)^i |L_i| \leq |L_0|, \forall d_y, b_y, Q_y \text{ for } y \in \bigsqcup_{j\geq 0} L_\ell :$$

$$\left|\left(\bigcup_{y\in L_0} \sigma_y[F_y]\right) \setminus \left(\bigcup_{y\in L_{>0}} \sigma_y[F_y]\right)\right| \geq \sum_{y\in L_0} d_y\,. \tag{5.5}$$

Then for the specific configuration $(S, (d_y)_{y \in S}, (b_y)_{y \in S}, (Q_y)_{y \in S})$, let $A \subseteq \{0, \dots, \lambda\}$ contain all ℓ for which $|L_\ell| \geq \max_{i > \ell} (3/2)^{i-\ell} |L_i|$. Because the condition of Eq. 5.4 holds for all $i \in A$ with $i > 0$, these "levels" of the configuration are associated with enough entries of C that they "pay for" all levels with smaller degrees that also do not have many more vertices. Level L_0 pays for itself if $0 \in A$, by Eq. 5.5. Formally, we have:

$$\begin{aligned}
|\bigcup_{y \in S} \sigma_y[F_y]| &= \sum_{i \in \{0,\dots,\lambda\}} |(\bigcup_{y \in L_i} \sigma_y[F_y]) \setminus (\bigcup_{y \in L_{>i}} \sigma_y[F_y])| \\
&\geq \sum_{i \in A} |(\bigcup_{y \in L_i} \sigma_y[F_y]) \setminus (\bigcup_{y \in L_{>i}} \sigma_y[F_y])| \\
&\geq \mathbb{1}_{0 \in A} \left(\sum_{y \in L_0} d_y \right) + \sum_{i \in A \setminus \{0\}} 8 \sum_{y \in L_i} d_y \\
&\geq \mathbb{1}_{0 \in A} \left(\sum_{y \in L_0} d_y \right) + \sum_{i \in A \setminus \{0\}} 8 \cdot 2^{i-1} \tau |L_i| \\
&= \mathbb{1}_{0 \in A} \left(\sum_{y \in L_0} d_y \right) + \sum_{i \in A \setminus \{0\}} 4 \cdot 2^{i} \tau |L_i| \\
&\geq \mathbb{1}_{0 \in A} \left(\sum_{y \in L_0} d_y \right) + \sum_{i \in A \setminus \{0\}} \sum_{j \leq i} \left(\frac{3}{4}\right)^{i-j} \left(2^i \tau |L_i|\right) \\
&\geq \mathbb{1}_{0 \in A} \left(\sum_{y \in L_0} d_y \right) + \sum_{i \in A \setminus \{0\}} \sum_{j \leq i : |L_j| \leq (3/2)^{i-j} |L_i|} 2^j \tau \left(\frac{3}{2}\right)^{i-j} |L_i| \\
&\geq \mathbb{1}_{0 \in A} \left(\sum_{y \in L_0} d_y \right) + \sum_{i \in A \setminus \{0\}} \sum_{j \leq i : |L_j| \leq (3/2)^{i-j} |L_i|} 2^j \tau |L_j| \\
&\geq \mathbb{1}_{0 \in A} \left(\sum_{y \in L_0} d_y \right) + \sum_{i \in A \setminus \{0\}} \sum_{j \leq i : |L_j| \leq (3/2)^{i-j} |L_i|} \sum_{y \in L_j} d_y \\
&= \sum_{i \in \{0,\dots,\lambda\}} \sum_{y \in L_j} d_y = \sum_{y \in S} d_y \,.
\end{aligned}$$

We first observe that Eq. 5.4 matches the conditions for Lemma 5.4.4. Specifically, if $y \in L_\ell$ for $\ell > 1$, then $d_y \geq s/16$, and we have both $|F_y| \geq s$ and $|F_y| \geq 64 d_y$. Also, since the $d_y \in [2^{\ell-1}\tau, 2^\ell \tau)$, we will have $\max_{y \in L_\ell} |F_y| \leq 2 \min_{y \in L_\ell} F_y$. Letting $X = \bigcup_{y \in L_{>i}} \sigma_y[F_y]$, we have $|X| \leq \sum_{y \in L_{>i}} |F_y| \leq \sum_{y \in L_{>\ell}} (2s + 64 d_y) \leq 96\Delta \leq \frac{1}{8} C$. Similarly, $\sum_{y \in L_\ell} \sigma_y[F_y] \leq 96\Delta \leq \frac{1}{2} C$. Thus, Lemma 5.4.4 applies, and gives an $\exp(-O(s|L_\ell|))$ upper bound on the probability that $|\bigcup_{y \in L_\ell} \sigma_y[F_y] \setminus X| \geq 4 \sum_{y \in L_0} d_y$.

For Eq. 5.5, if $\sum_{y \in L_0} |F_y| \geq \frac{1}{2} C$, we apply Lemma 5.4.5. As argued above, the set $X = \bigcup_{y \in L_{>0}} \sigma_y[F_y]$ will have size $\leq \frac{1}{8} C$. If the bad event in Lemma 5.4.5 does not hold, then the condition in Eq. 5.5 will, since $\sum_{y \in L_0} d_y \leq \Delta \leq \frac{1}{16}C$. On the other hand, if $\sum_{y \in L_0} |F_y| < \frac{1}{2}C$, we apply Lemma 5.4.4; this works because we have $|F_y| \geq (1 - 2^{-17})s \geq \frac{1}{2}s$.

Since Lemma 5.4.4 and Lemma 5.4.5 both ensure a $\exp(-\Omega(s|L_i|))$ type upper bound for the probability of conditions from Eqs. 5.4 and 5.5, we can bound the probability that none of the individual event fails in a single sum. Since for each y, Q_y is refreshed after at least $s/2^{17}$ elements are removed from it, there are only $\sum_{i=0}^{\lceil s/2^{17} \rceil} \binom{s}{i} \leq 2^{sH(1/2^{16})} \leq \exp(s/2^{11})$ possible values for Q_y; here H is the binary entropy function.

$$
\begin{aligned}
&\Pr[\text{Eqs. 5.4 and 5.5 hold}] \\
&\quad \leq \sum_{\ell \in \{0,\dots,\lambda\}} \sum_{\substack{w \in \{1,\dots,\Delta\} \\ \text{where } |L_\ell| = w}} \sum_{\substack{L_\lambda,\dots,L_\ell \\ \text{all } |L_j| \leq (2/3)^{j-\ell} w}} \sum_{(d_y, b_y, Q_y)_{y \in \bigsqcup_{j \geq \ell} L_j}} \exp(-sw/2^9) \\
&\quad \leq \sum_{\ell \in \{0,\dots,\lambda\}} \sum_{\substack{w \in \{1,\dots,\Delta\} \\ \text{where } |L_\ell| = w}} \prod_{j \geq \ell} \left((n+1)C^2 \exp(s/2^{11})\right)^{(2/3)^{j-\ell} w} \exp(-sw/2^9) \\
&\quad \leq \sum_{\ell \in \{0,\dots,\lambda\}} \sum_{\substack{w \in \{1,\dots,\Delta\} \\ \text{where } |L_\ell| = w}} \left((n+1)C^2 \exp(s/2^{11})\right)^{3w} \exp(-sw/2^9) \\
&\quad \leq \sum_{\ell \in \{0,\dots,\lambda\}} \sum_{\substack{w \in \{1,\dots,\Delta\} \\ \text{where } |L_\ell| = w}} (nC)^{6w} \exp(-sw/2^{11}) \qquad \text{since } (n+1) \leq n^2 \text{ and } 2^{-9} - 3 \cdot 2^{-11} = 2^{-11} \\
&\quad \leq \sum_{\ell \in \{0,\dots,\lambda\}} \sum_{\substack{w \in \{1,\dots,\Delta\} \\ \text{where } |L_\ell| = w}} \exp(-sw/2^{12}) \qquad \text{since } s \geq 6 \cdot 2^{12}(18 + \log(n\Delta)) \geq 6 \cdot 2^{12} \ln(nC) \\
&\quad \leq (\lambda+1)\Delta \exp(-s/2^{12}) \leq \delta \,. \qquad \text{since } s \geq 2^{13} \log(\Delta/\delta) \geq 2^{13} \ln(\Delta/\delta)
\end{aligned}
$$

Thus,

$$\Pr[\text{any configuration fails Hall's condition}] \leq \delta \,. \qquad \square$$

If the (ε, s)-wise random permutations over $[C]$ are constructed using Lemma 5.7.1 (assuming Δ is a power of two), then the total number of bits of randomness needed to sample advice for the algorithm will be $O(ns(\log C)^4 \log \frac{1}{\varepsilon}) = O(ns^2(\log C)^5) = O\left(n \left(\log \frac{n\Delta}{\delta}\right)^2 (\log \Delta)^5\right)$.

Combining Corollary 5.3.4 with Lemma 5.4.6, we immediately get the following theorem.

Theorem 5.4.7. *There is a deterministic online $O(\Delta)$-edge coloring algorithm for vertex arrival streams over multigraphs using $O(n \log(n\Delta))$ bits of space, using an advice string of length $\widetilde{O}(n)$, which works for all inputs. (By picking a uniformly random advice string, the same algorithm can alternatively be used as a robust algorithm with $1/\operatorname{poly}(n)$ error; the advice can also be computed in exponential time.)*

5.5 Edge coloring on edge arrival streams

5.5.1 W-streaming algorithm for edge arrivals

In this section, we describe a W-streaming (not online) algorithm for edge arrivals that builds on a given vertex arrival algorithm. Let $D = O(\sqrt{\Delta})$. We initiate $O(\sqrt{\Delta})$ copies of an $\widetilde{O}(n)$ space $O(D)$-coloring one-sided vertex arrival algorithm, where the max-degree can go up to D. Now, as the edges of a bipartite graph[5] with max-degree Δ arrive (in any arbitrary edge arrival order), for each node, we store its incident edges until its degree reaches $\sqrt{\Delta}$. Then, we feed all these edges to some copy of the vertex arrival algorithm. Observe that since each node can have $\sqrt{\Delta}$ such "blocks" of $\sqrt{\Delta}$ incident edges, $\sqrt{\Delta}$ copies suffice. However, when we feed the nodes from one side to the vertex arrival algorithm, we only ensure that the nodes on that side have degree at most $\sqrt{\Delta}$, but there is no guarantee on the nodes on the fixed side. We work around this by randomly assigning the nodes to the $\sqrt{\Delta}$ copies: this ensures that in all the copies, each node on the fixed side also has degree at most $O(\sqrt{\Delta}) = D$. Thus, each copy uses $O(\sqrt{\Delta})$ colors and since the $O(\sqrt{\Delta})$ copies use disjoint palettes, the total number of colors used is $O(\Delta)$. For the space bound, notice that each vertex arrival copy uses $\widetilde{O}(n)$ space, and hence the copies use $\widetilde{O}(n\sqrt{\Delta})$ space in total. Additionally, we store edges on a node until its degree reaches $\sqrt{\Delta}$, pass these edges to a vertex arrival copy, and delete them. Hence, this takes $\widetilde{O}(n\sqrt{\Delta})$ space as well.

We can extend this algorithm to handle multigraphs. Hence, by the space-color tradeoff technique of Lemma 5.3.5, we can obtain an $O(\Delta t)$-coloring algorithm in $\widetilde{O}(n\sqrt{\Delta/t})$ space for any $1 \le t \le \Delta$.

Lemma 5.5.1. *Given a streaming algorithm $\mathcal{A}$ for $O(\Delta)$ edge coloring for one-sided vertex arrival streams over bipartite multigraphs using $\le f(n,\Delta)$ space, we can construct a streaming algorithm $\mathcal{B}$ for $O(\Delta)$ edge coloring of edge arrival streams over bipartite multigraphs using $O(\sqrt{\Delta} f(n, O(\sqrt{\Delta})) + n\sqrt{\Delta}(\log n\Delta)\log(n/\delta))$ bits of space. The new streaming algorithm $\mathcal{B}$ is randomized, runs in polynomial time, and has additional $\le \delta$ probability of error, even if the input stream is adaptively generated.*

To prove this, we will need the following inequality:

Lemma 5.5.2 ([Fre75], Thm 1.6)**.** *Let $X_1, \ldots, X_n$ be $[-1,1]$ random variables, with zero conditional expectation ($\mathbb{E}[X_i | X_1, \ldots, X_{i-1}] = 0$). Let $S_k = \sum_{i=1}^{k} X_i$. (Note that $S_1, \ldots, S_n$ is a martingale.) Define $V_1, \ldots, V_n$ so that $V_i = \mathbb{E}[\mathrm{Var}[X_i] | X_1, \ldots, X_{i-1}]$. Then for $a, b > 0$,*

$$\Pr[\exists k : S_k \ge a \text{ and } V_k \le b] \le \exp(-a^2/(2a+b))\,.$$

[5]Recall that by Corollary 5.3.3 it's enough to consider only bipartite graphs.

Proof of Lemma 5.5.1. The W-streaming edge-arrival algorithm is given by Algorithm 5.5.1. The algorithm uses $s = O(\sqrt{\Delta})$ instances of $\mathcal{A}$. This algorithm maintains a pool P of edges, and whenever it receives a new edge it adds it to the pool. Edges with high multiplicity ($\geq \tau = \Omega(\sqrt{\Delta}/\log(n/\delta))$ in P are moved to a different pool L; since there are not many of this type, they can be stored using only $\widetilde{O}(n\sqrt{\Delta})$ space. When a vertex v reaches a high degree ($\geq \sqrt{\Delta}$) in the pool, it and its incident edges are removed from P and assigned to a random instance of $\mathcal{A}$ which has not yet received v. At the end of the stream, all edges still stored in $P \cup L$ are colored.

Algorithm 5.5.1 W-streaming algorithm for $O(\Delta)$ edge coloring on edge-arrival stream given black-box access to algorithm $\mathcal{A}$ for $C\Delta$ edge coloring on vertex-arrival stream

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

    Initialize:
 1: Let $s = 2\left\lceil\sqrt{\Delta}\right\rceil$
 2: Let $\tau = \left\lfloor \frac{\sqrt{\Delta}}{12\ln(n/\delta)} \right\rfloor$
 3: $P \leftarrow \emptyset$ is a multiset of edges                ▷ used to cache all arriving edges
 4: $L \leftarrow \emptyset$ is a multiset of edges   ▷ used to efficiently store edge types which have high multiplicity
 5: for $i \in [s]$ do
 6:     $\mathcal{I}^{(i)} \leftarrow$ instance of algorithm $\mathcal{A}$ for graphs of max degree $\lceil 4\Delta/s \rceil$; this will use $C\lceil 4\Delta/s \rceil$
    colors
 7:     $x^{(i)} \leftarrow [0, \dots, 0] \in \{0,1\}^A$      ▷ tracks for which vertices $w$ in $A$, $\mathcal{I}^{(i)}$ has received $(w, M_w)$
    Process(edge $\{x, y\}$)
 8: $P \leftarrow P \cup \{\{x, y\}\}$.
 9: if edge $\{x, y\}$ has multiplicity ¿ $\tau$ in $P$ then
10:     Remove all copies of $\{x, y\}$ from $P$, and add them to $L$
11:     return
12: if $\exists v \in A$ with degree $\geq \left\lceil\sqrt{\Delta}\right\rceil$ in $P$ then
13:     Pick random $i$ from $\left\{ j \in [s] : x_v^{(j)} = 0 \right\}$
14:     $x_v^{(i)} \leftarrow 1$                          ▷ Mark instance $\mathcal{I}^{(i)}$ as having received $v$
15:     Let $M_v$ be edges incident on $v$ in $P$       ▷ i.e, $M_v$ forms a "star" of degree $\sqrt{\Delta}$ around $v$
16:     Send $(v, M_v)$ to $\mathcal{I}^{(i)}$ to be colored
17:     Remove $M_v$ from $P$
    End of Stream
18: Color edges in $P \cup L$ greedily using an independent set of $2\Delta - 1$ colors
```

Algorithm 5.5.1 requires $sf(n)$ bits of space to store the instances $\mathcal{I}^{(1)}, \dots, \mathcal{I}^{(s)}$, and sn bits to keep track of the vectors $x^{(1)}, \dots, x^{(s)}$. Since the edges adjacent to a vertex in A are removed from P as soon as it reaches degree $\left\lceil\sqrt{\Delta}\right\rceil$, the total number of edges in P, counting multiplicity, will be $\leq |A|(\left\lceil\sqrt{\Delta}\right\rceil - 1) = O(n\sqrt{\Delta})$. Thus, P can be stored using $O(n\sqrt{\Delta}\log(n)$ bits of space. Finally, since L receives only edges whose multiplicity was at least τ, it will contain at most $(n\Delta/2)/\tau = O(n\sqrt{\Delta}\log(n/\delta))$ distinct edges; keeping track of them and their multiplicity can be done in $O\left(n\sqrt{\Delta}\log(n\Delta)(\log n)\log(n/\delta)\right)$ space. In total, Algorithm 5.5.1 will require $O(\sqrt{\Delta}(f(n) + n(\log n\Delta)^2)\log(n/\delta)))$ bits of space.

The total number of colors used is $2\left\lceil\sqrt{\Delta}\right\rceil \cdot C\lceil 4\Delta/s\rceil + (2\Delta - 1) = O(\Delta)$.

Because Algorithm 5.5.1 only sends a star around a vertex v to an instance $\mathcal{I}^{(i)}$ (Line 16) when the vertex v has degree $= \left\lceil\sqrt{\Delta}\right\rceil$ in P, the maximum degree of arriving vertices that any instance of $\mathcal{A}$ will process will be $\left\lceil\sqrt{\Delta}\right\rceil$. However, it is still possible that over the course of the stream, for some sketch $\mathcal{I}^{(i)}$, a vertex $v \in B$ will receive too many edges from vertices in A.

For each pair $i \in [s]$, $z \in B$, we will show that sketch $\mathcal{I}^{(i)}$ receives $\leq 4\Delta/s$ edges (counting multiplicity) for z, with $\geq 1 - \frac{\delta}{n^2}$ probability. For $j \in [\Delta]$, let W_j be the random variable giving the number of times the edge between between z and the center of jth star adjacent to z is present in the star. If the stream ends before a jth star is removed, then $W_j = 0$. Because Line 10 removes all edges with multiplicity $> \tau$ in P, we have $0 \leq W_j \leq \tau$. Also for each $j \in [\Delta]$, define Z_j to be the indicator random variable for the event that the jth star is sent to $\mathcal{I}^{(i)}$, by Line 13 ; if there is no jth star, $Z_j = 0$. Because $s = 2\left\lceil\sqrt{\Delta}\right\rceil$, and each star has root degree only $\left\lceil\sqrt{\Delta}\right\rceil$, there will always be $\geq s/2$ instances that have not received a given vertex as the root of a star, so $\mathbb{E}[Z_j] \leq \frac{2}{s}$. Now for each $j \in [\Delta]$, define the random variable $X_j = W_j Z_j$; this counts the number of edges from the jth star for $\mathcal{I}^{(i)}$. Because the choice of where to put a star is made *after* the star is picked, we in fact have $\mathbb{E}[Z_j | X_1, \ldots, X_{j-1}, W_j] \leq \frac{2}{s}$. This is true even if the input stream is produced by an adaptive adversary. Consequently,

$$\begin{aligned}\sum_{j\in[\Delta]} \mathbb{E}[X_j | X_1, \ldots, X_{j-1}] &= \sum_{j\in[\Delta]} \mathbb{E}[Z_j W_j | X_1, \ldots, X_{j-1}] \\ &\leq \frac{2}{s} \sum_{j\in[\Delta]} \mathbb{E}[W_j | X_1, \ldots, X_{j-1}] \leq \frac{2}{s} \max \sum_{j\in[\Delta]} W_j \leq \frac{2\Delta}{s}\,.\end{aligned}$$

The total number of edges received for z in $\mathcal{I}^{(i)}$ is $\sum_{j\in[\Delta]} X_j$, which has expectation $\leq \frac{2\Delta}{s}$. To bound the probability that this number exceeds its expectation by much, we apply the following argument, using martingales.

Let $Y_1, \ldots, Y_\Delta$ be random variables, where for each $i \in [\Delta]$, we define the difference $Y_i = \frac{1}{\tau}(X_i - \mathbb{E}[X_i | X_1, \ldots X_{i-1}])$. This ensures that $\mathbb{E}[Y_i | Y_1, \ldots, Y_{i-1}] = 0$. We also have $|Y_i| \leq 1$. Furthermore, since $\mathbb{E}|Y_i|^2 = \mathrm{Var}[X_i/\tau | X_1, \ldots X_{i-1}]] \leq \mathbb{E}[(X_i/\tau)^2 | X_1, \ldots X_{i-1}]]$, we have:

$$\begin{aligned}\sum_{i=1}^{\Delta} \mathbb{E}[Y_i^2 | Y_1, \ldots, Y_{i-1}] &\leq \sum_{i=1}^{\Delta} \mathbb{E}[\frac{1}{\tau^2} X_i^2 | X_1, \ldots, X_{i-1}] \\ &\leq \frac{1}{\tau} \sum_{i=1}^{\Delta} \mathbb{E}[X_i | X_1, \ldots, X_{i-1}] \qquad \text{since } X_i \leq \tau \\ &\leq \frac{1}{\tau} \cdot \frac{2\Delta}{s}\,.\end{aligned}$$

We now apply one of the inequalities stated in [Fre75], which we reproduced as Lemma 5.5.2, to find:

$$\Pr\left[\sum_{i=1}^{\Delta} Y_i \geq \frac{2\Delta}{\tau s}\right] \leq \Pr\left[\exists k \in [\Delta] : \sum_{i=1}^{k} Y_i \geq \frac{2\Delta}{\tau s} \text{ and } \sum_{i=1}^{k} \mathbb{E}[Y_i^2 | Y_1, \ldots, Y_{i-1}] \geq \frac{2\Delta}{s}\right]$$
$$\leq \exp\left(\frac{-(\frac{2\Delta}{\tau s})^2}{2(\frac{2\Delta}{\tau s} + \frac{2\Delta}{\tau s})}\right) = \exp\left(-\frac{\Delta}{2\tau s}\right) .$$

Since $\mathbb{E}[\sum_{j\in[\Delta]} X_j] \leq 2\Delta/s$,

$$\Pr\left[\sum_{i=1}^{\Delta} X_i \geq \frac{4\Delta}{s}\right] \leq \exp\left(-\frac{\Delta}{2\tau s}\right) = \exp\left(-\frac{\Delta}{2(2\left\lceil\sqrt{\Delta}\right\rceil)\left\lfloor\sqrt{\Delta}/(12\ln(n/\delta)\right\rfloor}\right)$$
$$\leq \exp\left(-2\ln(n/\delta)\right) \leq \frac{\delta}{n^2} . \qquad \text{since } 2\left\lceil\sqrt{\Delta}\right\rceil \leq 3\sqrt{\Delta} \text{ and } 1/\lfloor x \rfloor \geq 1/x$$

By a union bound over all $\leq n$ vertices $v \in B$, and all $\leq n$ instances in $\{\mathcal{I}^{(j)}\}_{j\in[s]}$, we have that the total probability of any vertex v in an instance $\mathcal{I}^{(i)}$ receiving more than $4\Delta/s$ edges is $\leq \delta$. □

Combining Lemma 5.5.1 with Lemma 5.4.1, and then applying Corollary 5.3.3 proves the following.

Theorem 5.5.3. *There is a randomized W-streaming algorithm for $O(\Delta)$ edge coloring on edge arrival streams for multigraphs which uses $O(n\sqrt{\Delta}(\log(n\Delta))^2)$ bits of space, with error $\leq 1/\operatorname{poly}(n)$ against any adaptive adversary. The algorithm also requires $\widetilde{O}(n\Delta)$ bits of oracle randomness.*

5.5.2 Randomized online algorithm for edge arrivals

The W-streaming algorithm is not online since it waits until a vertex has degree $\sqrt{\Delta}$ before passing it to the vertex arrival algorithm and announcing its edges' colors. To design an online algorithm under the same color and space bounds, we maintain a "tracker" set of $O(\sqrt{\Delta})$ colors available to a vertex. This set is obtained by shuffling through $O(\sqrt{\Delta})$-size "blocks" of an appropriate random permutation of the $O(\Delta)$-size palette. When an edge $\{u, v\}$ arrives, we find a color in the intersection of these sets of available colors for u and v, assign it to $\{u, v\}$, and remove it from both sets. We "refresh" the tracker set of a vertex to get a new block of colors when it shrinks by a (large enough) constant fraction. Although the algorithm is simple, the challenge is in proving that for each vertex, we can use the same tracker set of $s = O(\sqrt{\Delta})$ colors for $\Omega(s)$ of the neighbors of v. Specifically, for all those neighbors, we shall find a color in the intersection of the two tracker sets with high probability. For this, we need (ε, s)-wise independent random permutations (see Definition 5.2.1) and an involved analysis using their guarantees. Further, we show in Lemma 5.7.1 that even though the permutations are over $[c\Delta]$ for some large constant c, we can store only

$O(s \text{ polylog}(c\Delta)) = \widetilde{O}(\sqrt{\Delta})$ bits and generate such a permutation. Hence, we can store all the permutations in our memory of $\widetilde{O}(n\sqrt{\Delta})$.

The following online edge coloring algorithms will both use the same core primitive; a pool of random colors, which is periodically refreshed, along with data to keep track of which colors in the pool have been used so far. The times at which the pool are refreshed only depend on the number of colors that were used, and not which colors where used; this property makes the primitive easier to handle in proofs. For the randomized algorithm, we will not need the optional reference tracking.

Listing 5.5.2 Storing free regions from a permutation

```
    $F \leftarrow$ InitFreeTracker($C$,$s$,$\Delta$,$\sigma$):    ▷ Assume $C$,$s$,$\Delta$ are powers of two, and $\sigma$ permutation of $[C]$, and $C \geq \Delta$
 1: $H \leftarrow [s]$ be a subset of $[s]$
 2: $b \leftarrow 1$ be a counter between 1 and $C/s$
 3: Optional: $Q \leftarrow \emptyset$ is a set of references to objects

    Interpreting $F$ as subset of $[C]$
 4: return $\sigma[H + (b-1)s]$

    $F$.RemoveAndUpdate($c$, optional: $o$)    ▷ Requires $c \in F_v$
 5: $H \leftarrow H \setminus \{\sigma^{-1}(c)\}$
 6: Optional: Add a reference to $o$, and store it in $Q$
 7: if $|H| \leq s - s\Delta/C$ then    ▷ Switch to next block
 8:     $H \leftarrow [s]$
 9:     $b \leftarrow b + 1$
10:     Optional: Drop all references in $Q$ and set $Q \leftarrow \emptyset$
```

Theorem 5.5.4. *Given any adversarial edge-arrival stream of a simple graph, there is a randomized algorithm for online $O(\Delta)$-edge-coloring using $O(n\sqrt{\Delta \log n})$ bits of space and $\widetilde{O}(n\sqrt{\Delta})$ oracle random bits.*

Proof of Theorem 5.5.4. We will show that Algorithm 5.5.3 satisfies the claims of the lemma, if $\Delta = \Omega(\log(n/\delta))$. (For smaller values of Δ, fall back to the greedy algorithm (Algorithm 5.2.1).) In the following argument, we shall assume that the permutations $(\sigma_v)_{v \in S}$ are s-wise independent. The pseudocode states (ε, s)-wise independence, since that is attainable per Lemma 5.7.1 using only $O(s \operatorname{poly}(\log 1/\varepsilon, \log s))$ bits of randomness per permutation. This will not affect the validity of the proof, since it at most increases the probabilities of events $H_{\{u,v\}}$ and $J_{u,v}$ defined later by ε, which is polynomially smaller than the losses in the argument due to bounding the number of events by n^2 instead of $\binom{n}{2}$ or $n^2 - n$. We also assume that Δ is a power of two; if not, we can increase Δ to the nearest power of two, and the algorithm will still give an $O(\Delta)$ coloring.

Each color tracker F_v can be stored using $O(\log \Delta)$ bits for b, and $O(s)$ bits for H; thus Algorithm 5.5.3 will use $O(n(s + \log \Delta)) = O(n\sqrt{\Delta \log(n/\delta)})$ bits in total. For $\Delta = O(\log(n/\delta))$, the Algorithm 5.2.1 uses $O(n\Delta)$ bits, which is also $O(n\sqrt{\Delta \log(n/\delta)})$.

Algorithm 5.5.3 Randomized algorithm for $O(\Delta)$ edge coloring for simple graph edge arrival streams

```
    Input: Stream of vertex arrivals n-vertex graph $G = (A \sqcup B, E)$
    Assume $\Delta$ is a power of two, and $\Delta = \Omega(\log(n/\delta))$

    Initialize:
 1: Let $C = 128\Delta$
 2: Let $s$ be the least power of two which is $\geq 128\sqrt{\Delta \log(n/\delta)}$
 3: Let $\mathcal{H}$ be an $(\varepsilon, s)$-wise independent distribution of permutations on $[C]$, with $\varepsilon \leq \exp(-s^2/C) \leq (\delta/n)^{128}$
 4: for $v \in B$ do
 5:     Let $\sigma_v$ be a random permutation from $\mathcal{H}$
 6:     $F_v \leftarrow$ InitFreeTracker($C, s, \Delta, \sigma_v$), without reference count tracking

    Process(edge $\{x, y\}$) → color
 7: if $F_x \cap F_y = \emptyset$ then
 8:     abort
 9: Let $c$ be chosen uniformly at random from $F_x \cap F_y$.
10: $F_x$.RemoveAndUpdate(c)
11: $F_y$.RemoveAndUpdate(c)
12: return color $c$
```

For each $v \in V$, $i \in [C/s]$, write $P_{v,i}$ for the set $\sigma[[s] + (i-1)s]$ of the free region tracker F_v for vertex v. (See Listing 5.5.2.) Since we are assuming the σ_v are s-wise independent, the set $P_{v,i}$ will be uniformly distributed over over $\binom{[C]}{s}$.

Consider a fixed input stream $e_1, e_2, \ldots$, where the edges of the stream together form the simple graph G. Write $b_{x,\{u,v\}}$ for the value of the counter b inside F_x just before the algorithm processed edge $\{u, v\}$. Let $D_{\{u,v\}} := P_{u,b_{u,\{u,v\}}} \cap P_{v,b_{v,\{u,v\}}}$. Also define $M_{u,v} := \{x : \{x, u\} \in G \wedge b_{u,\{x,u\}} = b_{u,\{u,v\}} \wedge \{x, u\} \prec \{u, v\}\}$; this is the set of vertices which were adjacent to u, for which the edge $\{x, u\}$ was added before $\{u, v\}$, and while the value of the counter b inside F_u for vertex u was the same as it was at the time $\{u, v\}$ was added. This is the set of vertices whose color choices *might* reduce the size of $F_u \cap F_v$ at the time $\{u, v\}$ is added.

We will first show that of the following two classes of m events, the probability that any of the events is true is $\leq \delta/2$.

$$\forall \{u, v\} \in G : H_{\{u,v\}} := \left\{ |D_{\{u,v\}}| \leq \frac{1}{2} s^2/C \right\}$$

$$\forall (u, v) \text{ where } \{u, v\} \in G : J_{u,v} := \left\{ \sum_{x \in M_{u,v}} |D_{\{u,v\}} \cap P_{x,b_{x,\{x,u\}}}| \geq 2 \cdot \frac{3}{2} \frac{s^3}{C^2} \frac{\Delta s}{C} \right\}.$$

To bound the probability of $H_{\{u,v\}}$, we let $X_1, \ldots, X_C$ be indicator random variables where $X_i = 1$ iff $i \in P_{u,b_{u,\{u,v\}}}$. Since the X_i are negatively associated, by the Chernoff bound applies (see Lemma 2.3.3), and

$$\begin{aligned}
\Pr[|D_{\{u,v\}}| \leq \frac{1}{2}s^2/C] &= \Pr[|D_{\{u,v\}}| \leq \frac{1}{2}\mathbb{E}[|D_{\{u,v\}}|]] \\
&= \Pr[\sum_{i \in [P_{v,b_{v,\{u,v\}}}]} X_i \leq \frac{1}{2}\mathbb{E}[|D_{\{u,v\}}|]] \\
&\leq \exp(-\frac{1}{8}\mathbb{E}[|D_{\{u,v\}}|]) = \exp(-\frac{s^2}{8C}) \\
&\leq \exp(-16\log\frac{n}{\delta}) \leq \frac{\delta}{2n^2}\,.
\end{aligned}$$

To bound the probability of the events $\{J_{u,v}\}$, we will show that $|D_{u,v}|$ is not too large w.h.p, and conditioned on that, the sum $\sum_{x \in M_{u,v}} |D_{\{u,v\}} \cap P_{x,b_{x,\{x,u\}}}|$ is not too large w.h.p. With $\{X_i\}_{i \in [C]}$ as defined above:

$$\begin{aligned}
\Pr[|D_{\{u,v\}}| \geq \frac{3}{2}s^2/C] &= \Pr[|D_{\{u,v\}}| \geq \frac{3}{2}\mathbb{E}[|D_{\{u,v\}}|]] \\
&\leq \Pr[\sum_{i \in [P_{v,b_{v,\{u,v\}}}]} X_i \geq \frac{3}{2}\mathbb{E}[|D_{\{u,v\}}|]] \\
&\leq \exp(-\frac{1}{10}\mathbb{E}[|D_{\{u,v\}}|]) = \exp(-\frac{s^2}{10C}) \\
&\leq \exp(-\frac{128}{10}\log\frac{n}{\delta}) \leq \frac{\delta}{4n^2}\,.
\end{aligned}$$

The permutations $\{\sigma_x\}_{x \in M_{u,v}}$ are independent of σ_u and σ_v. For each $x \in M_{u,v}$, let $Y_{1,x}, \ldots, Y_{C,x}$ be indicator random variables where $Y_{i,x}$ is 1 iff $i \in P_{x,b_{x,\{x,u\}}}$, and zero otherwise. Due to the frequency of free color buffer refreshing, $|M_{u,v}| \leq s\Delta/C$; and since $|P_{x,b_{x,\{x,u\}}}| = s$, $\mathbb{E}Y_{i,x} = s/C$. Since the $\{Y_{i,x}\}_{i \in D_{\{u,v\}}, x \in M_{u,v}}$ are negatively associated, we can apply a Chernoff bound. If we assume that $|D_{\{u,v\}}| \leq \frac{3s^2}{2C}$, then we have:

$$\begin{aligned}
\Pr\left[\sum_{x \in M_{u,v}} |D_{\{u,v\}} \cap P_{x,b_{x,\{x,u\}}}| \geq 2\frac{\Delta s^2}{C^2}\frac{3s^2}{2C}\right] &= \Pr\left[\sum_{i \in D_{\{u,v\}}, x \in M_{u,v}} Y_{i,x} \geq 2\frac{\Delta s^2}{C^2}\frac{3s^2}{2C}\right] \\
&\leq \exp\left(-\frac{1}{8}\frac{\Delta s^2}{C^2}\frac{3s^2}{2C}\right) \\
&\leq \exp(-12(\log(n/\delta))^2) \leq \frac{\delta}{4n^2}\,.
\end{aligned}$$

Thus, the probability that either $|D_{\{u,v\}}| \geq \frac{3s^2}{2C}$ or event $J_{u,v}$ does not hold is $\frac{\delta}{2n^2}$.

For the rest of the proof, we will consider the case where none of the events $J_{u,v}$ or $H_{\{u,v\}}$ holds; this happens with probability $\geq 1 - \delta/2$. Fix values of the $(\sigma_v)_{v \in V}$ satisfying none of the events.

The only other random decisions made by the algorithm are the choices made on Line 9, randomly choosing the edge color $\chi_{\{u,v\}}$ for $\{u,v\}$ from $F_u \cap F_v$. We will prove by induction on the number of edges processed that the probability of $|F_u \cap F_v| \leq \frac{1}{4}s^2/C$ holding at the time Line 9 is executed, in total over all t edges so far is, $\leq \delta \cdot t/(2n^2)$.

To do this, we will use the following lower bound:

$$|F_u \cap F_v| \geq |D_{\{u,v\}}| - \sum_{x \in M_{u,v}} W_{x,u} - \sum_{x \in M_{u,v}} W_{x,v} \,. \tag{5.6}$$

Here $W_{x,u}$ is the indicator random variable for the event that the color chosen for $\{x,u\}$ was in $D_{\{u,v\}}$. The lower bound overcounts the number of colors in $D_{\{u,v\}}$ that have been removed from $F_u \cap F_v$.

The base case of the induction (0 edges) is immediate. Assume that we are processing edge $\{u,v\}$, and that all edges $\{x,y\}$ earlier in the stream, when they were processed, had $|F_x \cap F_y| \geq \frac{s^2}{4C}$. For each $x \in M_{u,v}$, the color $\chi_{\{x,u\}}$ was drawn uniformly at random from *some* set $F_x \cap F_u$, which we assume satisfies $|F_x \cap F_u| \geq \frac{s^2}{4C}$. For any subset H of $D_{\{x,u\}}$ of size $\geq \frac{s^2}{4C}$, if $\hat{\chi}$ is chosen u.a.r. from H, then

$$\Pr[\hat{\chi} \in D_{\{u,v\}}] \leq \frac{|H \cap D_{\{u,v\}}|}{|H|} \leq \frac{4C}{s^2} |P_{x,b_{x,\{x,u\}}} \cap D_{\{u,v\}}|$$

Conditioned on the color choices of all earlier edges, we thus have $\mathbb{E} W_{x,u} \leq \frac{4C}{s^2} |P_{x,b_{x,\{x,u\}}} \cap D_{\{u,v\}}|$. Thus

$$\begin{aligned} \mathbb{E}\left[\sum_{x \in M_{u,v}} W_{x,u} + \sum_{x \in M_{v,u}} W_{x,v}\right] &\leq \frac{4C}{s^2}\left(\sum_{x \in M_{u,v}} |P_{x,b_{x,\{x,u\}}} \cap D_{\{u,v\}}| + \sum_{x \in M_{v,u}} |P_{x,b_{x,\{x,v\}}} \cap D_{\{u,v\}}|\right) \\ &\leq \frac{4C}{s^2} \cdot 2\frac{\Delta s^2}{C^2}\frac{3s^2}{2C} = 12\frac{\Delta}{C}\frac{s^2}{C} < \frac{s^2}{8C} \,, \end{aligned}$$

where the last inequality holds because $\Delta \leq C/128$.

By the multiplicative formulation of Azuma's inequality, Lemma 2.3.1,

$$\begin{aligned} &\Pr\left[\sum_{x \in M_{u,v}} W_{x,u} + \sum_{x \in M_{v,u}} W_{x,v} \geq \frac{s^2}{4C}\right] \\ &\leq \exp(-\frac{1}{3}\frac{s^2}{4C}) \leq \exp\left(-\frac{32}{3}\log\frac{n}{\delta}\right) \leq \frac{\delta}{2n^2} \,. \end{aligned}$$

By a union bound over all edges, the probability that any edge $\{u,v\}$ has $|F_u \cap F_v| \leq \frac{s^2}{4C}$ is $\leq \delta/2$.

We have shown that, in total, the probability of the algorithm aborting because $F_u \cap F_v = \emptyset$ is $\leq \delta$. □

Algorithm 5.5.3 can be generalized to produce $O(\Delta^2/t)$ edge colorings using $\widetilde{O}(n\sqrt{t})$ bits of space, by increasing the parameters C and s while ensuring that $s^2/C = \Omega(\log(n/\delta))$. Then as at most $s\Delta/C$ colors are removed from each free color tracker, it will be possible to store each free color tracker using $\widetilde{O}(s\Delta/C)$ bits of space. However, further adjustment would be necessary to make the algorithm use $\widetilde{O}(n\sqrt{t})$ random bits. We suspect that picking $(\varepsilon, O(s^2/C))$-wise independent distributions will be sufficient. We did not attempt this possibly tedious proof, as the following Theorem 5.5.8 already provides a color-space tradeoff for the edge arrival setting.

5.5.3 Deterministic online algorithm for edge arrivals

We did not find a way to directly derandomize the randomized algorithm for online edge arrival. Instead, we created an algorithm that manages to *partially* solve the edge coloring problem, only assigning a color to a $\geq 1/3$ fraction of the incoming edges, and leaving the rest uncolored. Now, say we run $O(\log \Delta)$ instances of this algorithm in parallel, each using a distinct palette of $O(\Delta)$ colors. When an edge arrives, we pass it to the first instance of the algorithm, and if it wasn't assigned a color, pass it to the second instance; and if that didn't assign a color, pass it to the third instance, and so on. All in all, only a $\leq 1/\operatorname{poly}(\Delta)$ fraction of the input stream will fail to be colored by this process, and there are few enough of these leftover edges that one can store them all and color them using $O(\Delta)$ colors. This approach uses $O(\Delta \log \Delta)$ colors in total.

The partial edge coloring algorithm itself uses an interesting trick. Each vertex v has an associated permutation $\sigma_v \in \mathbb{S}_{O(\Delta)}$, which is partitioned into a number of blocks $P_{v,1}, P_{v,2}, \ldots$ of $\widetilde{O}(\sqrt{\Delta})$ colors each. Whenever an edge $\{u, v\}$ arrives, it must be colored using a color in the set $P_{v,i} \cap P_{u,j}$, where i and j depend on the degrees of vertices v and u at the time. Parameters are set up so this set has size $\Omega(\log n)$, and the algorithm knows which colors in $P_{v,i}$ and $P_{u,j}$ have been used so far. We prove that, if the algorithm could preview the future of the stream, it could always pick the "right" color in $P_{v,i} \cap P_{u,j}$, and thereby find a valid edge coloring. On the other hand, without being able to look at future edges, if one just greedily picks valid colors in $P_{v,i} \cap P_{u,j}$ that don't conflict with colors chosen earlier — assuming there are any — then the algorithm will color at least a $1/3$ fraction of the edges.

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

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

To handle the "bad" type of repeated edges, we maintain $O(\log \Delta)$ modified instances of an edge arrival algorithm. (We use the basic deterministic online edge arrival algorithm, but the same argument would work for the randomized one.) The first instance processes all edges in the stream, filtering out the edges which it detects to have repeated at least once. The repeated edges are sent to the second instance, which filters out edges that it finds to have repeated twice, and sends those to the third instance. In general, the ith instance will receive edges which have been seen $\Theta(2^i)$ times in the stream. (The exact condition is a bit more complicated.) The ith instance is also modified to handle edges with high repetition rates, assigning batches of colors to each edge type that it processes.

We now introduce a technical lemma which will be useful in the proof of Theorem 5.5.8.

Lemma 5.5.5. *Let V be a set of size n, $\delta \in (0,1)$, and let Δ be a power of two, satisfying $\Delta \geq 256 \log \frac{n}{\delta}$. Define $C = 32\Delta$, and let s be the least power of two which is $\geq 512\sqrt{\Delta \log \frac{n}{\delta}}$. Let $(\sigma_v)_{v \in V}$ be randomly chosen permutations from an (ε, s)-wise independent family, where $\varepsilon \leq \exp(-s^2/C) \leq (\delta/n)^{1024}$. For $i \in [C/s]$, $v \in V$, let $P_{v,i} := \sigma[s(i-1)+[s]]$.*

We say that the permutations $(\sigma_v)_{v \in V}$ are good *if, for all simple graphs H on $V \times [C/s]$ for which, for any $u, w \in V$ and $i \in [C/s]$, there is at most one j for which edge $\{(u,i),(v,j)\}$ is in H, and the max degree of H is $\leq s\Delta/C$; that the graph H can be list-edge colored where edge $\{(u,i),(v,j)\}$ may only use colors in $P_{u,i} \cap P_{v,j}$.*

The probability that the $(\sigma_v)_{v \in V}$ are good *is $\geq 1 - \delta$.*

Proof of Lemma 5.5.5. We will prove this in two steps. First, define a specific Property U that the $(\sigma_v)_{v \in V}$ should satisfy with probability $\geq 1-\delta$; second, prove that if this property holds, then any graph H can be colored.

The permutations $(\sigma_v)_{v \in V}$ satisfy Property U if:

- For all pairs $(u,i),(v,j) \in V \times [C/s]$, with $u \neq v$, we have $|P_{u,i} \cap P_{v,i}| \geq \frac{s^2}{2C}$.
- For each $(u,i) \in V \times [C/s]$, $S \subseteq (V \setminus \{s\}) \times [C/s]$ where $|S| \leq s\Delta/C$ and S includes no two vertices $(v,i),(u,j)$ with $v = u$, and all $T \in \binom{P_{u,i}}{|S|-1}$, there exists some $(x,j) \in S$ for which $|P_{x,j} \cap T| < \frac{1}{10}|P_{x,j} \cap P_{u,i}|$. (This is, in effect, a stronger version of Hall's condition).

For the first part of Property U, it is straightforward to bound the probability that it does not hold. For a given pair $(u,i),(v,j) \in V \times [C/s]$, $u \neq v$, because the permutations are (ε, s)-wise independent, the sets $P_{u,i}$ and $P_{v,j}$ are within ε-total-variation distance of being uniformly random subsets of $[C]$ of size s, we can apply a Chernoff bound (specifically, Lemma 2.3.3) for the number of elements in $P_{u,i}$ that lie in $P_{v,j}$:

$$\Pr[|P_{u,i} \cap P_{v,j}| \leq \frac{1}{2}\frac{s^2}{C}] \leq \exp(-\frac{1}{8}\frac{s^2}{C}) + \varepsilon \leq \exp(-2^{10}\log(n/\delta)) + \varepsilon \leq \frac{\delta}{2n^2}\,. \tag{5.7}$$

(The additive factor ε accounts for the maximum difference in probabilities for this event between the case where $P_{u,i}$ is exactly uniform and the case where it is ε-far from such.)

For the second part, consider a specific combination (u, i, S, T), and fix $P_{u,i}$. Then the probability that this combination violates property U is:

$$\Pr\left[\bigwedge_{(x,j)\in S}\left\{|P_{x,j} \cap T| \geq \frac{1}{10}|P_{x,j} \cap P_{u,i}|\right\}\right] \leq \prod_{(x,j)\in S}\Pr_{P_{x,j}}\left[|P_{x,j} \cap T| \geq \frac{1}{10}|P_{x,j} \cap P_{u,i}|\right]\,. \tag{5.8}$$

since the $P_{x,j} \in S$ are all independent, since S contains at most one entry for each $v \in V$. Since $P_{x,j}$ is a uniformly random subset $[C]$, $P_{x,j} \cap P_{u,i}$ is symmetrically distributed over $P_{u,i}$. Now let $\hat{T}$ be a uniformly random element of $\binom{P_{u,i}}{s}$, and define indicator random variables $\{Y_k\}_{k\in P_{u,i}}$ so that $Y_k = 1$ iff $k \in \hat{T}$; these are negatively associated and $\mathbb{E}[Y_k] = \frac{|S|-1}{s}$. Thus, if we assume $|P_{x,j} \cap P_{u,i}| = h$:

$$\begin{aligned}
&\Pr_{P_{x,j}}\left[|P_{x,j} \cap T| \geq \frac{1}{10}|P_{x,j} \cap P_{u,i}|\middle|\,|P_{x,j} \cap P_{u,i}| = h\right] \\
&\quad\leq \Pr_{\hat{T}}\left[|P_{x,j} \cap \hat{T}| \geq \frac{1}{10}h\middle|\,|P_{x,j} \cap P_{u,i}| = h\right] + \varepsilon \\
&\quad= \Pr_{\{Y_k\}_{k\in P_{u,i}}}\left[\sum_{k\in P_{u,i}} Y_k \geq \frac{1}{10}h\middle|\,|P_{x,j} \cap P_{u,i}| = h|\right] + \varepsilon \\
&\quad\leq \exp\left(-2\left(\frac{1}{10} - \frac{|S|-1}{s}\right)^2 h\right) + \varepsilon \\
&\quad\leq \exp\left(-2\left(\frac{1}{10} - \frac{\Delta}{C}\right)^2 h\right) + \varepsilon \leq \exp\left(-h/200\right) + \varepsilon\,. \qquad \text{since } C = 32\Delta
\end{aligned}$$

This bound is useful only if h is large enough. By the law of total probability, and using the bound from Eq. 5.7 to handle the case where h is small:

$$\begin{aligned}
&\Pr_{P_{x,j}}\left[|P_{x,j} \cap T| \geq \frac{1}{10}|P_{x,j} \cap P_{u,i}|\right] \\
&\quad\leq \Pr_{P_{x,j}}\left[|P_{x,j} \cap T| \geq \frac{1}{10}|P_{x,j} \cap P_{u,i}|\middle|\,|P_{x,j} \cap P_{u,i}| \geq \frac{s^2}{2C}\right]\Pr\left[|P_{x,j} \cap P_{u,i}| \geq \frac{s^2}{2C}\right] + \\
&\qquad\qquad \Pr\left[|P_{x,j} \cap P_{u,i}| \leq \frac{s^2}{2C}\right]
\end{aligned}$$

$$\leq (\exp\left(-\frac{s^2}{400C}\right) + \varepsilon) \cdot 1 + (\exp(-\frac{s^2}{8C}) + \varepsilon) \leq 2\exp\left(-\frac{s^2}{400C}\right) .$$

Substituting this result into Eq. 5.8 gives:

$$\Pr\left[\bigwedge_{(x,j)\in S}\left\{|P_{x,j}\cap T| \geq \frac{1}{10}|P_{x,j}\cap P_{u,i}\right\}\right] \leq 2^{|S|}\exp\left(-|S|\frac{s^2}{400C}\right) .$$

Taking a union bound over all (u, i, S, T) tuples gives:

$$\begin{aligned}
&\Pr[\text{second part of Property U fails}] \\
&\leq \sum_{(u,i)\in V\times[C/s]}\sum_{k=1}^{s\Delta/C}\sum_{\text{valid } S \text{ with } |S|=k}\sum_{T\subseteq\binom{P_{u,i}}{k-1}} 2^k\exp\left(-k\frac{s^2}{400C}\right) \\
&\leq n\cdot\sum_{k=1}^{s\Delta/C}\cdot\binom{n-1}{k}\left(\frac{C}{s}\right)^k\cdot\binom{s}{k-1}\cdot 2^k\exp\left(-k\frac{s^2}{400C}\right) \\
&\leq n\sum_{k=1}^{s\Delta/C}\left(2nCs\exp\left(-\frac{s^2}{400C}\right)\right)^k \\
&\leq 2n\cdot 2nCs\exp\left(-\frac{s^2}{400C}\right) \qquad \text{for large enough } s^2/C \\
&\leq 4n^2\cdot(32n)^2\exp(-\frac{2048}{100}\log(n/\delta)) \leq \frac{\delta}{2} . \qquad \text{since } s \leq C = 32\Delta
\end{aligned}$$

Combining this with a union bound over Eq. 5.7 implies that Property U fails to hold with probability $\leq \delta$.

For the second stage of the proof, we consider the following iterative process to color any graph H satisfying the given conditions. Consider an arbitrary ordering $v_1, \ldots, v_n$ of the vertices in $V \times [C/s]$. For a given vertex v_t, let $A(v_t)$ be the set of vertices in $\{v_1, \ldots, v_{t-1}\}$ which are adjacent to (v_t, i), and let $B(v_t, i)$ be the set of vertices in $\{v_{t+1}, \ldots, v_n\}$ which are adjacent to (v_t, i). For each $z \in A(v_t) \cup B(v_t)$, define $U_{t,v_t,z}$ to be the set of colors in $P_{v_t} \cap P_z$ that were already used by edges to vertices in $A(v_t)$ just *after* step t. The color assignment chosen will maintain the Invariant W that $|U_{t,z,(v_{t'})}| \leq \frac{1}{3}|P_z \cap P_{v_{t'}}|$ for all $t' > t$ and $z \in A(v_{t'})$. In other words, that when it is time to color the edges from a future vertex $v_{t'}$ to $A(v_{t'})$, only a $\leq 1/3$ fraction of the initially possible color options will have been used.

Invariant W automatically holds when $t = 0$, since no edges have been colored. Say the invariant holds at time $t - 1$. Then we are guaranteed that $|U_{t-1,z,v_t}| \leq \frac{1}{3}|P_z \cap P_{v_t}|$ for all $z \in A(v_t)$, and want to find color assignments for the edges from $A(v_t)$ to z so that $|U_{t,z,v_t}| \leq \frac{1}{3}|P_z \cap P_{v_t}|$ for all

$z \in B(v_t)$. To do this, we will first pick a set $F \subseteq P_{v_t}$ that satisfies:

$$\forall x \in A(v_t): \quad |(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F| \geq \frac{1}{10}|P_x \cap P_{v_t}| \tag{5.9}$$

$$\forall x \in B(v_t): \quad |(P_y \cap P_{v_t}) \setminus F| \leq \frac{1}{3}|P_y \cap P_{v_t}|\,. \tag{5.10}$$

That such a set F exists follows by the probabilistic method; say F were chosen so that each element of P_{v_t} is included u.a.r with probability $\frac{7}{10}$. For $i \in P_{v,t}$, let X_i be the indicator random variable for the event that $i \in F$. Then the probability of Eq. 5.9 is bounded by:

$$\begin{aligned}
&\Pr\left[|(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F| \geq \frac{1}{10}|P_x \cap P_{v_t}|\right] \\
&\quad \leq \Pr\left[|(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F| \geq \frac{3}{20}|(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t}|\right]
\end{aligned}$$

(since $|(P_x \cap P_{v_t}) \setminus U_{x,v}| \geq \frac{2}{3}|P_x \cap P_{v_t}|$)

$$\begin{aligned}
&\leq \Pr\left[\sum_{i \in (P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t}} X_i \geq \frac{17}{20}|(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t}|\right] \\
&\leq \exp\left(-\frac{9}{200}|(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t}|\right)
\end{aligned}$$

(by the Chernoff bound, since $\mathbb{E}\sum_{\dots} X_i$ is $\frac{14}{20}|(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t}|$)

$$\begin{aligned}
&\leq \exp\left(-\frac{3}{100}|P_x \cap P_{v_t}|\right) \qquad \text{since } |(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t}| \geq \frac{2}{3}|P_x \cap P_{v_t}| \\
&\leq \exp\left(-\frac{3s^2}{200C}\right) = \exp\left(-\frac{3072}{25}\log(n/\delta)\right) < \frac{1}{2n}\,.
\end{aligned}$$

And for Eq. 5.10:

$$\begin{aligned}
\Pr\left[|(P_x \cap P_{v_t}) \setminus F| \geq \frac{1}{3}|P_x \cap P_{v_t}|\right] &= \Pr\left[\sum_{i \in (P_x \cap P_{v_t})} X_i \leq \frac{2}{3}|P_x \cap P_{v,t}|\right] \\
&\leq \exp\left(-2\left(\frac{7}{10} - \frac{2}{3}\right)^2 |P_x \cap P_{v,t}|\right) \\
&\leq \exp\left(-\frac{1}{450}|P_y \cap P_v|\right) = \exp\left(-\frac{s^2}{900C}\right) \\
&= \exp\left(-\frac{2048}{225}\log(n/\delta)\right) < \frac{1}{2n}\,.
\end{aligned}$$

Applying a union bound for the complements of Eq. 5.10 and Eq. 5.9 over all applicable z, we find that both conditions hold with positive probability, so a suitable F exists.

Now that F has been chosen, we will select the colors for the edges from v_t to each $x \in A(v_t)$ from the set $(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F$. Since no colors in F are chosen, for any $z \in B(v_t)$, U_{t,z,v_t} will not contain any element of F; thus $|U_{t,z,v_t}| \leq |(P_x \cap P_{v_t}) \setminus F| \leq \frac{1}{3}|P_x \cap P_{v_t}|$.

Construct the bipartite graph J between $A(v_t)$ and P_{v_t}, where $x \in A(v_t)$ has an edge to each of the colors in $(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F$. We claim there is an $A(v_t)$-saturating matching M of J; given this matching, we assign to edge $\{x, v_t\}$ its matched color $M(x)$. For all $x \in A(v_t)$, we will have $M(x) \in (P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F$; since $M(x) \notin U_{t-1,x,v_t}$, the edge color for $\{x, v_t\}$ will not have been used before by any edge adjacent to x. As the matching assigns a unique color to each edge, the edge coloring constraint will also be satisfied for v_t.

To prove there exists a matching M in J, we verify that Hall's condition holds. For any subset S of the vertices in $A(v_t)$, we want to show that

$$\left| \bigcup_{x \in S} ((P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F) \right| \geq |S|$$

The construction of H ensures that $A(v_t)$ and all subsets thereof satisfy the conditions for the second part of Property U. (Specifically, $|A(v_t)| \leq s\Delta/C$, and that for any $b \in V$ there is at most one j for which $(b, j) \in A(v, t)$.) By this property, we are guaranteed that for all $T \subseteq P_{v_t}$ of size $k - 1$, that there exists some $x \in S$ for which $|P_x \cap T| \leq \frac{1}{10}|P_x \cap P_{v_t}$. If Hall's condition does not hold for S, then there must exist some $T \subseteq P_{v,t}$ for which:

$$\begin{aligned}
& & \bigcup_{x \in S} ((P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F) \subseteq T \\
\implies & \quad \forall x \in S: & (P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F \subseteq T \\
\implies & \quad \forall x \in S: & (P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F \subseteq T \cap P_x \\
\implies & \quad \forall x \in S: & |(P_x \cap P_{v_t}) \setminus U_{t-1,x,v_t} \setminus F| \leq |T \cap P_x| \\
\implies & \quad \forall x \in S: & \frac{1}{10}|P_x \cap P_{v_t}| \leq |T \cap P_x| . \qquad \text{by Eq. 5.9}
\end{aligned}$$

But by the second part of Property U, there must exist an $x \in S$ for which $\frac{1}{10}|P_x \cap P_{v_t}| > |T \cap P_x|$; this is a contradiction, so it follows that Hall's condition does hold for S. Since Hall's condition holds for all $S \subseteq A(v_t)$, J will contain a matching, and step t will ensure Invariant W holds for step $t + 1$.

By induction, it follows that Invariant W holds for all $t \in [n]$, and thus that the process to color the edges of the graph will always work. □

At the core of the algorithm used by Theorem 5.5.8 will be an algorithm for partial coloring of input streams. We will prove that this inner algorithm Algorithm 5.5.4 works for a specific class of edge arrival streams. These are categorized by a Property Z, which is closely linked to the way the free color tracker (Listing 5.5.2) refreshes its pool of colors.

Definition 5.5.6. For each edge $\{u,v\}$ that arrives at time t, let $d_{u,t}$ and $d_{v,t}$ be the degrees of u and v respectively in the multigraph formed by all stream edges up to t. The edges adjacent to each $x \in V$ are assigned to blocks depending on the degree of x after they were added; thus edge $\{x,y\}$ arriving at time t is assigned to block number $b_{x,t} := \lceil d_{x,t} \cdot \frac{C}{s\Delta} \rceil$. Note that $b_{x,t} \in [C/s]$. The stream satisfies Property Z when, for all $v \in V$, $i \in [C/s]$, and $w \in V \setminus \{v\}$, the stream contains at most one edge $\{v,w\}$, added at time t, for which $b_{v,t} = i$.

Algorithm 5.5.4 Partial coloring algorithm: (1/3)-partial $O(\Delta)$ edge coloring for graph edge arrival streams satisfying Property Z, plus reference counting

```
    **Input**: Stream of edge arrivals $n$-vertex graph $G = (V, E)$.
    Assume $\Delta$ is a power of two

    **Initialize($\ell$,$\xi$,$\Delta$,$C$,$(\sigma_v)_{v \in V}$, $s$, $R$):**
    Input $\Delta$ is the maximum degree of the input graph stream
    Input $C$ the number of colors this sketch will use
    Input $s$ is block size parameter, and $(\sigma_v)_{v \in V}$ are $s$-wise almost independent permutations
    Input $R$ is a reference counted pool of edges
    Each edge $e \in R$ will have associated counter $M_e^{(\ell)} \in [2^\ell]$ and color class $\chi_e^{(\ell)} \in \{0, \dots, \lceil \log_{3/2} \Delta^{(\ell)} \rceil\} \times [C^{(\ell)}]$
 1: for $v \in V$ do
 2:     $F_v \leftarrow$ InitFreeTracker($C, s, \Delta, \sigma_v$).        ▷ Also referred to as: $F_v^{(\ell,\xi)}$

    **Process**(edge $\{x,y\}$) $\to$ **Option⟨color⟩** $\in \{\perp\} \cup [C]$
 3: if $F_x \cap F_y \neq \emptyset$ then
 4:     Choose $c \in F_x \cap F_y$ arbitrarily
 5:     $F_x$.RemoveAndUpdate(c, $\{x,y\}$)        ▷ This will increase $\{x,y\}$'s refcount in $R$
 6:     $F_y$.RemoveAndUpdate(c, $\{x,y\}$)
 7:     $\chi_{\{x,y\}}^{(\ell)} \leftarrow (\xi, c)$
 8:     $M_{\{x,y\}}^{(\ell)} \leftarrow 1$
 9:     return color $c$
10: else
11:     return $\perp$
```

Lemma 5.5.7. *Algorithm 5.5.4 properly edge-colors a $\geq 1/3$ fraction of the edges in its input stream, if the permutations it is given are good according to Lemma 5.5.5, and the input stream satisfies Property Z.*

Proof of Lemma 5.5.7. Say that the input stream for Algorithm 5.5.4 satisfies Property Z. To each edge $\{x,y\}$, arriving at time t, we can associate a set of possible colors $P_{x,b_{x,t}} \cap P_{y,b_{y,t}}$, where $P_{x,i} := \sigma_x[(i-1)s + [s]]$ indicates the ith set of colors used by the free color tracker F_x. Of course, as the algorithm progresses some of the colors in $P_{x,b_{x,t}} \cap P_{y,b_{y,t}}$ may be used by other edges adjacent to x and y; the color trackers F_x and F_y precisely record these.

Let H be the simple graph on $V \times [C/s]$ formed by mapping each edge $\{x, y\}$ arriving in the input stream at time t to the edge $\{(x, b_{x,t}), (y, b_{y,t})\}$. Because the stream satisfies Property Z, for any u, i, v, there is at most one j for which $\{(u, i), (v, j)\}$ is in H. Thus, by Lemma 5.5.5, with probability $\geq 1 - \delta$ over randomly chosen advice, the permutations $(\sigma_v)_{v \in V}$ are good, and there exists an edge coloring χ of H where each edge $\{(x, i), (y, j)\}$ is given a color from $P_{x,i} \cap P_{y,j}$. This implies that, *if* the color chosen at Line 4 were to exactly match the color from χ at each step, it would be possible to for the first layer to assign a color to every edge.

However, Line 4, when processing edge $\{x, y\}$ at time t, chooses a color arbitrarily from the set of available colors in $P_{x,b_{x,t}} \cap P_{y,b_{y,t}}$. As a result, the algorithm may select colors so that at some point, a given edge has $F_x \cap F_y = \emptyset$ on Line 3, and cannot be colored. We claim that nevertheless, it will color a $\geq \frac{1}{3}$ fraction of all input edges. Consider any fixed input graph stream of length T for Algorithm 5.5.4, whose edges form multiset E. Consider a run of this algorithm on the stream. At each time t, let $\rho_t : E \mapsto [C] \cup \{\perp\}$ indicate the partial coloring produced by the stream after t edges were processed. Let $\chi : E \mapsto [C]$ be the coloring produced by Lemma 5.5.5. Call an edge e "good" at time t if $\rho_t(e) = \perp$ and it is possible to assign color $\chi(e)$ to e. (In other words, there is no edge f incident on one of e's endpoints for which $\rho_{t-1}(f) = \chi(e)$.) Initially, all edges in E are good. Each time t that the algorithm processes an edge $\{u, v\}$, it will either fail to color the edge, or set $\rho_t(\{u, v\}) = c$ for some color c. If $c = \chi(\{u, v\})$, then the number of "good" edges will be reduced by 1, because χ is a valid edge coloring. If $c \neq \chi(\{u, v\})$, then the number of "good" edges will be reduced by 3; $\{u, v\}$ will no longer be good, and there are at most two edges f that are incident to either u or v and have $\chi(f) = c$. If the algorithm fails to color edge $\{u, v\}$, then this means $\chi(\{u, v\})$ was not available (because some edge incident on u or v used that color); so in all cases, after the algorithm processes an edge, it will no longer be "good". Thus, at the end of the stream, there will be no "good" edges remaining; and since the number of "good" edges is reduced by at most 3 per edge that was colored, the number of colored edges must be at least $|E|/3$. □

Finally, we state and prove the formal version of Theorem 5.5.8.

Theorem 5.5.8. *There is a deterministic algorithm for online $O(\Delta(\log \Delta)^2)$ edge coloring in edge arrival streams for multigraphs, using $O(n\sqrt{\Delta}(\log n)^{2.5}(\log \Delta)^3)$ bits of space, and an advice string of $\widetilde{O}(n\sqrt{\Delta})$ bits, which works for all inputs. (By picking a uniformly random advice string, the same algorithm can alternatively be used as a robust algorithm with $1/\operatorname{poly}(n)$ error; the advice can also be verified and computed in exponential time.)*

Proof of Theorem 5.5.8. We claim that Algorithm 5.5.6 satisfies the conditions of the theorem. This algorithm uses an advice string of length $\widetilde{O}(n\sqrt{\Delta})$, for which we do not know of an efficient polynomial time construction. $\delta \in (0,1)$ is a parameter which gives, if the advice string is chosen uniformly at random, an upper bound on the probability that the advice string does not work for all possible inputs. If we ran this algorithm with a random advice string, it would be robust to adversarially generated inputs, with failure probability $\leq \delta$.

Algorithm 5.5.6 runs $O(\log \Delta)$ instances of an inner algorithm, Algorithm 5.5.5, which is designed to give correct edge colorings for graph streams with a specific low-repetition guarantee: that for any vertex $v \in V$, if one considers the sequence of edges adjacent to v in stream order, and splits them into $O(\sqrt{\Delta / \log n})$ contiguous lists, that no lists will include a given edge more than once. To handle edges that are more commonly repeated, Algorithm 5.5.6 will keep track of all edges which, if added, might violate the guarantee, and send them to another instance of Algorithm 5.5.5 which handles graph streams where no contiguous lists of edges adjacent to a vertex includes a given edge twice; and if an edge in the stream might violate that condition, the algorithm sends it to another copy of Algorithm 5.5.5, and so on.

We will first prove that the inner algorithm, Algorithm 5.5.5, works. This algorithm runs a number of instances of Algorithm 5.5.4, which perform a greedy partial coloring of the input stream, with constraints on the set of colors that it can use for any edge, as per Lemma 5.5.5. This greedy-coloring can be performed in $O(n\sqrt{\Delta})$ space, and by Lemma 5.5.7 is guaranteed to color at least $1/3$ of the edges in the input stream. By sending all edges which the greedy procedure did not color to an independent greedy coloring instance, the number of uncolored edges can be reduced further; after $O(\log \Delta)$ iterations of this, an $O(1/\Delta)$ fraction of the input stream has not been colored; this part of the stream can be stored entirely and colored with $2\Delta - 1$ colors.

In Algorithm 5.5.5, the for loop at Line 8 sends the edge $\{x, y\}$ being processed to each of the $\left\lceil \log_{3/2} \Delta^{(\ell)} \right\rceil$ instances of Algorithm 5.5.4, until either one of them assigns a color to the edge, or all the instances fail to color the edge. Since each instance is guaranteed to color a $\geq 1/3$ fraction of the edges it processes, only a $(2/3)^{\left\lceil \log_{3/2} \Delta^{(\ell)} \right\rceil} \leq \frac{1}{\Delta^{(\ell)}}$ fraction of the edges received by Algorithm 5.5.5 will reach Line 12 of the algorithm and be stored in $D^{(\ell)}$; since there are only $O(n)$ such edges, storing them does not significantly affect the space usage of the algorithm. These edges will be greedily colored using a fresh set of colors.

To handle general multigraphs, Algorithm 5.5.6 divides the stream into a series of levels, for ℓ from 0 to $\log \Delta$. The higher levels process very common edges, which are assigned blocks of 2^ℓ colors for layer ℓ. Each level uses an instance of Algorithm 5.5.5 to assign color blocks for the edges it receives. If a copy e of a given edge $\{u, v\}$ arrives, and the color block for $\{u, v\}$ is not full, then the e will be assigned the next available color in the block. The specific scheme, as we shall show, ensures that every edge is either colored from an existing color block, or passed to an instance of Algorithm 5.5.5; and in the latter case, ensures that Property Z holds for the stream sent to Algorithm 5.5.5.

Algorithm 5.5.5 Inner algorithm: $O(\Delta \log \Delta)$ edge coloring for graph edge arrival streams which have certain substreams satisfying Property Z, plus reference counting

```
Input: Stream of edge arrivals $n$-vertex graph $G = (V, E)$.
Assume $\Delta$ is a power of two
Superscript $(\ell)$ indicates the *level* of the algorithm

Initialize($\ell$, $\Delta^{(\ell)}$, $R$):
Input $\Delta^{(\ell)}$ is the maximum degree of the input graph
Let $C^{(\ell)} = 32\Delta^{(\ell)}$
1:  $D^{(\ell)} \leftarrow \emptyset$ be a set of $O(n \log n)$ "overflow" edges
2:  if $\Delta^{(\ell)} \geq 256 \log(n/\delta)$ then        ▷ Condition for Lemma 5.5.5 to apply
3:      Let $s^{(\ell)}$ satisfy constraints of Lemma 5.5.5
4:      Advice: $\{\sigma_v^{(\ell)}\}_{v \in V}$ are permutations over $[C^{(\ell)}]$, "good" for Lemma 5.5.5
5:      for each layer $\xi \in [\lceil \log_{3/2} \Delta^{(\ell)} \rceil]$ do
6:          $\mathfrak{I}^{(\ell,\xi)} \leftarrow$ INITIALIZE($\ell$,$\xi$,$\Delta$,$C$,$s$,$\{\sigma_v^{(\ell)}\}_{v \in V}$,$R$) from Algorithm 5.5.4

Process(edge $\{x, y\}$) $\rightarrow$ color $\in \left[\left\lceil \log_{3/2} \Delta^{(\ell)} \right\rceil\right] \times \left[C^{(\ell)}\right]$
7:  if $\Delta^{(\ell)} \geq 256 \log(n/\delta)$ then
8:      for $\xi \in \left[\left\lceil \log_{3/2} \Delta^{(\ell)} \right\rceil\right]$ do
9:          Let $c \leftarrow \mathfrak{I}^{(\ell,\xi)}$.PROCESS($\{x, y\}$) from Algorithm 5.5.4
10:         if $c \neq \perp$ then
11:             return color $(\xi, c)$
12: $D^{(\ell)} \leftarrow D^{(\ell)} \cup \{\{x, y\}\}$
13: Increase reference count for $\{x, y\}$ in $R$
14: Greedily pick a color class $c \in [C^{(\ell)}]$ not used by any edge in $D^{(\ell)}$ adjacent to $x$ or $y$
15: $\chi_{\{x,y\}}^{(\ell)} \leftarrow (0, c)$
16: $M_{\{x,y\}}^{(\ell)} \leftarrow 1$
17: return color $(0, c)$
```

Algorithm 5.5.6 maintains a global reference counted pool R, which keeps track of every edge $\{x, y\}$ that arrives for some amount of time. Each level ℓ can provide references for the edge; $\{x, y\}$ will only be dropped from R if no levels have a reference. At level ℓ, we have two cases, depending on how $\{x, y\}$ was processed by Algorithm 5.5.5. If $\{x, y\}$ was not colored by any layer ξ, it will be stored in $D^{(\ell)}$ for the rest of the stream, and any further copies of that edge will not be sent by Algorithm 5.5.6 to the level ℓ instance of Algorithm 5.5.5. If $\{x, y\}$ was successfully colored by layer ξ, the edge will be recorded until both of the free color trackers $F_x^{(\ell,\xi)}$ and $F_y^{(\ell,\xi)}$ have been refreshed. This only happens if the block numbers $b_{x,t}$ and $b_{y,t}$ of $\{x, y\}$ with respect to the substream received by the layer ξ instance of Algorithm 5.5.4 have increased. Consequently, that substream will satisfy Property Z – if Algorithm 5.5.6 receives a second copy of edge $\{x, y\}$ at time t' while either $b_{x,t'} = b_{x,t}$ or $b_{y,t} = b_{y,t}$, because $\{x, y\}$ will be still be in R, Algorithm 5.5.6 will not

Algorithm 5.5.6 Deterministic algorithm for $O(\Delta(\log\Delta)^2)$ edge coloring for multigraph edge arrival streams

```
Input: Stream of edge arrivals $n$-vertex graph $G = (V, E)$
Assume $\Delta$ is a power of two

Initialize:
1: ▷ The pool R will be a set of “recent” edges for each layer in each level, including edges that are either in $D^{(\ell)}$ or have a reference from one of the $F_x^{(\ell,\xi)}$
2: Let $R \leftarrow \emptyset$ be a reference counted pool of edges.
3: Each edge $e \in R$ will have, per level $\ell$, one associated counter $M_e^{(\ell)} \in [0, 2^\ell]$ and color class $\chi_e^{(\ell)} \in \{0, \ldots, \lceil \log_{3/2} \Delta^{(\ell)} \rceil\} \times [C^{(\ell)}]$. When an edge is added to the pool, set $M_e^{(\ell)} = 0$ for all layers.
4: for each level $\ell$ in $0, \ldots, \log\Delta$ do
5:     Define $\Delta^{(\ell)} = \Delta/2^\ell$
6:     $\mathfrak{K}^{(\ell)} \leftarrow$ INITIALIZE($\ell$,$\Delta^{(\ell)}$) from Algorithm 5.5.5

Process(edge $\{x, y\}$) → color
7: for $\ell$ in $0, \ldots, \log\Delta$ do
8:     if $\{x, y\}$ is in $R$ and $M_{\{x,y\}}^{(\ell)} > 0$ then
9:         if $M_{\{x,y\}}^{(\ell)} = 2^\ell$ then
10:            continue
11:        else
12:            $M_{\{x,y\}}^{(\ell)} \leftarrow M_{\{x,y\}}^{(\ell)} + 1$
13:            Let $(i, j) \leftarrow \chi_{\{x,y\}}^{(\ell)}$
14:            return color $\left(\ell, i, (j-1)2^\ell + M_{\{x,y\}}^{(\ell)}\right)$
15:    else
16:        Let $(\xi, c) \leftarrow \mathfrak{K}^{(\ell)}$.PROCESS($\{x, y\}$) for level $\ell$ algorithm
17:        return color $\left(\ell, \xi, (c-1)2^\ell + 1\right)$
```

send the second copy to the level ℓ instance of Algorithm 5.5.5.

We now check that the level ℓ instance of Algorithm 5.5.5 does not receive a graph stream of degree more than $\Delta^{(\ell)}$. When a copy of an edge $\{u, v\}$ arrives, the loop at Line 7 only continues from level ℓ to to level $\ell + 1$ if $M_{\{u,v\}}^{(\ell)} = 2^\ell$ was true. Thus, for an edge e to be processed by the level ℓ instance of Algorithm 5.5.5, the edge e must have arrived at least $2^0 + 2^1 \ldots + 2^{\ell-1} = 2^\ell - 1$ times before in the stream, and 2^ℓ times in total, since the last time e was dropped from R. Since the maximum degree of the graph is Δ, and an edge must be added $\geq 2^\ell$ times for each time that it is sent to the level ℓ instance of Algorithm 5.5.5, the level ℓ instance will receive at most $\Delta/2^\ell$ edges adjacent to any given vertex.

When $\ell = \log\Delta$ in the for loop, Line 10 will not be executed; because for $M_{\{x,y\}}^{(\ell)}$ to equal $2^\ell = \Delta$, this must be the $\Delta + 1$st copy of edge $\{x, y\}$ to arrive. Thus Algorithm 5.5.6 will assign a color to every edge in the stream.

The total space usage of Algorithm 5.5.6 is dominated by the sets $D^{(\ell)}$, free color trackers $F_v^{(\ell,\xi)}$, and the pool R (along with its linked properties $\chi_e^{(\ell)}, M_e^{(\ell)}$, for each $\ell \in \{0,\dots,\log\Delta\}$.) Over all levels ℓ, layers ξ, and vertices in V, there are $O((\log\Delta)^2 n)$ color trackers, each of which uses $O(s^{(\ell)}\log n + \log\Delta) = O(\sqrt{\Delta\log(n/\delta)}\log n)$ bits of space to store colors and $O(\log n)$-bit references to edges. Each $D^{(\ell)}$ is guaranteed to contain $O(n\log n)$ edges, at most, and needs $O(n(\log n)^2)$ bits of space. Finally, the ℓth level references at most $|D^{(\ell)}| + O(ns^{(\ell)}\log\Delta)$ edges in R, so in total R will have $O(n\log n + n\sqrt{\Delta\log(n/\delta)}\log\Delta)$ edges. Each each needs $O(\log n)$ bits to identify, and there are $O((\log\Delta)^2)$ bits of associated information in the $(\chi_e^{(\ell)}, M_e^{(\ell)})_\ell$. Thus, in total, Algorithm 5.5.6 uses:

$$O(n\sqrt{\Delta\log(n/\delta)}(\log\Delta)^2\log n\log(n\Delta))\,.$$

bits of space.

If the advice $(\sigma_v^{(\ell)})_{v\in V,\ell\in\{0,\dots,\log\Delta\}}$ was chosen randomly using Lemma 5.7.1, then we would need $O(s\operatorname{poly}(\log\Delta,\log n))$ truly random bits per permutation for a level of accuracy ($\le 1/\operatorname{poly}(n)$ total variation variation distance from uniformity) under which the proof of Lemma 5.5.5 works. At $\delta = 1/2$, this is $O(n\sqrt{\Delta}\operatorname{poly}(\log\Delta,\log n)$ bits. Given exponential time, the advice can also be computed on demand, since checking that Property U from the proof of Lemma 5.5.5 can be done in exponential time.

□

5.6 A lower bound for deterministic edge coloring

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

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

First, we prove a lemma counting the number of $(\Delta, 1)$-biregular[6] graphs which are compatible with an array of additional constraints on the colors of edges.

Lemma 5.6.1. *Let B be a set of n vertices; let Δ be an integer with $\Delta \mid n$, and let $C \in [\Delta, 2\Delta - 1]$ be another integer. Define $\beta := C/\Delta$. Consider the case where $(2 - \beta)n \geq 32C$.*

For each $v \in B$, say we have a nonempty set $S_v \subseteq [C]$ of possible colors, where $\sum_{i \in B} |S_i| \leq \beta n$. If G is a uniformly random $(\Delta, 1)$-biregular graph between a set A of size n/Δ and B, then the probability $p(n, \Delta, \beta)$ that G has a valid edge coloring where each edge (a, b) is given a color from S_b is:

$$p(n, \Delta, \beta) \leq \exp(-\frac{1}{2^{13}}(2 - \beta)^3 n)\,. \tag{5.11}$$

Proof of Lemma 5.6.1. The graph G can be interpreted as a random partition of B into sets $P_1, \ldots, P_{n/\Delta}$, where all sets P_i have size Δ. Note that P_1 is a uniformly random subset of size Δ in B, P_2 is a uniformly random subset of size Δ in $B \setminus P_1$, and so on. For each $i \in [n/\Delta]$, let C_i be the event that there is a P_i-saturating matching in the bipartite graph between P_i and $[C]$, where each $v \in P$ is adjacent to all $c \in S_v$. Let $\gamma = 2 - \beta$. Then we have:

$$p(n, \Delta, \beta) \leq \prod_{i=1}^{\lceil \frac{1}{2}\gamma n/\Delta \rceil} \Pr[C_i | C_1, \ldots, C_{i-1}]\,. \tag{5.12}$$

We will prove Eq. 5.11 by proving upper bounds on $\Pr[C_i | C_1, \ldots, C_{i-1}]$, for all $i \in [\lceil \frac{1}{2}\gamma n/\Delta \rceil]$, and then applying Eq. 5.12.

By Markov's inequality, the fraction of vertices in B for which $|S_v| - 1 \geq 1$ is $\leq \beta - 1 = 1 - \gamma$, so $\Pr_{v \sim B}[|S_v| = 1] \geq \gamma$. For each $i \in [n/\Delta]$, let $T_i = B \setminus \bigcup_{j<i} P_j$. Then for all $i \leq \lceil \frac{1}{2}\gamma n/\Delta \rceil$:

$$\frac{|\{v \in T_i : |S_v| = 1\}|}{|T_i|} \geq \frac{|\{v \in B : |S_v| = 1\}| - \Delta(i-1)}{|B| - \Delta i} \geq \frac{\gamma n - (i-1)\Delta}{n - \Delta(i-1)} \geq \frac{\gamma n - \frac{1}{2}\gamma n}{n - \frac{1}{2}\gamma n} \geq \frac{1}{2}\gamma\,.$$

Consequently, conditioned on $P_1, \ldots, P_{i-1}$, the set P_i will be drawn uniformly at random from a set T_i of vertices for which at least a $\gamma/2$ fraction have singleton color sets (have $|S_v| = 1$).

For a given i, we remark that if P_i contains two vertices v, w for which $|S_v| = |S_w| = 1$ and $S_v = S_w$, then it is not possible to match the vertices in P_i to colors, as v and w would conflict. Let us bound the probability that this occurs. Let $\hat{n} = |T_i|$; note that this is $\geq n/2$. To make the distribution of singleton sets drawn from $\{S_v\}_{v \in T_i}$ appear more uniform, we construct C disjoint sets

[6] Bipartite graph on $A \sqcup B$ where vertices in A have degree Δ, and vertices in B have degree 1.

$L_1, \ldots, L_C$ in T_i, so that for each L_i, the associated color sets are all a singleton $|\bigcup_{j \in L_i} S_j| = 1$; and for which $|L_i| \geq \gamma\hat{n}/4C$. (If each singleton color set were equally likely, we could get $|L_i| \geq \gamma\hat{n}/2C$ – but it is possible that $\{1\}$ is rare, while $\{2\}$ more common than average. One way to construct the $L_1, \ldots, L_C$ is by iteratively removing sets of $\gamma\hat{n}/4C$ vertices from T_i whose associated color sets are all the same singleton set.)

For each $j \in T_i$, let X_j be the indicator random variable for the event that $j \in P_i$. For each $k \in [C]$, let $Y_k = \sum_{j \in L_k} X_j$. Since the random variables $\{X_j\}_{j \in T_i}$ are negatively associated, sums of disjoint sets of them, the $\{Y_k\}_{k \in [C]}$, are also negatively associated. (See Definition 2.3.2.) We have:

$$\begin{aligned}
&\Pr[P_i \text{ has no two elements from same } L_k] \\
&\quad \leq \Pr[\bigwedge_{k \in [C]} \{Y_k \leq 1\}] \\
&\quad \leq \prod_{k \in [C]} \Pr[Y_k \leq 1]\,. \qquad \text{since negative association} \implies \text{negative orthant dependence}
\end{aligned}$$

We calculate $\Pr[Y_k \leq 1]$ exactly, and then prove an upper bound on it.

$$\begin{aligned}
\Pr[Y_k \leq 1] &= \frac{\binom{\hat{n}-|L_k|}{\Delta} + |L_k|\binom{\hat{n}-|L_k|}{\Delta-1}}{\binom{\hat{n}}{\Delta}} \\
&= \frac{\frac{\hat{n}-|L_k|-\Delta+1}{\Delta}\binom{\hat{n}-|L_k|}{\Delta-1} + |L_k|\binom{\hat{n}-|L_k|}{\Delta-1}}{\frac{\hat{n}}{\Delta}\binom{\hat{n}-1}{\Delta-1}} \\
&= \frac{\frac{\hat{n}-|L_k|-\Delta+1}{\Delta} + |L_k|}{\hat{n}/\Delta} \cdot \frac{\binom{\hat{n}-|L_k|}{\Delta-1}}{\binom{\hat{n}-1}{\Delta-1}} \\
&\leq \frac{\frac{\hat{n}-|L_k|-\Delta+1}{\Delta} + |L_k|}{\hat{n}/\Delta} \cdot \left(\frac{\hat{n}-|L_k|}{\hat{n}-1}\right)^{\Delta-1} \qquad \text{since } \binom{a}{c}/\binom{b}{c} \leq (a/b)^c \text{ if } c \leq a \leq b \\
&= \left(1 + \frac{|L_k|-1}{\hat{n}}(\Delta-1)\right)\left(1 - \frac{|L_k|-1}{\hat{n}-1}\right)^{\Delta-1}
\end{aligned}$$

(The next two inequalities hold because $1 - x \leq \exp(x)$ and $\ln(1+x) \leq x - x^2/4$ for $x \leq 1$.)

$$\begin{aligned}
&\leq \exp\left(\ln\left(1 + \frac{|L_k|-1}{\hat{n}}(\Delta-1)\right) - \frac{|L_k|-1}{\hat{n}-1}(\Delta-1)\right) \\
&\leq \exp\left(\frac{|L_k|-1}{\hat{n}}(\Delta-1) - \frac{1}{4}\left(\frac{|L_k|-1}{\hat{n}}(\Delta-1)\right)^2 - \frac{|L_k|-1}{\hat{n}-1}(\Delta-1)\right) \\
&\leq \exp\left(-\frac{(|L_k|-1)(\Delta-1)}{\hat{n}(\hat{n}-1)} - \frac{1}{4}\left(\frac{|L_k|-1}{\hat{n}}(\Delta-1)\right)^2\right) \\
&\leq \exp\left(-\frac{1}{4}\frac{\gamma^2}{2^{10}}\right) = \exp(-\gamma^2/2^{12})\,.
\end{aligned}$$

For the last inequality, we used the fact that $(|L_k|-1)(\Delta-1)/\hat{n} \geq (\gamma\hat{n}/4C-1)(\Delta-1)/\hat{n} \geq \gamma/32C$.

Having bounded $\Pr[Y_k \leq 1]$, it follows that:

$$p(n,\Delta,\beta) \leq \prod_{i=1}^{\lceil \frac{1}{2}\gamma n/\Delta \rceil} \Pr[C_i | C_1, \ldots, C_{i-1}] \leq \left(\exp\left(-\gamma^2/2^{12}\right)^C\right)^{\lceil \frac{1}{2}\gamma n/\Delta \rceil}$$
$$\leq \exp\left(-\gamma^2/2^{12} \cdot C \cdot \left\lceil \frac{1}{2}\gamma n/\Delta \right\rceil\right) \leq \exp(-\gamma^3 n/2^{13}) \,. \qquad \square$$

Theorem 5.6.2. *For all $\beta \in (1,2)$, and integers n, Δ satisfying $\Delta \leq n(2-\beta)/(64\beta)$, every deterministic online streaming algorithm for edge-coloring that uses $\beta\Delta$ colors requires $\Omega((2-\beta)^3 n)$ bits of space. In particular, $(2\Delta-1)$-edge-coloring requires $\Omega(n/\Delta^3)$ space.*

Proof of Theorem 5.6.2. Say we have an algorithm $\mathcal{A}$ to provide an online $\beta\Delta$-edge-coloring of an input stream, presented in one-sided vertex arrival order, using S bits of space. We assume that $\Delta | n$; if this is not the case, we can reduce n to the nearest multiple of Δ, weakening our final lower bound by at most a factor of 2. With the algorithm, we can implement a protocol for a Δ-player one-way communication game in which each message uses $\leq S$ bits. We will then prove a communication lower bound for this game.

Specifically, let $P_1, \ldots, P_\Delta$ be the players of the game. Let $A_1, \ldots, A_\Delta$ and B be sets of vertices, where for each $i \in [\Delta]$, $|A_i| = n/\Delta$, and $|B| = n$. For each $i \in [\Delta]$, the player P_i is given a $(\Delta, 1)$-biregular graph G_i from A_i to B. Player P_1 starts the communication game by outputing an edge coloring χ_1 of G_1, using colors in $[\beta\Delta]$; and then it sends a message m_1 to Player P_2. For each $i \in \{2, \ldots, \Delta\}$, the player P_i will receive a message m_{i-1} from its predecessor, output an edge coloring χ_i of G_1 which is compatible with the edge colorings $\chi_1, \ldots, \chi_{i-1}$ made by the earlier players, and then (if $i < \Delta$) send a message m_i to the next player.

The conversion from an algorithm $\mathcal{A}$ to a protocol for this game is straightforward; P_1 initializes an instance A of $\mathcal{A}$, uses it to process G_1 in arbitrary order, and reports the colors the algorithm output; then it encodes the state of the instance A as an S-bit message m_1. P_2 receives this message, and uses it to continue running the instance A, this time having it process G_2; P_2 outputs the results, and sends the new state of A to P_3 as m_2. The players continue in this way until P_Δ produces output.

For each $i \in \{1, \ldots, \Delta-1\}$, and message m_i, we define $\mathcal{S}_{m_i} = (S_{m_i,v})_{v \in B}$. Here $S_{m_i,v}$ is the set of all colors which players $P_1, \ldots, P_i$ could have assigned to edges incident on v for executions of the protocol in which P_i sent message m_i. If player P_{i+1} receives message m_i, the coloring χ_{i+1} that it outputs must be disjoint from $\mathcal{S}_{m_i}$; specifically, if we view χ_{i+1} as a vector in $[\beta\Delta]^B$ whose vth entry gives the color assigned to the edge incident to vertex v, then $\forall v \in B : \chi_{i+1,v} \notin S_{m_i,v}$. If this were not the case, and there was a vertex x for which $\chi_{i+1,x} \in S_{m_i,x}$, then there would be an execution of the protocol on which some player P_j output color $\chi_{i+1,x}$ for an edge incident on x, later P_i sent message m_i, and now P_{i+1}'s assignment of $\chi_{i+1,x}$ to the edge incident on x violates the edge coloring constraint.

Let $p(n,\Delta,\beta)$ be the probability from Lemma 5.6.1. We claim that there exists an input for which some player must send a message with more than $\log(1/p(n,\Delta,\beta))$ bits. If this is not the case, then we shall construct an input on which the protocol must give an incorrect output, a contradiction.

Let M_1 be the set of all messages that player P_1 can send. Let H be chosen uniformly at random from the set of $(\Delta,1)$-regular bipartite graphs from A_1 to B, and let $m_1(H)$ be the message P_1 would send if $G_1 = H$. Then, if for all $m \in M_1$, we were to have $\sum_{v\in B} |S_{m_i,v}| \le \beta n$,

$$1 = \sum_{m\in M_1} \Pr[m = m_1(H)] \le |M_1| p(n,\Delta,\beta)\,.$$

But since we have assumed messages in M_1 need $< \log(1/p(n,\Delta,\beta))$ bits, and hence $|M_1| < 1/p(n,\Delta,\beta)$, the above equation would imply $1 < 1$; thus there must be some $m_1^\star \in M_1$ for which $\sum_{v\in B} S_{m_i,v} \ge \beta n$. Let $\mathcal{G}_1$ be the set of graphs for which $H \in \mathcal{G}_1 \iff m_1(H) = m_1^\star$; then on being given any graph in $\mathcal{G}_1$, player P_1 will output $m_1^\star$.

We will now iterate over $i \in \{2,\ldots,\Delta-1\}$ and build a sequence of messages $m_1^\star, m_2^\star, \ldots, m_\Delta^\star$, along with sets of input graphs $\mathcal{G}_1,\ldots,\mathcal{G}_{\Delta-1}$ on which the protocol will send these messages. For each $i \in [\Delta]$, define $\mathcal{T}_{m,i,i} = (T_{v,m,i})_{v\in B}$, where $T_{v,m,i} := S_m \setminus S_{m_{i-1}^\star,i}$. Any coloring χ_i that P_i outputs which is compatible with all inputs leading to $m_i^\star$ will satisfy $\chi_{i,v} \in T_{v,m,i}$. (If $\chi_{i,v} \in S_{m_{i-1}^\star,i}$, then as noted above there is a set of inputs where this will violate the edge coloring constraint for v.) As argued for M_1, there must be some message $m_i^\star \in M_i$ for which $\sum_{v\in B} |T_{v,m,i}| \ge \beta n$.

Finally, for each $v \in B$, define $R_v = [\beta\Delta] \setminus S_{m_{\Delta-1}^\star,v}$. Since $S_{m_{\Delta-1}^\star,v} = \sqcup_{i=1}^{\Delta-1} T_{v,m_i^\star,i}$ we will have $|R_v| \le \beta\Delta - \beta(\Delta-1) = \beta$. On receiving $m_{\Delta-1}^\star$, player P_Δ can only assign edge colors so that edges incident on v use colors in R_v; for any other color, there is a input which uses it and which makes $P_{\Delta-1}$ send $m_{\Delta-1}^\star$. If G_Δ were chosen uniformly at random from its set of possible values, then the probability that G_Δ has an edge coloring compatible with $\{R_v\}_{v\in m}$ is $\le p(n,\Delta,\beta)$. Since this is < 1, there must exist a specific graph $G_\Delta^\dagger$ on which the protocol uses a color not in R_v for some $v \in B$. We have thus shown that if the protocol always uses fewer than $\log(1/p(n,\Delta,\beta))$ bits for its messages, it will give incorrect outputs for some input.

We conclude:

$$S \ge \log \frac{1}{p(n,\Delta,\beta)} = \Omega((2-\beta)^3 n)\,.$$

□

Remark 5.6.3. In some ways, the proof of Theorem 5.6.2 is similar to the deterministic lower bound proof for MissingItemFinding, Theorem 3.5.1; it also has, at its core, an iterated counting argument (specifically Lemma 3.3.2), like Lemma 5.6.1. However, here the lower bound appears rather weak; this may be a consequence of the way in which the proof of Theorem 5.6.2 does not use the fact that later players, like P_Δ, only have a limited amount of information about *all* the preceding players; the proof would still hold if P_Δ exactly knew the messages from all players $P_1, \ldots, P_{\Delta-1}$, which is an unrealistic assumption.

5.7 Details of constructing random permutations

By [Mor13] plus some algebra, for any $\varepsilon > 0$, a sequence of $O\left(d^3 + d\ln(\frac{1}{\varepsilon})\right)$ Thorp shuffle steps will produce a permutation on $[2^d]$ whose distribution has total variation distance at most ε away from the uniform distribution.[7]

Lemma 5.7.1 (Random permutations through switching networks)**.** *For any C which is a power of 2, there is an explicit construction of an (ε, s)-wise independent random permutation, using $r = O(s(\log C)^4 \log \frac{1}{\varepsilon})$ bits. Furthermore, we can evaluate $\sigma(i)$ and $\sigma^{-1}(i)$ in $O(s(\log C)^4 \log \frac{1}{\varepsilon} \log C)$ time.*

Proof of Lemma 5.7.1. Let $k = O(d^3 + d\ln(1/\varepsilon))$ be the constant for which k Thorp shuffle steps would permute $[C] = [2^d]$ within total variation distance of ε of the uniform distribution over permutations on $[2^d]$.

The switching network $\mathcal{N}$ corresponding to the k Thorp shuffle steps has depth k and uses exactly $kC/2$ gates. Assign each gate a unique number in $[kC/2]$. Then given a uniformly random bit vector $x \in \{0,1\}^{kC/2}$, we evaluate the switching network by having the gate numbered i switch its inputs iff $x_i = 1$. A key property of switching networks is that one can evaluate their action on a single input by only evaluating one gate per layer – for this network, only k gates. Reversing the order in which the layers are applied will produce the inverse of the original permutation. Thus, one can evaluate $\mathcal{N}(i)$ by reading only k entries of x, and similarly for $\mathcal{N}^{-1}(i)$.

Now, say the bits of x are the output of a hash function drawn from a ks-wise independent hash family. (For example, using a family of [WC81], let $h = \lceil \log_2 kC/2 \rceil$, take the family of random polynomials of degree $ks - 1$ inside $\mathbb{F}_{2^h}$, and output the least bit of the output. The polynomial coefficients can be encoded using $ksh = O(ks\log(C))$ bits, and the polynomials evaluated at any point in $O(ksh^2) = O(ks(\log C)^2)$ time.)

[7]While there exist more efficient switching networks that also permute sets whose sizes are not powers of two, we use the result of [Mor13] here because it is simple to work with. The results claimed by [Czu15] might be better, but we could not find the full version of that paper.

If π is a uniformly random permutation on $[C]$, then for all lists of distinct $h_1, \dots, h_s$, and all lists of distinct $j_1, \dots, j_s$, we have

$$\Pr[\bigwedge_{i \in [s]} \pi(h_i) = j_i] = \frac{1}{\prod_{i \in [s]}(C - i + 1)}.$$

Now, let $f : \{0,1\}^{kC/2} \times [C] \mapsto [C]^k$ be the function which maps the gate-controlling vector $x \in \{0,1\}^{kC/2}$ and an input $a \in [C]$ to the path $b_1, \dots, b_k$ that a takes through the switching network $\mathcal{N}$ if its gates are configured according to x. The last node of this path, $f(x,a)_k$ is the output of $\mathcal{N}$ given x and a. Each path $P = (a, b_1, \dots, b_k)$ through the switching network corresponds to a restriction $R_P \in \{0,1,\star\}^{kC/2}$ which has value $\star$ on gates not touched by the path, and for each gate traversed by the path assigns either 0 or 1 depending on whether a straight or switched configuration of the gate is compatible with P. Since all paths through the network have length k, R_P only sets k coordinates. Now, for each pair $(a,b) \in [C]^2$, let

$$\mathcal{F}_{a,b} = \{R_P : P = (a, b_1, \dots, b_{k-1}, b) \text{ is a possible path}\}.$$

Then, for lists $(h_1, \dots, h_s)$, $(j_1, \dots, j_s)$, define

$$\mathcal{K}_{h_1,\dots,h_s,j_1,\dots,j_s} = \{R \in \{0,1,\star\}^{kC/2} : \forall i \in [k], \exists T \in \mathcal{F}_{h_i,j_i} \text{where } R \text{ is a minimal refinement of } T\},$$

i.e., the set of minimal restrictions for vectors in $\{0,1\}^{kC/2}$ which completely determine the paths through the switching network of inputs $(h_1, \dots, h_s)$.

Now, let $Y \in \{0,1\}^{kC/2}$ be ks-wise independent, and $X \in \{0,1\}^{kC/2}$ be fully independent. We have:

$$\begin{aligned}\Pr[\bigwedge_{i \in [s]} f(Y, h_i)_k = j_i] &= \sum_{R \in \mathcal{K}_{h_1,\dots,h_s,j_1,\dots,j_s}} \Pr[Y \text{ compatible with } R] \\ &= \sum_{R \in \mathcal{K}_{h_1,\dots,h_s,j_1,\dots,j_s}} \Pr[X \text{ compatible with } R] \\ &= \Pr[\bigwedge_{i \in [s]} f(X, h_i)_k = j_i],\end{aligned}$$

where the second inequality follows because each restriction $R \in \mathcal{K}_{h_1,\dots,h_s,j_1,\dots,j_s}$ only constrains $sk/2$ coordinates corresponding to the gates on the paths in the switching network from $h_1, \dots, h_s$ to $j_1, \dots, j_s$ that it fixes.

Thus,

$$\frac{1}{2} \sum_{\text{distinct } b_1, \dots, b_k \text{ in } [C],} \left| \Pr\left[\bigwedge_{i \in [s]} \{f(X, h_i)_k = j_i\}\right] - \frac{1}{\prod_{i \in [s]}(C - i + 1)} \right| =$$

$$\frac{1}{2} \sum_{\text{distinct } b_1, \ldots, b_k \text{ in } [C],} \left| \Pr\left[\bigwedge_{i \in [s]} \{ f(Y, h_i)_k = j_i \} \right] - \frac{1}{\prod_{i \in [s]} (C - i + 1)} \right| \leq \varepsilon \,,$$

which proves that the switching network evaluated on X produces outputs that are (ε, s)-wise independent. □

5.8 Conclusion

Several of the algorithms in this chapter rely on the availability of an oracle random string, in order to avoid the space penalty of explicitly storing many independent random permutations. In practice, where we optimistically assume that cryptographic pseudo-random number generators exist, it is straightforward to generate the bits of the oracle random string on demand, ensuring that computationally bounded systems essentially cannot produce hard inputs for the algorithm.

The randomized algorithms Algorithm 5.4.1 and Algorithm 5.5.3 both use the same idea of trying and discarding (making unavailable for future use), either immediately or periodically, a set of fresh colors chosen by iterating over a random permutation. This construction has the downside that, since many colors are discarded, the total number C of colors that the algorithm might use must be large. Instead of discarding colors, a possibly more efficient approach is to retain, for each vertex, a pool of all the colors that were tried but not used; this ensures that colors are only removed from consideration when they have been used. The downside of retaining unused colors is an increased space usage that is harder to prove upper bounds for. We suspect that the following two algorithms, Algorithm 5.8.1 and Algorithm 5.8.2, for one-sided vertex arrival, and edge arrival streams, will use only $\widetilde{O}(n)$ and $\widetilde{O}(n\sqrt{\Delta})$ bits of space with high probability, but have not been able to prove this. The second algorithm in particular is rather similar to an online edge coloring algorithm conjectured to use $\Delta + O(\sqrt{\Delta} \log n)$ colors by [BMN92], in which each edge is assigned a uniformly random color from the set of colors that no edges incident to its endpoints have used.

Algorithm 5.8.1 Randomized algorithm for $(2\Delta - 1)$-edge coloring in the one-sided vertex arrival model, conjectured to use $O(n \log \Delta)$ space w.h.p.

```
Input: Stream of one-sided vertex arrivals on $n$-vertex graph $G = (A \sqcup B)$.
Let $C := 2\Delta - 1$.

Initialize():
1: for $z \in B$ do
2:     $\sigma_z \leftarrow$ uniformly randomly chosen permutation over $[C]$
3:     $h_z \leftarrow 1$
4:     $F_z \leftarrow \emptyset$

Process(vertex $a \in A$, adjacent edges $M_a$)
5: $S \leftarrow \emptyset$
6: for $\{a, b\} \in M_a$, in random order do
7:     while $F_b \subseteq S$ do
8:         $F_b \leftarrow F_b \cup \sigma_b[h_b]$
9:         $h_b \leftarrow h_b + 1$
10:    Let $c$ be random color from $F_b \setminus S$
11:    Assign color $c$ to edge $\{a, b\}$
12:    $F_b \leftarrow F_b \setminus \{c\}$
13:    $S \leftarrow S \cup \{c\}$
```

Algorithm 5.8.2 Randomized algorithm for $(2\Delta - 1)$-edge coloring in the edge arrival model, conjectured to use $O(n\sqrt{\Delta} \log \Delta)$ space w.h.p.

```
Input: Stream of edge arrivals on $n$-vertex graph $G = (V, E)$.
Let $C := 2\Delta - 1$.

Initialize:
1: for $v \in B$ do
2:     $\sigma_v \leftarrow$ uniformly randomly chosen permutation over $[C]$
3:     $h_v \leftarrow 1$
4:     $F_v \leftarrow \emptyset$

Process(edge $\{x, y\}$):
5: while $F_x \cap F_y = \emptyset$ do
6:     $F_x \leftarrow F_x \cup \sigma_x[h_y]$
7:     $h_x \leftarrow h_x + 1$
8:     $F_y \leftarrow F_y \cup \sigma_y[h_y]$
9:     $h_y \leftarrow h_y + 1$
10: Let $c$ be random color from $F_x \cap F_y$
11: Assign color $c$ to edge $\{x, y\}$
12: $F_x \leftarrow F_x \setminus \{c\}$
13: $F_y \leftarrow F_y \setminus \{c\}$
14: $S \leftarrow \emptyset$
```

Bibliography

[AA20] Noga Alon and Sepehr Assadi. Palette sparsification beyond (Δ+1) vertex coloring. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 6:1–6:22, 2020. 116, 118, 119

[ABJ$^+$22] Miklós Ajtai, Vladimir Braverman, T.S. Jayram, Sandeep Silwal, Alec Sun, David P. Woodruff, and Samson Zhou. The white-box adversarial data stream model. In *Proc. 41st ACM Symposium on Principles of Database Systems*, page 15–27, 2022. 4, 13, 15, 18, 75

[ACGS23] Sepehr Assadi, Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Coloring in graph streams via deterministic and adversarially robust algorithms. In *Proc. 42nd ACM Symposium on Principles of Database Systems*, page 141–153, 2023. iii

[ACK19] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for (Δ+ 1) vertex coloring. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 767–786, 2019. 4, 28, 112, 113, 115, 116, 118, 127

[ACKP19] Amir Abboud, Keren Censor-Hillel, Seri Khoury, and Ami Paz. Smaller cuts, higher lower bounds. *CoRR*, abs/1901.01630, 2019. 113, 118, 119

[ACS22] Sepehr Assadi, Andrew Chen, and Glenn Sun. Deterministic graph coloring in the streaming model. In *Proc. 54th Annual ACM Symposium on the Theory of Computing*, pages 261—-274, 2022. 5, 28, 113, 114, 115, 116, 119, 131, 151, 154, 155, 156

[ACSS21] Idan Attias, Edith Cohen, Moshe Shechner, and Uri Stemmer. A framework for adversarial streaming via differential privacy and difference estimators. *CoRR*, abs/2107.14527, 2021. 13

[AGM12] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 459–467, 2012. 118

[AKM22] Sepehr Assadi, Pankaj Kumar, and Parth Mittal. Brooks' theorem in graph streams: a single-pass semi-streaming algorithm for Δ-coloring. In *Proc. 54th Annual ACM Symposium on the Theory of Computing*, pages 234–247, 2022. 4, 113, 119

[AL12] Noga Alon and Shachar Lovett. Almost k-wise vs. k-wise independent permutations, and uniformity for general group actions. In *Proc. 16th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 350–361. Springer, 2012. 167

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. Preliminary version in *Proc. 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996. 12

[AMSZ03] Gagan Aggarwal, Rajeev Motwani, Devavrat Shah, and An Zhu. Switch scheduling via randomized edge coloring. In *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 2003,*, pages 502–512. IEEE, 2003. 159, 160, 164

[ASZZ22] Mohammad Ansari, Mohammad Saneian, and Hamid Zarrabi-Zadeh. Simple Streaming Algorithms for Edge Coloring. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:4, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 6, 160, 161, 162, 164, 166

[BBMU21] Anup Bhattacharya, Arijit Bishnu, Gopinath Mishra, and Anannya Upasana. Even the easiest(?) graph coloring problem is not easy in streaming! In *12th Innovations in Theoretical Computer Science Conference, ITCS 2021, January 6-8, 2021, Virtual Conference*, volume 185 of *LIPIcs*, pages 15:1–15:19, 2021. 118, 119

[BCG20] Suman K. Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 11:1–11:21, 2020. 113, 116, 118, 119, 128

[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 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1–20. SIAM, 2018. 116

[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, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 15:1–15:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 6, 163, 166

[BEEO22] Omri Ben-Eliezer, Talya Eden, and Krzysztof Onak. Adversarially robust streaming via dense-sparse trade-offs. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 214–227, 2022. 115

[BG18] Suman Kalyan Bera and Prantar Ghosh. Coloring in graph streams. *CoRR*, abs/1807.07640, 2018. 116, 118, 127

[BGW21] Sayan Bhattacharya, Fabrizio Grandoni, and David Wajc. Online edge coloring algorithms via the nibble method. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2830–2842. SIAM, 2021. 160, 164

[BHM$^+$21] Vladimir Braverman, Avinatan Hassidim, Yossi Matias, Mariano Schain, Sandeep Silwal, and Samson Zhou. Adversarial robustness of streaming algorithms through importance sampling. *CoRR*, abs/2106.14952, 2021. 117, 118

[BJWY20] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proc. 39th ACM Symposium on Principles of Database Systems*, page 63–80, 2020. 2, 10, 13, 18, 118, 127

[BKKS23] Vladimir Braverman, Robert Krauthgamer, Aditya Krishnan, and Shay Sapir. Lower bounds for pseudo-deterministic counting in a stream. *arXiv preprint arXiv:2303.16287*, 2023. 14

[BKM20] Philipp Bamberger, Fabian Kuhn, and Yannic Maus. Efficient deterministic distributed coloring with small bandwidth. In Yuval Emek and Christian Cachin, editors, *Proc. 39th ACM Symposium on Principles of Distributed Computing*, pages 243–252. ACM, 2020. 116, 142

[BMM12] Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. Online graph edge-coloring in the random-order arrival model. *Theory of Computing*, 8(1):567–595, 2012. 160, 164

[BMN92] Amotz Bar-Noy, Rajeev Motwani, and Joseph Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, 44(5):251–253, 1992. 5, 159, 164, 165, 213

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993. 15

[BS23] Soheil Behnezhad and Mohammad Saneian. Streaming edge coloring with asymptotically optimal colors. *arXiv preprint arXiv:2305.01714*, 2023. 166

[BSS22] Sayan Bhattacharya, Thatchaphol Saranurak, and Pattara Sukprasert. Simple dynamic spanners with near-optimal recourse against an adaptive adversary. *arXiv preprint arXiv:2207.04954*, 2022. 13

[BY20] Omri Ben-Eliezer and Eylon Yogev. The adversarial robustness of sampling. In *Proc. 39th ACM Symposium on Principles of Database Systems*, pages 49–62. ACM, 2020. 118

[CAD$^+$18] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018. 14

[CD07] Charles Colbourne and Jeffrey Dinitz. *Handbook of combinatorial designs*. CRC press Boca Raton, FL, 2007. 38

[CGS22] Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Adversarially robust coloring for graph streams. In *Proc. 13th Conference on Innovations in Theoretical Computer Science*, pages 37:1–37:23, 2022. iii, 157

[CKL$^+$24] Melanie Cambus, Fabian Kuhn, Etna Lindy, Shreyas Pai, and Jara Uitto. A (3+epsilon)-approximate correlation clustering algorithm in dynamic streams. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2861–2880. SIAM, 2024. 13

[CL21] Moses Charikar and Paul Liu. Improved algorithms for edge colouring in the W-streaming model. In *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 181–183. SIAM, 2021. 6, 160, 161, 162, 166, 169

[CLN$^+$22] Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, Moshe Shechner, and Uri Stemmer. On the robustness of countsketch to adaptive inputs. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 4112–4140. PMLR, 2022. 14

[CLP18] Yi-Jun Chang, Wenzheng Li, and Seth Pettie. An optimal distributed (Δ+1)-coloring algorithm? In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 445–456. ACM, 2018. 116

[CMZ23] Shiri Chechik, Doron Mukhtar, and Tianyi Zhang. Streaming edge coloring with subquadratic palette size. *arXiv preprint arXiv:2305.07090*, 2023. 166, 167

[CNSS23] Edith Cohen, Jelani Nelson, Tamás Sarlós, and Uri Stemmer. Tricking the hashing trick: A tight lower bound on the robustness of countsketch to adaptive inputs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 7235–7243, 2023. 14

[Cor23] Graham Cormode. Applications of sketching and pathways to impact. In *Proc. 42nd ACM Symposium on Principles of Database Systems*, pages 5–10, 2023. 2

[CPS19] David Clayton, Christopher Patton, and Thomas Shrimpton. Probabilistic data structures in adversarial environments. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, page 1317–1334, 2019. 14

[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 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1–25. IEEE Computer Society, 2019. 160, 161, 164

[CS23] Amit Chakrabarti and Manuel Stoeckl. When a random tape is not enough: lower bounds for a problem in adversarially robust streaming. *arXiv preprint arXiv:2310.03634*, 2023. iii

[CW79] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979. 129

[Czu15] Artur Czumaj. Random permutations using switching networks. In *Proc. 47th Annual ACM Symposium on the Theory of Computing*, pages 703–712, 2015. 211

[DEMR10] Camil Demetrescu, Bruno Escoffier, Gabriel Moruz, and Andrea Ribichini. Adapting parallel algorithms to the w-stream model, with applications to graph problems. *Theoretical Computer Science*, 411(44):3994–4004, 2010. 166

[DFR06] Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. Trading off space for passes in graph streaming problems. In *Proceedings of the Seventeenth Annual ACM-SIAM*

Symposium on Discrete Algorithms, SODA 2006, pages 714–723. ACM Press, 2006. 166

[DHP$^+$22] Ben Davis, Hamed Hatami, William Pires, Ran Tao, and Hamza Usmani. On public-coin zero-error randomized communication complexity. *Information Processing Letters*, 178:106293, 2022. 12

[EFKM10] Martin R. Ehmsen, Lene M. Favrholdt, Jens S. Kohrt, and Rodica Mihai. Comparing first-fit and next-fit for online edge coloring. *Theor. Comput. Sci.*, 411(16-18):1734–1741, 2010. 166

[EJ01] Thomas Erlebach and Klaus Jansen. The complexity of path coloring and call scheduling. *Theoretical Computer Science*, 255(1):33–50, 2001. 159

[Fei19] Uriel Feige. A randomized strategy in the mirror game. *arXiv preprint arXiv:1901.07809*, 2019. 27, 28

[FM18] Lene M. Favrholdt and Jesper W. Mikkelsen. Online edge coloring of paths and trees with a fixed number of colors. *Acta Informatica*, 55(1):57–80, 2018. 160, 164, 166

[FN03] Lene M. Favrholdt and Morten N. Nielsen. On-line edge-coloring with a fixed number of colors. *Algorithmica*, 35(2):176–191, 2003. 160, 164, 166

[FPUV22] Mia Filic, Kenneth G. Paterson, Anupama Unnikrishnan, and Fernando Virdia. Adversarial correctness and privacy for probabilistic data structures. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, page 1037–1050, 2022. 14

[Fre75] David A. Freedman. On tail probabilities for martingales. *The Annals of Probability*, 3(1):61–68, 1975. 187, 190

[FW23] Ying Feng and David Woodruff. Improved algorithms for white-box adversarial streams. 2023. 15

[GDP05] S. Gandham, M. Dawande, and R. Prakash. Link scheduling in sensor networks: distributed edge coloring revisited. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 4, pages 2492–2501 vol. 4, 2005. 159

[GG11] Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. In *Electron. Colloquium Comput. Complex.*, volume 18, page 136, 2011. 14

[GGMW20] Shafi Goldwasser, Ofer Grossman, Sidhanth Mohanty, and David P. Woodruff. Pseudo-Deterministic Streaming. In *Proc. 20th Conference on Innovations in Theoretical Computer Science*, volume 151, pages 79:1–79:25, 2020. 4, 14, 18

[GGS23] Ofer Grossman, Meghal Gupta, and Mark Sellke. Tight space lower bound for pseudo-deterministic approximate counting. *arXiv preprint arXiv:2304.01438*, 2023. 14

[GK21] Mohsen Ghaffari and Fabian Kuhn. Deterministic distributed vertex coloring: Simpler, faster, and without network decomposition. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science*, pages 1009–1020, 2021. 116, 142

[GPK95] Daniel M Gordon, Oren Patashnik, and Greg Kuperberg. New constructions for covering designs. *Journal of Combinatorial Designs*, 3(4):269–284, 1995. 38

[GS18] Sumegha Garg and Jon Schneider. The Space Complexity of Mirror Games. In *Proc. 10th Conference on Innovations in Theoretical Computer Science*, pages 36:1–36:14, 2018. 27

[GS23] Prantar Ghosh and Manuel Stoeckl. Low-memory algorithms for online and W-streaming edge coloring. *arXiv preprint arXiv:2304.12285*, 2023. iii

[GSS22] Christian Glazik, Jan Schiemann, and Anand Srivastav. A one pass streaming algorithm for finding Euler tours. *Theory of Computing Systems*, pages 1–23, 12 2022. 166

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999. 115

[HJN$^+$11] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. Doug Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58, 2011. 14

[HKM$^+$20] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. 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. 13

[HKNT22] Magnus M. Halldorsson, Fabian Kuhn, Alexandre Nolin, and Tigran Tonayan. Near-optimal distributed degree+1 coloring. In *Proc. 54th Annual ACM Symposium on the Theory of Computing*, pages 450–463, 2022. 113, 119, 142

[Hol81] Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981. 159

[HW13] Moritz Hardt and David P. Woodruff. How robust are linear sketches to adaptive inputs? In *Proc. 45th Annual ACM Symposium on the Theory of Computing*, pages 121–130, 2013. 13

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006. 14

[IR08] Piotr Indyk and Milan Ruzic. Near-optimal sparse recovery in the l1 norm. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 199–207, 2008. 32

[JP83] Kumar Joag-Dev and Frank Proschan. Negative association of random variables, with applications. *Ann. Stat.*, 11(1):286–295, 1983. 16, 17

[JST11] Hossein Jowhari, Mert Saglam, and Gábor Tardos. Tight bounds for l_p samplers, finding duplicates in streams, and related problems. In *Proc. 30th ACM Symposium on Principles of Database Systems*, pages 49–58, 2011. 27, 31, 32

[JURdW16] Tiago Januario, Sebastián Urrutia, Celso C. Ribeiro, and Dominique de. Werra. Edge coloring: A natural model for sports scheduling. *European Journal of Operational Research*, 254(1):1–8, 2016. 159

[JW23] Rajesh Jayaram and David P Woodruff. Towards optimal moment estimation in streaming and distributed models. *ACM Trans. Alg.*, 19(3):1–35, 2023. 14

[Kah65] William Kahan. Pracniques: further remarks on reducing truncation errors. *Commun. ACM*, 8(1):40, Jan 1965. 12

[KLS$^+$22] Janardhan Kulkarni, Yang P. Liu, Ashwin Sah, Mehtaab Sawhney, and Jakub Tarnawski. Online edge coloring via tree recurrences and correlation decay. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 104–116. ACM, 2022. 160, 161, 164, 165

[KMNS21] Haim Kaplan, Yishay Mansour, Kobbi Nissim, and Uri Stemmer. 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*, volume 12827 of *Lecture Notes in Computer Science*, pages 94–121. Springer, 2021. 3, 13, 115

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, Cambridge, 1997. 35

[KNP$^+$17] Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P. Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In *Proc. 58th Annual IEEE Symposium on Foundations of Computer Science*, pages 475–486, 2017. 27

[KP20] John Kallaugher and Eric Price. Separations and equivalences between turnstile streaming and linear sketching. In *Proc. 52nd Annual ACM Symposium on the Theory of Computing*, page 1223–1236, 2020. 13

[Kuh20] Fabian Kuhn. Faster deterministic distributed coloring through recursive list coloring. In Shuchi Chawla, editor, *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1244–1259. SIAM, 2020. 116

[LNW14] Yi Li, Huy L. Nguyen, and David P. Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proc. 46th Annual ACM Symposium on the Theory of Computing*, pages 174–183, 2014. 13

[LPS88] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988. 170

[LS11] Luigi Laura and Federico Santaroni. Computing strongly connected components in the streaming model. In Alberto Marchetti-Spaccamela and Michael Segal, editors, *Theory and Practice of Algorithms in (Computer) Systems*, pages 193–205. Springer Berlin Heidelberg, 2011. 166

[Mag24] Roey Magen. Are we still missing an item? *arXiv preprint arXiv:2401.06547*, 2024. 28

[McG14] Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014. 12, 117

[MG92] Jayadev Misra and David Gries. A constructive proof of vizing's theorem. *Information Processing Letters*, 41(3):131–133, 1992. 159, 174

[Mik15] Jesper W. Mikkelsen. Optimal online edge coloring of planar graphs with advice. In *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, volume 9079 of *Lecture Notes in Computer Science*, pages 352–364. Springer, 2015. 164, 166

[Mik16] Jesper W. Mikkelsen. Randomization can be as helpful as a glimpse of the future in online computation. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 39:1–39:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. 160, 164, 166

[MN22a] Roey Magen and Moni Naor. Mirror games against an open book player. In *11th International Conference on Fun with Algorithms (FUN 2022)*, volume 226, pages 20:1–20:12, 2022. 28

[MN22b] Boaz Menuhin and Moni Naor. Keep that card in mind: Card guessing with limited memory. In *Proc. 13th Conference on Innovations in Theoretical Computer Science*, pages 107:1–107:28, 2022. 27, 28

[Mor93] Pieter Moree. Bertrand's postulate for primes in arithmetical progressions. *Computers & Mathematics with Applications*, 26(5):35–43, 1993. 170

[Mor13] Ben Morris. Improved mixing time bounds for the thorp shuffle. *Combinatorics, Probability and Computing*, 22(1):118–132, 2013. 165, 211

[MR14] Michael Molloy and Bruce Reed. Colouring graphs when the number of colours is almost the maximum degree. *Journal of Combinatorial Theory, Series B*, 109:134–195, 2014. 112

[Mut05] S. Muthukrishnan. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, 2005. 12, 27

[MWY13] Marco Molinaro, David Woodruff, and Grigory Yaroslavtsev. Beating the direct sum theorem in communication complexity with implications for sketching. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, page to appear, 2013. 12

[New91] Ilan Newman. Private vs. common random bits in communication complexity. *Inform. Process. Lett.*, 39(2):67–71, 1991. 9, 74

[Nis90] Noam Nisan. Pseudorandom generators for space-bounded computation. In *Proc. 22nd Annual ACM Symposium on the Theory of Computing*, pages 204–212, 1990. 14, 115

[Nis93] Noam Nisan. On read once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993. 15

[NO22] Moni Naor and Noa Oved. Bet-or-pass: Adversarially robust bloom filters. In *Theory of Cryptography Conference*, pages 777–808, 2022. 14

[NSW23] Joseph Naor, Aravind Srinivasan, and David Wajc. Online dependent rounding schemes. *CoRR*, abs/2301.08680, 2023. 160, 161, 164, 165

[NY19] Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. *ACM Trans. Alg.*, 15(3):35:1–35:30, 2019. 14, 28

[PR22] Kenneth G Paterson and Mathilde Raynal. Hyperloglog: Exponentially bad in adversarial settings. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 154–170. IEEE, 2022. 14

[PR23] Binghui Peng and Aviad Rubinstein. Near optimal memory-regret tradeoff for online learning. *arXiv preprint arXiv:2303.01673*, 2023. 14

[RU94] Prabhakar Raghavan and Eli Upfal. Efficient routing in all-optical networks. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 134–143, 1994. 159

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. 43

[Sha49] Claude E. Shannon. A theorem on coloring the lines of a network. *Journal of Mathematics and Physics*, 28(1-4):148–152, 1949. 159

[SS96] Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. 165, 169, 170

[SSS95] Jeanette P Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff–hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995. 33

[SSS23] Menachem Sadigurschi, Moshe Shechner, and Uri Stemmer. Relaxed Models for Adversarial Streaming: The Bounded Interruptions Model and the Advice Model. In *31st Annual European Symposium on Algorithms (ESA 2023)*, volume 274 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 91:1–91:14, 2023. 13

[Ste21] Uri Stemmer. Separating adaptive streaming from oblivious streaming. Lecture at STOC 2021 Workshop: Robust Streaming, Sketching and Sampling, available online at `https://www.youtube.com/watch?v=svgv-xw9DZc&t=7679s`, 2021. Based on joint work with Haim Kaplan, Yishay Mansour, and Kobbi Nissim. 115

[Sti96] Douglas R. Stinson. On the connections between universal hashing, combinatorial designs and error-correcting codes. *Congressus Numerantium*, pages 7–28, 1996. 150

[Sto23] Manuel Stoeckl. Streaming algorithms for the Missing Item Finding problem. In *Proc. 34th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 793–818, 2023. iii

[SW10] Johannes Schneider and Roger Wattenhofer. A new technique for distributed symmetry breaking. In Andréa W. Richa and Rachid Guerraoui, editors, *Proc. 29th ACM Symposium on Principles of Distributed Computing*, pages 257–266. ACM, 2010. 116

[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 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 109:1–109:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 160, 161, 164, 165, 169

[Tar07] Jun Tarui. Finding a duplicate and a missing item in a stream. In *Proc. 4th International Conference on Theory and Applications of Models of Computation*, pages 128–135, 2007. 26

[Viz64] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Discret Analiz*, 3:25–30, 1964. 159

[Viz65] Vadim G Vizing. The chromatic class of a multigraph. *Cybernetics*, 1(3):32–41, 1965. 5

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981. 35, 211

[WZ21] David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science*, pages 1183–1196, 2021. 13

[WZ22] David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. In *Proc. 62nd Annual IEEE Symposium on Foundations of Computer Science*, pages 1183–1196, 2022. 13

[WZZ23] David Woodruff, Fred Zhang, and Samson Zhou. On robust streaming for learning with experts: Algorithms and lower bounds. In *Advances in Neural Information Processing Systems*, volume 36, pages 79518–79539, 2023. 14

[ZA21] Alaettin Zubaroğlu and Volkan Atalay. Data stream clustering: a review. *Artificial Intelligence Review*, 54(2):1201–1236, 2021. 13