When a random tape is not enough: lower bounds for a problem in adversarially robust streaming*

Amit Chakrabarti† Manuel Stoeckl†

October 5, 2023

Abstract

Adversarially robust streaming algorithms are required to process a stream of elements and produce correct outputs, even when each stream element can be chosen depending on earlier algorithm outputs. As with classic streaming algorithms, which must only be correct for the worst-case fixed stream, adversarially robust algorithms with access to randomness can use significantly less space than deterministic algorithms. We prove that for the Missing Item Finding problem in streaming, the space complexity also significantly depends on how adversarially robust algorithms are permitted to use randomness. (In contrast, the space complexity of classic streaming algorithms does not depend as strongly on the way randomness is used.)

For Missing Item Finding on streams of length r with elements in $\{1,...n\}$, and $\leq 1/\text{poly}(n)$ error, we show that when $r = O(2^{\sqrt{\log n}})$, "random seed" adversarially robust algorithms, which only use randomness at initialization, require $r^{\Omega(1)}$ bits of space, while "random tape" adversarially robust algorithms, which may make random decisions at any time, may use $O(\text{polylog}(r))$ random bits. When $r = \Theta(\sqrt{n})$, "random tape" adversarially robust algorithms need $r^{\Omega(1)}$ space, while "random oracle" adversarially robust algorithms, which can read from a long random string for free, may use $O(\text{polylog}(r))$ space. The space lower bound for the "random seed" case follows, by a reduction given in prior work, from a lower bound for pseudo-deterministic streaming algorithms given in this paper.

1 Introduction

Randomized streaming algorithms can achieve exponentially better space bounds than corresponding deterministic ones: this is a basic, well-known, easily proved fact that applies to a host of problems of practical interest. A prominent class of randomized streaming algorithms uses randomness in a very specific way, namely to sketch the input stream by applying a random linear transformation—given by a sketch matrix S—to the input frequency vector. The primary goal of a

*This work was supported in part by the National Science Foundation under award 2006589.
†Department of Computer Science, Dartmouth College.

streaming algorithm is to achieve sublinear space, so it is infeasible to store S explicitly. In some well-known cases, the most natural presentation of the algorithm is to explicitly describe the distribution of S, a classic case in point being frequency moment estimation [Ind06]. This leads to an algorithm that is very space-efficient *provided one doesn't charge the algorithm any space cost for storing S*. Algorithms that work this way can be thought of as accessing a "random oracle": despite their impracticality, they have theoretical value, because the standard ways of proving space *lower* bounds for randomized streaming algorithms in fact work in this model. For the specific frequency-moment algorithms mentioned earlier, [Ind06] goes on to design variants of his algorithms that use only a small (sublinear) number of random bits and apply a pseudorandom generator to suitably mimic the behavior of his random-oracle algorithms. Thus, at least in this case, a random *oracle* isn't necessary to achieve sublinear complexity. This raises a natural question: from a space complexity viewpoint, does it ever help to use a random oracle, as opposed to "ordinary" random bits that must be stored (and thus paid for) if they are to be reused?

For most classic streaming problems, the answer is "No," but for unsatisfactory reasons: Newman's Theorem [New91] allows one to replace a long oracle-provided random string by a much shorter one (that is cheap to store), though the resulting algorithm is non-constructive. This brings us to the recent and ongoing line of work on *adversarially robust* streaming algorithms where we shall find that the answer to our question is a very interesting "Yes." For the basic and natural MissingItemFinding problem, defined below, we shall show that three different approaches to randomization result in distinct space-complexity behaviors. To explain this better, let us review adversarial robustness briefly.

A few recent works have studied streaming algorithms in a setting where the input to the streaming algorithm can be adaptively (and adversarially) chosen based on the past outputs of the algorithm. Existing ("classic") randomized streaming algorithms may fail in this *adversarial setting* when the input-generating adversary learns enough about the past random choices of the algorithm to identify future inputs on which the algorithm will likely fail. There are, heuristically, two ways for algorithm designers to protect against this: (a) prevent the adversary from learning the past random choices of the algorithm (in the extreme, by making a pseudo-deterministic algorithm), or (b) prevent the adversary from exploiting knowledge of past random decisions, by having the algorithm's future behavior depend on randomness that it has not yet revealed. Concretely, algorithms in this setting use techniques such as independent re-sampling [BY20], sketch switching [BJWY20] using independent sub-instances of an underlying classic algorithm, rounding outputs to limit the number of computation paths [BJWY20], and differential privacy in conjunction with sketch switching [HKM$^+$20]. For the most part, these algorithms use at most as many random bits as their space bounds allow. However, some recently published adversarially robust streaming algorithms for vertex-coloring a graph (given by an edge stream) [CGS22, ACGS23], and one for the MissingItemFinding problem [Sto23], assume access to a large amount of oracle randomness: they prevent the adversary from exploiting the random bits it learns by making each output depend on an unrevealed part of the oracle random string. It is still open whether these last two problems have efficient solutions that do not use this oracle randomness hammer. This suggests the following question:

Are there problems for which space-efficient adversarially robust streaming algorithms provably require access to oracle randomness?

In this paper, we prove that for certain parameter regimes, MISSINGITEMFINDING (henceforth, MIF) is such a problem. In the problem MIF(n,r), the input is a stream $\langle e_1,\ldots,e_r\rangle$ of r integers, not necessarily distinct, with each $e_i \in \{1,\ldots,n\}$, where $1 \le r \le n$. The goal is as follows: having received the ith integer, output a number v in $\{1,\ldots,n\} \setminus \{e_1,\ldots,e_i\}$. We call this problem MIF(n,r). To state our results about MIF, we need to introduce some key terminology.

1.1 Models

We briefly describe different four different categories of streaming algorithms, by the way in which they use randomness; more formal definitions are given in Section 3.1.

Deterministic. No randomness is used.

Random seed. The algorithm may be randomly initialized before reading any input, but subsequently makes no new random decisions. This can be seen as all the random bits used by the algorithm counting towards its space cost.

Random tape. The algorithm can make random decisions at any time, but it cannot remember past random decisions without recording them (which would require more space).

Random oracle. Random bits are essentially free to the algorithm; it can read from a long random string which is not counted toward the space cost, and which remains consistent over the life of the algorithm.

These categories have been presented in (almost) increasing order of power. Every deterministic algorithm can be implemented as a random seed algorithm, with no increase in space; every random seed algorithm as a random tape algorithm, with no increase in space; and every random tape algorithm as a random oracle algorithm, possibly with a small additive $O(\log r)$ space increase.

We generally require that algorithms be "tracking" [BJWY20]; i.e., that they present an output after reach each input item and that this *entire sequence* of outputs be correct.[1] Streaming algorithms are also classified by the kind of correctness guarantee they provide: here are three possible meanings of the statement "algorithm $\mathcal{A}$ is δ-error" (we assume that $\mathcal{A}$ handles streams of length r with elements in $\mathcal{I}$ and has outputs in $\mathcal{O}$):

Static setting. For all input streams $\sigma \in \mathcal{I}^r$, running $\mathcal{A}$ against σ will produce incorrect output with probability $\le \delta$.

Adversarial setting. For all adaptive adversaries α (i.e., functions $\alpha\colon \mathcal{O}^\star \to \mathcal{I}$), running $\mathcal{A}$ against α will produce incorrect output with probability $\le \delta$.[2]

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

[2] 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.

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

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 $O(\log r + \log\log|\mathcal{I}| + \log \frac{1}{\varepsilon\delta})$ bits of space.[3]

1.2 Results

The main results of this paper follow. Combining these results with earlier results from [Sto23], we provide Table 1 and Figure 1 summarizing the state of the art for the space complexity of MISSINGITEMFINDING (n, r).

Table 1: These space bounds for MIF(n,r) hold for $r < n/2$ and $\delta = 1/n^2$. $\widetilde{\Theta}(\cdot)$ hides polylog(n,r) factors. As a natural consequence of the setting/randomness type definitions, values increase going down and to the right in this chart. Gray shaded cells are *partially* ours, and partly from results of [Sto23]; see Appendix A.1 for details.

<table>
<tr><th></th><th>Random Oracle</th><th>Random Tape</th><th>Random Seed</th><th>Deterministic</th></tr>
<tr><td>Static setting</td><td>$O(\log n)$</td><td colspan="2">$O((\log n)^2)$</td><td rowspan="3">$\widetilde{\Theta}(\frac{r}{\log n} + r^{1/2})$</td></tr>
<tr><td>Adversarial setting</td><td>$\widetilde{\Theta}(\frac{r^2}{n} + 1)$</td><td>$\widetilde{\Theta}(r^{\Theta(\log_n(r))})$</td><td>$\widetilde{\Theta}(\frac{r^2}{n} + r^{\Theta(1)})$</td></tr>
<tr><td>Pseudo-deterministic</td><td colspan="3">$\Omega(\frac{r}{(\log n)^3} + r^{1/4})$</td></tr>
</table>

Theorem 1.1. *Pseudo-deterministic δ-error random oracle algorithms for* MIF(n,r) *require*

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

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

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

[3] Note: the resulting algorithm is non-constructive/not polynomial time.

Corollary 1.2. *Adversarially robust random seed algorithms for* MIF(n,r) *with error* $\leq \frac{1}{6}$ *require* $\Omega\big(\sqrt{r/(\log n)^3} + r^{1/5}\big)$ *bits of space.*

Theorem 1.3. *There is a family of adversarially robust random tape algorithms, where for* MIF(n,r) *the corresponding algorithm has* $\leq \delta$ *error and uses*

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

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

Theorem 1.4. *Random tape* δ*-error adversarially robust algorithms for* MIF(n,r) *require*

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

bits of space, for $\delta \leq \frac{1}{128n}$.

By a result of [Sto23], when the error level $\delta = 1/\text{poly}(n)$, and $r = \sqrt{n}$, there are adversarially robust random oracle algorithms using $O((\log r)^2)$ bits of space. In contrast, Theorem 1.4 implies that when $r = \sqrt{n}$, adversarially robust random tape (and hence also random seed) algorithms require $\Omega(r^{1/4})$ bits of space, which is exponentially larger. There is also an exponential separation between random seed and random tape algorithms in the adversarial setting when $r = 2^{\sqrt{\log n}}$ and $\delta = 1/\text{poly}(n)$. Here Corollary 1.2 implies random seed algorithms need $\Omega(\sqrt{r/(\log r)^6})$ bits of space, while by Theorem 1.3 there is a random tape algorithm using $O((\log r)^3)$ space.

These separations rule out the possibility of a way to convert an adversarially robust random oracle algorithm to use a random tape or random seed, with only minor (e.g., $\times\,\text{polylog}(r,n)$) overhead. The fact that random oracle algorithms may be much more efficient than random tape algorithms in the adversarial setting implies that MISSINGITEMFINDING is a problem for which much lower space usage is possible if one has a computationally bounded adversary that one can fool with a pseudo-random generator, than if one has a computationally unbounded adversary.

1.3 Related work

We briefly survey related work. An influential early work [HW13] considered adaptive adversaries for *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

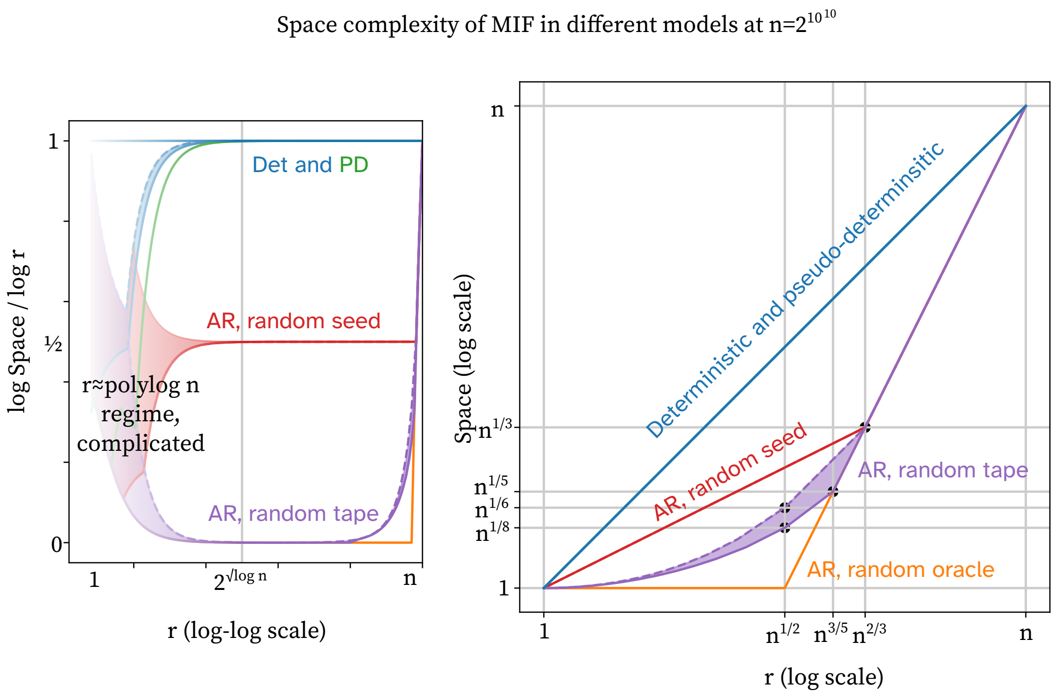

Figure 1: Plots of the upper and lower bounds for the space complexity of MIF(n,r) streaming algorithms in different models, at error level $\delta = 1/n^2$. Warning: scales are unusual, to show the asymptotic behavior in different regimes. AR = Adversarially Robust; PD = pseudo-deterministic. Upper and lower bounds are drawn using lines of the same color; the region in between them is shaded. In the plot on the right, the upper and lower bounds shown all match except for the case of adversarially robust, random tape algorithms.

oracle model, although later work ([WZ22]) removed this assumption. [BY20] observed that in sampling-based streaming algorithms, increasing the sample count is often all that is needed to make an algorithm adversarially robust. [HKM$^+$20] describe how to use differential privacy techniques as a more efficient alternative to sketch-switching, and [BEEO22] use this as part of a more efficient adversarially robust algorithm for turnstile F_2-estimation.

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 (of approximating a certain real-valued function) that requires exponentially more space in the adversarial setting than in the static setting. [CGS22] (in a brief comment) observe a similar separation for a simple problem along the lines of MIF. They also find and prove lower bounds for adversarially robust coloring algorithms for graph edge insertion streams. [Sto23] consider the MISSINGITEMFINDING problem and, among upper and lower bounds in a number of models, describe an adversarially robust algorithm for MISSINGITEMFINDING that requires a random oracle, and ask whether a random oracle is *necessary* for space-efficient algorithms.

In recent years, a few other settings for streaming have been described. The white-box adversarial setting [ABJ+22] is similar to the adversarial setting, although the adversary has the additional power of seeing the internal state of the algorithm, plus (if used) the random oracle used by the algorithm. [Sto23] proved an $\Omega(r/\text{polylog}(n))$ lower bound for $\textsc{mif}(n,r)$ for random tape algorithms in this setting, suggesting that any more efficient algorithms for MISSINGITEMFINDING must conceal some part of their internal state. Pseudo-deterministic streaming algorithms were introduced by [GGMW20], who found lower bounds for a few problems. [BKKS23, GGS23] find 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[4] uses only $O(\log\log m)$ space.

While it is not posed as a streaming task, the Mirror Game introduced by [GS18] is another problem with conjectured separation between the space needed for different types of randomness. In the Mirror Game, two players (Alice and Bob) alternately state numbers in the set $\{1,\dots,n\}$, where n is even, without repeating any number, until one player mistakenly states a number said before (loss) or the set is completed (tie). [GS18] show that if Alice has $o(n)$ bits of memory and plays a deterministic strategy, Bob can always win. [Fei19, MN22b] show that if Alice has access to a random oracle, she can tie-or-win w.h.p. using only $O(\text{polylog}(n))$ space. A major open question here is how much space Alice needs when she does not have a random oracle. [MN22a] do not resolve this, but find that if Alice is "open-book" (equivalently, that Bob is a white-box adversary and can see her state), then Alice needs $\Omega(n)$ bits of state to tie-or-win.

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 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.

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. [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.

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.[5]

For a more detailed history and survey of problems related to MISSINGITEMFINDING, we

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

[5]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.

direct the reader to [Sto23].

2 Technical overview

The three main results of this paper, Theorem 1.1, Theorem 1.3, and Theorem 1.4 are all significant generalizations of existing proofs from [Sto23] which handled different (and more tractable) models. To explain how each of them works, we will first describe how the basic proofs worked, and then what must be changed for the full version.

The AVOID problem. At the core of many of the MISSINGITEMFINDING lower bounds is the AVOID(t,a,b) communication problem, introduced in [CGS22]. Here we have two players, Alice and Bob. In the problem, Alice has a set $A \subseteq [t]$ of size a, and should send a message (as short as possible) to Bob, who should use the message to output a set $B \subseteq [t]$ of size b which is disjoint from A. [CGS22] showed that both deterministic and constant-error randomized protocols for this problem require $\Omega(ab/t)$ bits of communication. Some of the lower bounds for MIF operate by using a z-space algorithm for MIF as a subroutine to implement a z-bit protocol for an AVOID(t,a,b) task, thereby proving $z = \Omega(ab/t)$.

2.1 Random tape upper bound (Theorem 1.3 / Section 5)

The random oracle algorithm. The best (up to polylog factors) adversarially robust random oracle algorithm for MIF(n,r) has the following structure. Using oracle randomness, we sample a random sequence L of length $r+1$ with elements in $[n]$, chosen without replacement. As we process the stream, we keep track of which elements in L were in the input stream so far (were "covered"). When asked for an output, we report the element at the lowest index in L which is uncovered. Because L is chosen using oracle randomness, the space cost of this algorithm is just the cost of keeping track of covered positions. [Sto23] show that this can be done efficiently, in $\widetilde{O}(1+r^2/n)$ space. Because the list L is chosen uniformly at random, the adversary has no significant advantage in predicting most of the uncovered elements of the list, even if it knew the covered elements. There is one exception: the adversary always knows what the uncovered element with the lowest index is, since this is the output. As a result, the adversary at each point must choose between either echoing back the last output, or picking an essentially random element from $[n]$, in the hopes of hitting L. Due to this, with high probability, the set of covered positions can be expressed as the union of a contiguous interval starting at 1 (from the echo strategy) and a sparse random set of size $O(r^2/n)$ (from the random-pick strategy). Together, these can be encoded using $\widetilde{O}(1+r^2/n)$ space.

Recursion and random tape. The random oracle algorithm, implemented in the random tape setting, would have cost $\widetilde{O}(r)$, dominated by the cost of storing the random list L. But why does L need to have length $r+1$? This is needed for the algorithm to be resilient to the echo strategy, which could potentially make the algorithm reveal (and adversary cover) one element of L per turn.

The random tape algorithm presented in this paper mitigates the echo strategy by requiring many steps to cover the lowest uncovered element in L. To do this, we use recursion. Say that we have $n \geq r^2$, and then set $w_1 = O(r), w_2 = n/w_1$, $b_1 = b_2 = O(\sqrt{r})$. Instead of choosing a list L of elements r in $[n]$, we partition $[n]$ into w_1 sets $D_1, \ldots, D_{w_1}$ of size w_2. Then let L_1 be a random list of b_1 elements in the domain $[w_1]$, where each integer j in L_1 is associated with the set D_j. Within the set D_k, where k is the lowest index uncovered element in L_1, we recursively run a copy of our original algorithm, with a list L_2 of b_2 elements in D_k. Thus when the algorithm processes an integer $e \in [n]$, it now checks if e is contained by one of the uncovered sets D_i where $i \in L_1$, and if D_i is not equal to D_k, it marks it as covered. If e is in the set D_k, then e is passed to the recursive copy of the algorithm; and only if the list L_2 is entirely covered do we mark the top level set D_k covered, and start a new copy of the algorithm within the next uncovered set indicated by L_1.[6]

For this recursive algorithm to work, the values of b_1, b_2 and w_1, w_2 must be chosen so that neither the echo nor the recursive strategies for the adversary will make too much progress. For example, if we had set $b_1 = 2$ and $w_1 = 4$, then the adversary could cover all but the lowest-index element in L_1 with just 4 inputs, making the top level ineffective; but if $b_1 = 2$ and $w_1 = 100r$, the adversary would be unlikely to guess any element of L_1 at all. Some calculation gives the minimum reasonable size of w_1 as $\geq b_1 b_2$. On the other hand, to defend against the echo strategy, we need $b_1 b_2 = \Omega(r)$, and we have the natural constraints $w_2 \geq b_2$ and that $w_1 w_2 \leq n$. If $n \geq r^2$, the values for w_1, w_2, b_1, b_2 given above will satisfy these constraints, giving us an algorithm which stores two lists (and coverage information) of size $O(\sqrt{r})$ each.

For the full algorithm, we use ℓ levels of recursion, where ℓ is $\Theta(\min(\log r, \log_n r))$, with different domain splitting factors and list lengths. Optimizing parameters with all the constraints leads to the rather complicated space upper bound of Theorem 1.3.

2.2 Random tape lower bound (Theorem 1.4 / Section 4)

The random seed lower bound. The adversary used by the random seed, adversarially robust lower bound of [Sto23] proceeds in a number of steps, each of length t; in each step, it either can (A) learn some new information about the initial state of the algorithm (the "random seed"), by sending the algorithm a short sequence of inputs in $[n]^t$, looking at the resulting output, and ruling out the seed values that could not have produced the output; or (B) it cannot learn much information, because for any possible input sequence in $[n]^t$, the algorithm will produce a unique output with constant probability. Because there are a finite number of initial seed values, and because each time the adversary follows the (A) case a constant fraction are ruled out, either at some point the adversary will exactly learn the seed, at which point we have reached the (B) case; or the algorithm will not reveal much information about the seed in a given step, in which case we are also in the (B) case. Because case (B) means that the algorithm behaves pseudo-deterministically, the algorithm must use enough space to pseudo-deterministically solve MIF(n,t).

[6] Setting up a new copy of the algorithm within the next set requires picking a new random list for L_2; this is where the algorithm needs to have random tape access.

A reversed view of AVOID. While the communication lower bound for AVOID has been used by [Sto23] to find a deterministic lower bound for $\text{MIF}(n,r)$, and is used for our pseudo-deterministic lower bound, the easiest way to apply it requires that given a state of the algorithm, one can identify a large set of likely outputs which has high overlap with a certain known set. We have not found a way to do this. Instead, we find a lemma (Lemma 4.2) which states that, if we send a random input vector x with elements in $[n]$ and length t to the algorithm, it is possible to use the probability distribution of algorithm states resulting from each possible vector x to associate a set H_σ to every algorithm state σ, where H_σ contains precisely the values in $[n]$ which are unlikely to have been in x. If the algorithm, from a state σ, is likely to produce a value $i \in [n]$, then (by virtue of avoiding past inputs) i will probably be in H_σ. The lemma states that if the algorithm does not use many states, then H_σ will probably be small. Consequently, if the adversary sends a random input to the algorithm, it will probably reach a state from which most of the possible outputs are contained inside a small set. *Which state*, the adversary does not know.

Recursion. If the adversary knows that w.h.p. the next τ outputs of the algorithm will all be in a given set W, then it can run a random tape adversary for $\text{MIF}(|W|,\tau)$. Then, for the algorithm to have low failure probability, it must use at least whatever the space lower bound for $\text{MIF}(|W|,\tau)$ is. For our random tape lower bound, we design an adversary that repeatedly identifies a sequence of smaller and smaller sets $W^{(1)}, W^{(2)}$, and so on, in which the next few inputs are likely to be contained; eventually the adversary reaches a point where $W^{(\ell)}$ is small enough, and the number τ_ℓ of "next few inputs" large enough, that one can apply the lower bound for AVOID to prove the algorithm needs $\Omega(\tau_\ell^2/|W^{(\ell)}|)$ bits of space to be correct.

Searching with sub-adversaries. We are left with the question of how to identify a set W containing the next few outputs. Here, we adapt the approach of the random seed lower bound. We know (conditioned on the transcript of inputs and outputs so far) the distribution of possible algorithm states, and how that will change after we send a random vector v to the algorithm. Using our lemma, we can associate each state σ that could be reached upon sending v with a set of outputs H_σ defined so that, if σ is the actual state, the algorithm is unlikely to produce in the future any outputs outside of H_σ. Over a number of steps, we will try to learn what the actual state ρ after sending v was—more precisely, what approximately H_ρ was. In each step, we determine if, conditioned on the transcript, there exists any adversary which over the next few inputs is likely to have the algorithm produce an output that rules out a constant fraction of the possible states. (Output $y \in [n]^t$ rules out state σ if for any $i \in [t]$, $y_i \notin H_\sigma$.) Since there are a limited number of states, this can only happen a limited number of times, and as a result we will likely reach a point where there is *no* adversary that is likely to rule out many of the possible states. At this point, the algorithm output over the next few steps will likely be contained in a certain set, which is the W that the adversary was looking for.

2.3 Pseudo-deterministic lower bound (Theorem 1.1 / Section 6)

Lower bound for deterministic algorithms. The deterministic algorithm lower bound for $\textsc{mif}(n,r)$ from [Sto23] can be viewed as proving that, if the number of bits of state z of the algorithm is small, then the set of all possible outputs of the algorithm must be large. The method by which this is done is a recursive one. We split the time range $\{1,\dots,r\}$ into a number (ℓ, roughly $\log n$) of disjoint intervals of lengths $t_\ell, t_{\ell-1}, \dots, t_1$, with $\sum_{i=1}^{\ell} t_i = r$, with the lowest numbered interval going last. For each stream $x \in [n]^\star$ of length $r - t_1$, we define the set F_x of all possible outputs of the algorithm for streams of length r that start with x. The same definition works if x has length $r - t_1 - t_2$, or $r - t_1 - t_2 - t_3$, and so on. A key property of the algorithm is that as x gets shorter, the size of F_x must increase rapidly. For example, that the smallest value of $|F_x|$ for $x \in [n]^{r-t_2-t_1}$ is at least twice the smallest value of $|F_y|$ for $y \in [n]^{r-t_1}$, and so on. For some $x \in [n]^{r-t_2-t_1}$, this can be proven by using the algorithm to implement a z-bit protocol for $\textsc{avoid}(|F_x|, t_2, \min|F_y|)$, in which Alice sends a message that encodes a set F_z for $w \in [n]^{r-t_1}$. This is done by sending a state σ of the algorithm corresponding to w; Bob can then, using just σ and not knowing w at all, recover the set F_z by evaluating the algorithm for each possible suffix of the stream in $[n]^{t_1}$, starting from state σ each time, and reporting the union of all outputs.

Relaxing "all outputs" to "common outputs". This design does not work *as is* for pseudo-deterministic, random oracle algorithms. Here we can view the algorithm as randomly choosing one of a large number of deterministic algorithms, each of which makes errors on a different set of inputs. The (random) state σ associated with a given partial stream $z \in [n]^{r-t_1}$, when used to compute output values for $\textsc{mif}$, may give incorrect outputs with average probability δ. Trying to estimate the set F_z as in the deterministic case will fail, because the set of all outputs when we run the algorithm starting at σ for an additional t_1 steps could contain a δ fraction of garbage values. Instead, our approach is to define a collection of sets T_x of "common" outputs, for each partial stream $x \in [n]^{r-t_1} \cup [n]^{r-t_1-t_2} \dots$. Each T_x will be a subset of the set of possible canonical outputs for streams which are continuations of x. The sets T_x are defined recursively; if $x \in [n]^{r-t_1-t_2}$, then T_x depends on sets T_y for $y \in [n]^{r-t_1}$. The exact definition is fairly complicated, and we defer its explanation to the actual proof, due to two issues:

- A set T_x must be robustly computable from a state σ corresponding to x; even if σ produces incorrect values a δ fraction of the time, T_x should be unlikely to include a value that is not a canonical output for a stream with prefix x. One method we use to ensure this is the well known random threshold trick in which, instead of picking all outputs which occur more than α times in a sample, we choose a random threshold γ in e.g $[\alpha, 2\alpha)$ and pick outputs that occur more than γ times.

- The sets T_x must grow rapidly as x gets shorter. Selecting the "most likely" outputs for a random continuation of x will not work if the continuation is chosen uniformly at random, because some outputs might be incredibly common under this distribution, and others might occur with probability $\ll \delta$ and be impossible to robustly detect. Our definition leverages the AVOID lower bound to ensure that we find many different outputs which have probability $\Omega(\delta)$ under at least one of a specific sequence of distributions.

Error amplification and the case $n \gg r$. 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 $2^{O(z)}$ possible outputs; so if $n \gg r$, we can often obtain a stronger lower bound by assuming that n is actually $2^{O(z)}$, and then solving the system of inequalities to get a lower bound on z.

3 Preliminaries

Notation. Throughout this paper, $\log x = \log_2 x$, while $\ln x = \log_e x$. Further, $[k] := \{1,2,\ldots,k\}$. We use $\widetilde{\Omega}(x)$, $\widetilde{O}(x)$, and $\widetilde{\Theta}(x)$ when discussing space usage of MISSINGITEMFINDING (n,r) to hide $\text{polylog}(n,r)$ factors in x. $\mathbb{1}_E$ is the indicator random variable for the event E; it is 1 iff E occurs, and 0 otherwise. $[a,b)$ is a half open interval of real numbers. For vectors a,b in X^s and X^t, respectively, $a.b$ is their concatenation, in X^{s+t}. The function $\text{SORT} : 2^{\mathbb{Z}} \to \mathbb{Z}^\star$ takes as input a subset S of $\mathbb{Z}$ and outputs a vector of length $|S|$ containing the elements of S sorted in increasing order. $\binom{S}{k}$ is the set of k element subsets of S, while $\text{SORT}\binom{S}{k} = \{\text{SORT}(Y) : Y \in \binom{S}{k}\}$. We sometimes treat vectors as sets; e.g., for $y \in [n]^t$, write $y \subseteq S$ to mean that $\forall i \in [t] : y_i \in S$. For a given finite set X, define $\triangle[X]$ to be the set of probability distributions over X.

3.1 Types of randomness

In this paper, we view streaming algorithms as generalizations of finite state machines. An algorithm $\mathcal{A}$ has a finite set of states Σ, and finite input set $\mathcal{I}$, and finite output set $\mathcal{O}$. The algorithm also has a 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. The four cases described in Section 1.1 are more formally given as follows.

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 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 from a distribution $\mathcal{D}$ over Σ.[7] $\mathcal{R}$ 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 in Σ has a unique output value in $\mathcal{O}$ associated to it.[8]

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 family. The output of the algorithm is a function of the current state σ and R.

A brief 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, every z-bit random tape algorithm only has a corresponding $\leq (z + \log(r))$-bit random oracle algorithm, as 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 variant, is to assume the random oracle algorithm has access to a clock or knows the stream length for free; both are reasonable assumptions in practice.

3.2 Useful lemmas

These will be used in following sections. When no external work is cited, a proof is given for completeness either here or in Appendix A.2.

Lemma 3.1 (Multiplicative Azuma's inequality)**.** *Let $X_1,\dots,X_t$ be $[0,1]$ random variables, and $\alpha \geq 0$. If, for all $i \in [t]$, $\mathbb{E}[X_i \mid X_1,\dots,X_{i-1}] \leq p_i$, then*

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

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

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

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

[7] Requiring that this model use a fixed initial state could make some algorithms use one additional "INIT" state.

[8] 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.

Lemma 3.2 (Chernoff bound with negative association, from [JP83]). *The standard multiplicative Chernoff bounds work with negatively associated random variables.*

Lemma 3.3 (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.)

Lemma 3.4 (AVOID communication lower bound, from [CGS22]). *In the $\text{AVOID}(t, a, b)$ one-way communication problem, player Alice has a set $A \in \binom{[t]}{a}$, and should send a short binary message to player Bob, who should use the message to output a set $B \in \binom{[t]}{b}$ which is disjoint from A. Randomized protocols with $\leq \delta$ error on any input, or which have $\leq \delta$ error when Alice's input is chosen uniformly at random from $\binom{[t]}{a}$, require communication which is*

$$\geq \frac{ab}{t \ln 2} + \log(1 - \delta) .$$

Lemma 3.5. *Random tape and random seed streaming algorithms for $\text{MIF}(n, r)$ require $\geq \log(r + 1)$ bits of space.*

Proof of Lemma 3.5. Each state of a random tape or random seed streaming algorithm has a unique associated output value. If $z < \log(r + 1)$, then the streaming algorithm has at most r 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. $\square$

4 Random tape lower bound

In this section, we prove a space lower bound for adversarially robust random seed algorithms for $\text{MIF}(n, r)$ with error $\delta \leq \frac{1}{128n}$. For a given random tape algorithm $\mathcal{A}$ using z bits of state, we do this by describing an adversary for $\mathcal{A}$ so that, if the algorithm $\mathcal{A}$ has error $\leq \delta$, then z must be at least some lower bound. In what follows, let Σ be the set of states of $\mathcal{A}$.

The adversary is described by Listing 1. One of its key parameters is an integer ℓ, indicating the number of "stages" that the adversary will perform. As determining the value of ℓ that produces the best lower bound for a given algorithm with parameters (n, r) results in a complicated expression which is roughly (but not quite) $2\frac{\log n}{\log r}$, we will instead, for each ℓ, prove a slightly different lower bound; the final lower bound on z is the maximum of these. For example, with $\ell = 1$ we reproduce the random oracle lower bound of $z = \Omega(r^2/n)$, while with $\ell = 2$ we obtain $z = \Omega((r^3/n)^{1/4})$, and cases $\ell \geq 3$ are more complicated.

The "stages" of the ℓ-stage adversary proceed in order and are nested. Initially the adversary is in the 1st stage; after sending and receiving specific sequences of elements it may enter the 2*nd* stage, while remaining in the 1st; a stage is only "left" when the adversary ends. See for example Figure 2.

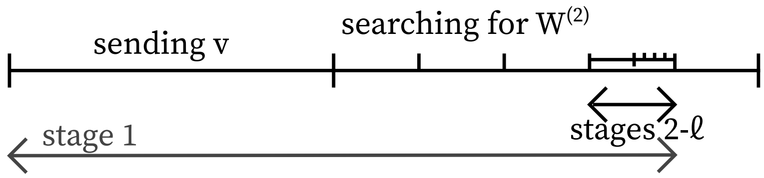

Figure 2: A diagram showing roughly the structure of the first stage of the adversary, and how it relates to following stages.

In the kth stage, roughly speaking, the adversary "knows" that the next set of τ_k outputs of the algorithm will probably be contained in a set $W^{(k)} \subseteq [n]$, no matter what it does; the goal of the adversary is to identify a smaller set $W^{(k+1)}$ within which the next τ_{k+1} outputs will probably be contained. There are two key tricks to this:

1. Because this is a random tape algorithm, there are a finite number of states; if the adversary sends a random vector in $W^{(k)}$, one can show (as a consequence of the upcoming Lemma 4.2) that the algorithm will probably enter a state ρ in which all future outputs are probably contained in a set H_ρ which is significantly smaller than $W^{(k)}$.

2. While the adversary does not know what ρ is, it can either approximately identify H_ρ or some similar set, using an $O(\ell z)$-step search procedure. In each step, either – conditioned on the transcript of all inputs and outputs – there exists a "sub-adversary" (read: way for the adversary to behave over the next few inputs) which may rule out a constant fraction of the possible states, or one can compute a set $W^{(k+1)}$ similar to H_ρ.

In order to explain Listing 1, we will need the following definitions:

Definition 4.1. 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 $|\{\sigma \in Q : y \subseteq H_\sigma\}| \leq \frac{1}{2}|Q|$.

Say Υ is a t-length deterministic adversary. (That is, a function which maps vectors in $[n]^\star$ of length $\leq t-1$ (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 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 α-SPLITTING for Q against a distribution $\mathcal{D} \in \triangle[\Sigma]$ if, when we choose a random state S from $\mathcal{D}$,

$$\Pr[G(S, \Upsilon) \text{ is divisive for } Q] \geq \alpha$$

[9]For example: say $e_1, \ldots, e_t$ is the set of inputs up to the start of the kth stage, and $o_1, \ldots, o_t$ the set of outputs. Let A be an instance of the algorithm. Then for any vector $v \in \text{SORT}\binom{W^{(k)}}{\lceil \tau_k/2 \rceil}$, $P(v)(\sigma) = \Pr[\text{the state of } A \text{ just after receiving } v \text{ is } \sigma | A \text{ produces outputs } o_1, \ldots, o_t \text{ when given input } e_1, \ldots, e_t]$

Listing 1 An ℓ-stage adversary for a random tape algorithm.

Parameters: $h_{\max}, \tau_1, \ldots, \tau_\ell, w_1, \ldots, w_\ell$ from Eqs. (2) to (5)

ADVERSARY(STAGE k, ON $W^{(k)}$)

$v \leftarrow$ a uniformly random vector in SORT$\binom{W^{(k)}}{\lceil \tau_k/2 \rceil}$.
send v to the algorithm
if $k = \ell$ **then**
  receive e_0, the current algorithm output
  for i in $1, \ldots, \lfloor \tau_\ell/2 \rfloor$ **do**
    send e_{i-1} to the algorithm
    receive e_i as current algorithm output
else
  Let P : SORT$\binom{W^{(k)}}{\lceil \tau_k/2 \rceil} \rightarrow \triangle[\Sigma]$ map possible values of v to the resulting distribution over states in Σ, conditioned on the transcript of inputs/outputs before v was sent[9]
  Compute H_σ for each state σ per Lemma 4.2, using P and with $\binom{W}{q} = \binom{W^{(k)}}{\lceil \tau_k/2 \rceil}$.
  Let $Q_0 = \{\sigma \in \Sigma : |H_\sigma| \leq \frac{1}{2} w_{k+1}\}$
  for h in $1, \ldots, h_{\max}$ **do**
    Let $\mathcal{D}$ be the distribution over alg. states conditioned on the transcript so far
    if $\exists$ a $1/(8\ell)$-splitting τ_{k+1}-length det. adversary Υ for Q_{h-1} given $\mathcal{D}$ **then**
      run Υ against the algorithm, and let $y \in [n]^{\tau_{k+1}}$ be the output
      $Q_h \leftarrow \{\sigma \in Q_{h-1} : y \subseteq H_\sigma\}$ ▷ *Have a $\geq 1/(8\ell)$ chance that $|Q_h| \leq \frac{1}{2}|Q_{h-1}|$*
      if $Q_h = \emptyset$ **then abort**
    else
      $W^{(k+1)} \leftarrow \{i \in W^{(k)} : |\{\sigma \in Q_{h-1} : i \in H_\sigma\}| \geq \frac{1}{2}|Q_{h-1}|\}$.
      run ADVERSARY($k+1$, $W^{(k+1)}$)
      return
  abort

For all ℓ, the ℓ stage adversary is only used when the following condition holds;[10] as a result, the ℓ-stage lower bound will be the minimum of the lower bound when the adversary exists, and of the right hand side of the following inequality:

$$z \leq \frac{1}{64\ell} r^{1/\ell} \tag{1}$$

Parameters for the adversary are set as follows.

$$h_{\max} = 64\ell z \tag{2}$$

$$\forall k \in [\ell], \qquad \tau_k = \left\lfloor \frac{r}{h_{\max}^{k-1}} \right\rfloor \tag{3}$$

$$w_1 = n \tag{4}$$

$$\forall k \in \{2,\dots,\ell\}, \qquad w_k = 2\left\lfloor \frac{z+1+\log(8\ell)}{\lceil \tau_{k-1}/2 \rceil} w_{k-1} \frac{2\ln 2}{1-\ln 2} \right\rfloor \tag{5}$$

There are a few consequences of these definitions. By Eq. 1, we have $h_{\max}^{\ell} \leq r$, which implies that $\tau_1 \geq \ldots \geq \tau_\ell \geq h_{\max} \geq 1$. Eq. 1 implies that $\ell \leq \frac{1}{6}\log r$, while Lemma 3.5 implies that $\log(r+1) \leq z$, so we have

$$z \geq 6\ell \geq 2\log(8\ell) \geq 2\ln(8\ell) \tag{6}$$

For $k \in \{2,\dots,\ell\}$:

$$w_k \leq 2\frac{z+1+\log 8\ell}{\lceil \tau_{k-1}/2 \rceil} w_{k-1} \frac{2\ln 2}{1-\ln 2} \leq \frac{4\ln 2}{1-\ln 2} \frac{z+1+\frac{1}{2}z}{\lceil \tau_{k-1}/2 \rceil} w_{k-1} \leq 60\frac{z}{\tau_{k-1}} w_{k-1}$$

As $\tau_{k-1} \geq h_{\max} \geq 64z$, we have $w_k < w_{k-1} < \ldots < w_1$, and furthermore that:

$$w_\ell \leq 60\frac{z}{\tau_{\ell-1}} w_{\ell-1} \leq n\frac{(60z)^{\ell-1}}{\tau_{\ell-1}\tau_{\ell-2}\ldots\tau_1}$$

We now present the following key lemma; typically when applied in the proof for the kth stage, $q = \Theta(\tau_k)$, $W = W^{(k)}$, $\hat{w} = \frac{1}{2}w_{k+1} + \Theta(1)$, and $\alpha = \Theta(1/\ell)$.

Lemma 4.2. *Let W and Σ be sets, q an integer, and P a function from $\binom{W}{q}$ to $\triangle[\Sigma]$, where $|\Sigma| \leq 2^z$. Let F be a random function $\binom{W}{q} \to \Sigma$ in which for all $x \in \binom{W}{q}$ and $\sigma \in \Sigma$, $\Pr[F(x) = \sigma] = P(x)(\sigma)$, and let X be a uniformly random element of $\binom{W}{q}$, chosen independently of F. For each $\sigma \in \Sigma$, define*

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

[10] This condition is chosen to ensure that $\tau_\ell \geq 1$. It is actually a bit tighter than strictly necessary – we could have used $z \leq \frac{1}{64\ell} r^{1/(\ell-1)}$ with some work – but as will be shown later, this does not worsen the final lower bound.

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

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

Proof of Lemma 4.2. Consider a specific $\sigma \in \Sigma$. By linearity of expectation:

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

Then by Markov's inequality,

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

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

Since X is drawn uniformly at random from $\binom{W}{q}$, the random variables $\{\mathbb{1}_{i \in X}\}_{i \in W}$ are negatively associated, with $\mathbb{E}\mathbb{1}_{i \in X} = \frac{q}{|W|}$ for each $i \in W$. For any set $A \subseteq W$, we use the multiplicative Chernoff bound (Lemma 3.2) to bound the probability that X's overlap with A is much smaller than the expected value:

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

We now bound:

$$\begin{aligned}
\Pr[F(X) = \sigma] &\leq \frac{\Pr\left[\sum_{i \in H_\sigma} \mathbb{1}_{i \in X} \leq \frac{q}{2|W|} |H_\sigma|\right]}{\Pr\left[\sum_{i \in H_\sigma} \mathbb{1}_{i \in X} \leq \frac{q}{2|W|} |H_\sigma| \,\middle|\, F(X) = \sigma\right]} \\
&\leq 2\exp\left(-\frac{1}{2}(1 - \ln(2)) \frac{q}{|W|} |H_\sigma|\right).
\end{aligned}$$

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

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

□

The core of the proof is in the following lemma:

Lemma 4.3. *For each $k \in [\ell]$, let E_k be the event that the adversary finds a set $W^{(k)}$, and that during the kth stage, the algorithm does not produce an output outside $W^{(k)}$.*

Since $W^{(1)} = [n]$, $\Pr[E_1] = 1$. This lemma claims that for all $k = 1, \ldots, \ell - 1$,

$$\Pr[E_{k+1}] \geq \Pr[E_k] - \frac{1}{2\ell}$$

and thus that $\Pr[E_\ell] \geq 1 - \frac{\ell-1}{2\ell}$.

Its proof uses the following short result:

Lemma 4.4. *For each $k \in [\ell]$, the set $W^{(k)}$ (if it is found) has size $\leq w_k$.*

Proof of Lemma 4.4. As $W^{(1)} = [n]$ and $w_1 = n$, the case $k = 1$ holds.

For each $k < \ell$, note that by Line 11 of Listing 1, the set Q_0 only contains states $\sigma \in \Sigma$ with $|H_\sigma| \leq \frac{1}{2} w_{k+1}$. When $W^{(k+1)}$ is chosen on Line 19, the value of Q_{h-1} is a subset of Q_0. By the definition of $W^{(k+1)}$, we have:

$$\begin{aligned} |W^{(k+1)}| &= \left| \left\{ i \in W^{(k)} : \frac{|\{\sigma \in Q_{h-1} : i \in H_\sigma\}|}{|Q_{h-1}|} \geq \frac{1}{2} \right\} \right| \\ &\leq \sum_{i \in W^{(k)}} 2 \frac{|\{\sigma \in Q_{h-1} : i \in H_\sigma\}|}{|Q_{h-1}|} \\ &= 2 \frac{\sum_{\sigma \in Q_{h-1}} |H_\sigma|}{|Q_{h-1}|} \leq 2 \frac{1}{2} w_k = w_k \,. \end{aligned}$$

□

Proof of Lemma 4.3. When the adversary (Listing 1) is run against a random tape algorithm $\mathcal{A}$, let ρ be the state of $\mathcal{A}$ after v is sent (Line 2).

We define the following three classes of bad events within stage k:

- B_{REPEAT} The algorithm produces an output in $W^{(k)} \setminus H_\rho$.
- B_{BIG} The state ρ has $|H_\rho| > \frac{1}{2} w_{k+1}$.
- B_{ABORT} The adversary aborts at some point instead of moving to the $(k+1)$st stage.

We now bound the probability of each.

B_{REPEAT} Say that α is the probability that during the kth stage, the algorithm produces an output in $W^{(k)} \setminus H_\rho$. For any $i \in W^{(k)} \setminus H_\rho$, there is a $\geq \frac{\lceil \tau_k/2 \rceil}{4|W^{(k)}|} \geq \frac{\tau_k}{8w_k}$ probability that the set v from Line 1 contained ρ, conditioned on the algorithm reaching state ρ. Consequently, if it has reached the kth stage, the probability that the algorithm fails because it produces an output which overlaps with the past input v is $\geq \frac{\alpha\tau_k}{8w_k}$. By induction, there is a $\geq 1/2$ chance of reaching the kth stage, so we must have $\frac{\alpha\tau_k}{16w_k} \leq \delta$, which implies $\alpha \leq \frac{16w_k}{\tau_k}\delta$. Since $\tau_\ell \geq 1$, and $k \leq \ell - 1$, $\tau_k \geq h_{\max} \geq 64\ell z \geq \ell$. Also, since $w_1 > w_2 > \ldots > w_k$, we have $w_k \leq n$. Thus $\frac{16w_k}{\tau_k}\delta \leq \frac{16n}{\ell}\delta \leq \frac{1}{8\ell}$.

B_{BIG} We apply Lemma 4.2 to P, with

$$\hat{w} = \left\lceil \frac{z+1+\log(8\ell)}{\lceil \tau_k/2 \rceil} |W^{(k)}| \frac{2\ln 2}{1-\ln 2} \right\rceil \leq 1 + \left\lfloor \frac{z+1+\log(8\ell)}{\lceil \tau_k/2 \rceil} w_k \frac{2\ln 2}{1-\ln 2} \right\rfloor = \frac{1}{2} w_{k+1} + 1$$

where $|W^{(k)}| \leq w_k$ by Lemma 4.4. As $\frac{1}{2} w_{k+1}$ is an integer,

$$\Pr\left[|H_\sigma| > \frac{1}{2} w_{k+1}\right] = \Pr\left[|H_\sigma| \geq \frac{1}{2} w_{k+1} + 1\right] \leq \Pr\left[|H_\sigma| \geq \hat{w}\right] \leq \frac{1}{8\ell}$$

B_{ABORT} There are two ways in which the algorithm can abort: if (Line 22) either $h_{\max}$ loop iterations are performed without failing to find a splitting adversary, or if (Line 17) at some point $Q_h = \emptyset$. If event E_k holds, and B_{REPEAT} does not, then all outputs produced by the algorithm will be contained in $W^{(k)}$ and not contained in $W^{(k)} \setminus H_\rho$; thus 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 16. Thus $\rho \in Q_h$ and the abort on Line 17 will not be used.

We now bound the probability that the algorithm will abort using Line 22. 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 $|Q_h| \leq |Q_{h_1}|$. Thus with $z+1$ divisive outputs, $|Q_{h_{\max}}| \leq |Q_0|/2^{z+1} \leq |\Sigma|/2^{z+1} \leq \frac{1}{2} < 1$, in which case Line 17 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 $\mathbb{E}[X_h | X_1, \ldots, X_{h-1}] \geq 1/(8\ell)$.[11] If not, then $\mathbb{E}[X_h | X_1, \ldots, X_{h-1}] = 1$. Applying Lemma 3.1 gives:

$$\begin{aligned}
\Pr[\sum_{h \in [h_{\max}]} X_h < z+1] &= \Pr\left[\sum_{h \in [h_{\max}]} X_h \leq \left(1 - \left(1 - \frac{8\ell z}{h_{\max}}\right)\right) \frac{h_{\max}}{8\ell}\right] \\
&\leq \exp\left(-\frac{1}{2}\left(1 - \frac{8\ell z}{h_{\max}}\right)^2 \frac{h_{\max}}{8\ell}\right) \\
&\leq \exp\left(-\frac{1}{8}\frac{h_{\max}}{8\ell}\right) \qquad \text{since } h_{\max} \geq 16\ell z \\
&\leq \frac{1}{8\ell} \qquad \text{since } h_{\max} \underset{\text{Eq. 6}}{\geq} 64\ell \ln(8\ell).
\end{aligned}$$

Thus, $\Pr[B_{\text{ABORT}}] \leq \frac{1}{8\ell} + \Pr[B_{\text{REPEAT}}] + \Pr[B_{\text{BIG}}] + \Pr[\neg E_k] \leq \Pr[\neg E_k] + \frac{3}{8\ell}$.

[11] While the adversary uses the state distribution conditioned on the transcript so far, this is actually a bit more fine grained than necessary. We suspect it is enough, in this part, to condition on e.g. $X_1, \ldots, X_{h-1}$...

If E_{k-1} holds and the adversary does not abort, then it will find a set $W^{(k+1)}$. We now prove that in this case, the probability that the algorithm produces an output outside $W^{(k+1)}$ is bounded. First, note that 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 distribution. Now, if there is no splitting adversary, for the distribution over current states $\mathcal{D}$ as defined in Listing 1, then for all τ_{k+1}-step adversaries Υ, where $G(\cdot,\cdot)$ is as in Definition 4.1:

$$\Pr_{\hat{\sigma}\sim\mathcal{D}}[G(\hat{\sigma},\Upsilon) \text{ is divisive for } Q_{h-1}] \leq \frac{1}{8\ell}$$

If a given output $y \in [n]^{\tau_{k+1}}$ 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 $i \in W^{(k+1)}$. Thus in fact $y \subseteq W^{(k+1)}$. It follows that no matter which adversary is used for the next τ_{k+1} steps, the probability that the algorithm will produce an output outside $W^{(k+1)}$ is $\leq \frac{1}{8\ell}$. Consequently, $\Pr[\neg E_{k+1}] \leq \Pr[B_{\text{ABORT}}] + \frac{1}{8\ell}$, which implies $\Pr[E_{k+1}] \geq \Pr[E_k] - \frac{1}{2\ell}$. □

Lemma 4.5. *If Eq.* (1) *holds, then:*

$$z \geq \frac{\tau_\ell^2}{w_\ell 8 \ln 2}$$

Proof of Lemma 4.5. By Lemma 4.3, there is a $\geq 1 - \frac{\ell-1}{2\ell}$ probability that the adversary will reach the ℓth stage, and that all outputs in this stage will be contained in $W^{(\ell)}$. In order for the algorithm to produce valid output for the ℓth stage, the elements $(e_0,\ldots,e_{\lfloor\tau_\ell/2\rfloor})$ recorded on Lines 4 to 7 must all be distinct and disjoint from the random set v chosen at Line 1.

Let α be the probability that the algorithm produces a sequence $u := (e_0,\ldots,e_{\lfloor\tau_\ell/2\rfloor})$ which is not all distinct and disjoint from v. The total failure probability of the algorithm against the adversary is then at least $\alpha\Pr[E_\ell] \geq \alpha/2$; thus $\alpha \leq 2\delta \leq \frac{1}{2\ell}$.

By modifying the adversary to return the sequence u, we can produce a protocol for $\text{AVOID}(w_\ell, \lceil\tau_\ell/2\rceil, \lfloor\tau_\ell/2\rfloor + 1)$ with total error $\leq \alpha$, using z bits of communication. In this protocol, when Alice receives a set $A \subseteq [w_\ell]$, both Alice and Bob use public randomness to initialize random instances of the algorithm and adversary, and run them together until $W^{(\ell)}$ has been found and the ℓth stage is reached . If this does not occur, they abort. They both pick the same uniformly randomly chosen injective map $f : [W^\ell] \to [w_\ell]$. Then Alice computes $A' \leftarrow f^{-1}(A)$ and adds random elements in $W^{(k)}$ to it until $|A'| = \lceil\tau_\ell/2\rceil$; this ensures A' is uniformly randomly distributed over $\binom{W^{(\ell)}}{\lceil\tau_\ell/2\rceil}$. Alice then send $\text{SORT}(A')$ to the algorithm, and transmits the new state σ of the algorithm to Bob, who runs Lines 4 to 7 to recover a sequence $(e_0,\ldots,e_{\lfloor\tau_\ell/2\rfloor})$. Bob then outputs $f(\{e_0,\ldots,e_{\lfloor\tau_\ell/2\rfloor}\})$.

This protocol for AVOID fails if either the ℓth stage is not reached, if the ℓth stage output is not in $W^{(\ell)}$, or if $(e_0,\ldots,e_{\lfloor\tau_\ell/2\rfloor})$ are not all distinct and not disjoint from A'. The total probability of these occurring is $\leq \Pr[\neg E_\ell] + \alpha \leq \frac{\ell-1}{2\ell} + \frac{1}{2\ell} = \frac{1}{2}$. Since the protocol uses z bits of communication, it follows from the AVOID lower bound (Lemma 3.4) that:

$$\begin{aligned} z &\geq \frac{\lceil \tau_\ell/2 \rceil (\lfloor \tau_\ell/2 \rfloor + 1)}{w_\ell \ln 2} + \log(1 - 1/2) \\ &\geq \frac{\tau_\ell^2}{w_\ell 4 \ln 2} - 1 \\ &\geq \max(1, \frac{\tau_\ell^2}{w_\ell 4 \ln 2} - 1) \geq \frac{\tau_\ell^2}{w_\ell 8 \ln 2}. \end{aligned}$$

(The last line uses the fact that by Lemma 3.5, $z \geq \log(r+1) \geq 1$.) □

With some algebra, we obtain the following, proven in Appendix A.3:

Lemma 4.6.

$$z \geq \frac{\tau_\ell^2}{w_\ell 8 \ln 2} \quad \implies \quad z \geq \frac{1}{128\ell} \left(\frac{r^{\ell+1}}{n} \right)^{\frac{2}{\ell^2+3\ell-2}}$$

Combining Lemma 4.5 and Lemma 4.6 gives a lower bound on the algorithm space; but it only applies when Eq. 1 holds. Taking the minimum of the lower bound and its precondition gives a lower bound which holds for all r:

$$z \geq \min\left(\frac{1}{64\ell} r^{1/\ell}, \frac{1}{128\ell} \left(\frac{r^{\ell+1}}{n} \right)^{\frac{2}{\ell^2+3\ell-2}} \right) \geq \frac{1}{128\ell} \min\left(r^{1/\ell}, \left(\frac{r^{\ell+1}}{n} \right)^{\frac{2}{\ell^2+3\ell-2}} \right)$$

Taking the maximum of this lower bound over all ℓ in $1, 2, \ldots$, we obtain:

$$z \geq \max_{\ell \in \mathbb{N}} \frac{1}{128\ell} \min\left(r^{1/\ell}, \left(\frac{r^{\ell+1}}{n} \right)^{\frac{2}{\ell^2+3\ell-2}} \right) \tag{7}$$

The following lemma, proven in Appendix A.3, simplifies this:

Lemma 4.7.

$$\max_{\ell \in \mathbb{N}} \frac{1}{128\ell} \min\left(r^{1/\ell}, \left(\frac{r^{\ell+1}}{n} \right)^{\frac{2}{\ell^2+3\ell-2}} \right) \geq \frac{1}{128\ell} \max_{\ell \in \mathbb{N}} \left(\frac{r^{\ell+1}}{n} \right)^{\frac{2}{\ell^2+3\ell-2}} \geq \frac{\log r}{256 \log n} r^{\frac{\log r}{4 \log n}} \tag{8}$$

Summarizing, we obtain the following theorem:

Theorem 1.4. *Random tape δ-error adversarially robust algorithms for* MIF(n,r) *require*

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

bits of space, for $\delta \leq \frac{1}{128n}$.

Remark. To prove a lower bound, we required $\delta \le \frac{1}{128n}$. For larger values of δ, random tape algorithms can be much more efficient. For example, there is a $(\log t)$-space algorithm for $\text{MIF}(n,r)$ with $\Theta(r^2/t)$ error probability, which on each step randomly picks a new state (and output value) from $[t]$.

That being said, if a random tape algorithm $\mathcal{A}$ with error $\le \delta$ provides the additional guarantee that it either produces a certainly correct output or aborts,[12] one can construct a new algorithm $\mathcal{B}$ with error $O(\frac{1}{n})$ by running $\Theta(\frac{\log n}{\log 1/\delta})$ 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 $\mathcal{A}$.

Remark. If the adversary could ensure that, instead of all the next τ_{k+1} elements, a constant fraction of the next $r/2$ elements were contained in $W^{(k)}$, then in Lemma 4.5 we could probably prove that $z = \Omega\left(\frac{\tau_\ell r}{w_\ell}\right)$. This would lead to a lower bound matching, within $\text{polylog}(r,n)$ factors, the upper bound from Theorem 1.4.

Remark. The adversary of Listing 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 $\text{MIF}(n,r)$ from [Sto23] using a pseudo-random generator that fools all polynomial-time adversaries. One can also prove by minimax theorem that universal adversaries for random tape $\text{MIF}(n,r)$ algorithms can not be used to prove any stronger lower bounds than the one for random oracle algorithms.

5 Random tape upper bound

In this section, we describe an adversarially robust random tape algorithm for $\text{MIF}(n,r)$ which obtains error $\le \delta$. The algorithm, shown in Listing 2, can be implemented for almost all pairs $r < n$, requiring only $r \le n/64$ and $r \ge 4$ for its parameters to be meaningful. It can be seen as a multi-level generalization of the random oracle algorithm from [Sto23].

In order to prove that the algorithm in Listing 2 is correct, we will need some additional notation. Let $\ell, \alpha, b_1, \ldots, b_\ell, w_1, \ldots, w_\ell$ be as defined in Listing 2. It is helpful to view this algorithm as traversing over the leaves of a random tree of height ℓ, 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 $\prod_{i=1}^{\ell} w_i$, 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

[12] This guarantee essentially rules out the possibility of algorithms that randomly and blindly guess outputs. The random tape algorithm in this paper and most of those from [Sto23] provide this guarantee.

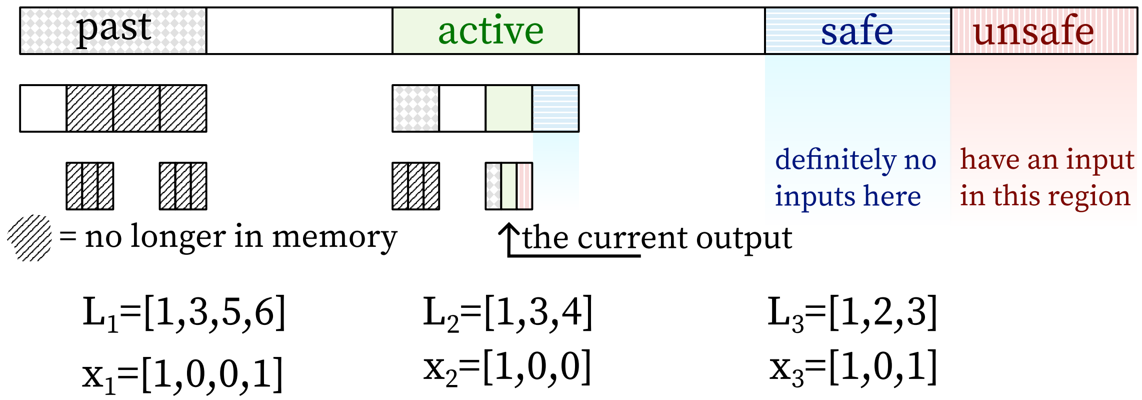

Figure 3: Diagram showing the state of the algorithm in Listing 2 and how it relates to the parts of the implicit random tree that the algorithm traverses. This example uses parameters $\ell = 3$, $w_1 = 6, w_2 = 4, w_3 = 3$, and $b_1 = 4, b_2 = 3, b_3 = 3$.

- Each leaf node u, at depth ℓ, is associated with a set of size 1, i.e., a single an unique integer in $[n]$. There are $\prod_{i=1}^{\ell} b_i$ leaf nodes in total.

See for example Figure 3. 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 $i \in [\ell]$ (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 send the value of the current leaf node, the algorithm the algorithm will mark it by setting the corresponding entry in x_ℓ to 1, thereby changing the value of the current active node. If it turns out that x_ℓ is an all-1s vector, then the adversary has sent an input for every child of the level ℓ node on the current branch, so the algorithm marks the current active child of the level $\ell - 1$ node with a 1, thereby moving the current branch to a new level ℓ node, u.[13] It the "loads the positions of the children of u" – the tree being randomly generated, this is implemented by L_ℓ being randomly sampled and x_ℓ set to be all zeros – and proceeds.

While there are $\prod_{i=1}^{\ell} b_i$ 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 such leaf nodes have been KILLED.

Listing 2 uses the following lemma to set some of its parameters; the specific rounding scheme for the values $b_2, \ldots, b_{\ell-1}$ ensures that b_1 can decrease relatively smoothly as r decreases. (Setting all $b_2 = \ldots = b_{\ell-1}$ to $\lfloor \alpha \rfloor$ can lead to having b_1 be significantly larger than necessary; setting all $b_2 = \ldots = b_{\ell-1}$ to $\lceil \alpha \rceil$ would violate the $\prod_{i=1}^{\ell} w_i \le n$ constraint.

Lemma 5.1. *Let $\alpha \ge 1$. Then for all $k \ge 0$, there exists an integer u depending on α and k so that*

$$\alpha^k \le \lceil \alpha \rceil^u \lfloor \alpha \rfloor^{k-u} \le 2\alpha^k$$

Proof of Lemma 5.1. If α is an integer, we are done. Otherwise, with

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

Similarly, $\lceil \alpha \rceil^u \lfloor \alpha \rfloor^{k-u} \le \alpha^k \lceil \alpha \rceil / \lfloor \alpha \rfloor$, which is $\le 2\alpha^k$ because $\alpha \ge 1$ implies $\lceil \alpha \rceil / \lfloor \alpha \rfloor \le 2$. □

The following lemma is straightforward but tedious, and we defer its proof to Appendix A.4.

Lemma 5.2. *The parameters of Listing 2 satisfy the following conditions:*

$$\prod_{i=2}^{\ell} b_i \ge \frac{r}{\alpha} \tag{9}$$

$$\prod_{i=2}^{\ell} b_i \le \frac{4r}{\alpha} \tag{10}$$

$$\prod_{i \in [\ell]} w_i \le n \tag{11}$$

We now prove the main lemma:

Lemma 5.3. *Listing 2 has error $\le \delta$ in the adversarial setting.*

Proof of Lemma 5.3. 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 $\ell + 2$ categories. For each $i \in [\ell]$, it could add an input which intersects the list of unrevealed child positions of the level i node, possibly killing $\prod_{j=i+1}^{\ell} b_i$ leaf nodes if it guesses correctly. It could also send the value of the current leaf

[13] If the level $\ell - 1$ node has no children marked with a 0 after this, we repeat the process at level $\ell - 2$, and so on.

Listing 2 Adversarially robust random tape algorithm for $\textsc{mif}(n,r)$ with error $\le \delta$

Requirements: $r \le n/64$ and $r \ge 4$.

Parameters: $\ell = \min(\lceil \log r \rceil, \left\lfloor 2\frac{\log(n/4)}{\log 16r} \right\rfloor)$.

$$\alpha = \begin{cases} 2 & \text{if } \lceil \log r \rceil < \left\lfloor 2\frac{\log(n/4)}{\log 16r} \right\rfloor \\ \frac{(4r)^{2/(\ell-1)}}{(n/4)^{2/(\ell(\ell-1))}} & \text{otherwise} \end{cases}$$

Let u be chosen via Lemma 5.1, so that $\alpha^{\ell-2} \le \prod_{i=2}^{\ell-1} b_i \le 2\alpha^{\ell-2}$

$b_2 = \ldots = b_u = \lceil \alpha \rceil$, and $b_{u+1} = \ldots = b_{\ell-1} = \lfloor \alpha \rfloor$

$b_1 = \min(r+1, \lceil 8\alpha \rceil + \lceil 3 \log 1/\delta \rceil)$; and $b_\ell = \left\lceil \frac{r}{\alpha^{\ell-1}} \right\rceil$

$w_1 = 16r$; and for each $i \in \{2,\ldots,\ell\}$, $w_i = \prod_{j=i}^{\ell} b_j$.

Let $\iota : [w_1] \times [w_2] \times \cdots \times [w_\ell] \to [n]$ be an arbitrary injective function

Initialization:

for $i \in [\ell]$ **do**
  $L_i \leftarrow$ random sequence without repetition in $[w_i]^{b_i}$
  $x_i \leftarrow (0,\ldots,0) \in \{0,1\}^{b}_{i}$

Update($a \in [n]$):

if $a \notin \iota^{-1}[n]$ **then return** ▷ *Any integer not in $\iota^{-1}[n]$ can never be an output*
$v_1,\ldots,v_\ell = \iota^{-1}(a)$ ▷ *Map input into $[w_1] \times \ldots \times [w_\ell]$*
For $i \in [\ell]$, define $c_i = \min |\{j : x_i[j] = 0\}|$
if for all $i \in [\ell]$, $v_i = L_i[c_i]$ **then**
  ▷ *Move to the next leaf node, sampling new child node positions as necessary*
  for $i = \ell,\ldots,1$ **do**
    $x_i[c_i] \leftarrow 1$
    if x_i is the all-1s vector **then**
      if $i = 1$ **then abort** ▷ *If we reach $i = 1$, then even the root node is full*
      $L_i \leftarrow$ random sequence without repetition in $[w_i]^{b_i}$
      $x_i \leftarrow (0,\ldots,0)$
    else break
else
  ▷ *Mark a branch as unsafe, if there was a hit*
  Let j be the smallest integer in $[\ell]$ for which $v_j \ne L_j[c_j]$.
  if $\exists y \in [b_j]$ for which $L_j[y] = v_j$ **then**
    $x_j[y] \leftarrow 1$

Output $\to [n]$:

For $i \in [\ell]$, define $c_i = \min |\{j : x_i[j] = 0\}|$
return $\iota[(L_1[c_1], L_2[c_2], \ldots, L_\ell[c_\ell])]$

node, thereby killing it (and only it). Finally, the adversary's input could be entirely wasted (outside $\iota([w_1] \times \ldots \times [w_\ell]$, 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 $i \in \{2, \ldots, \ell\}$, 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, $\prod_{j=i+1}^{\ell} b_j$), and remove up to that amount of charge from the node. The definitions of $(w_j)_{j=2,\ldots,\ell}$ ensure that $\prod_{j=i+1}^{\ell} b_j = w_j/b_j$.

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 $i \in \{2, \ldots, \ell-1\}$, 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 $\hat{w} \le w_j - u$ and $\hat{b} \le b_j - u$. 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 $\ge (w_j - u - \hat{w}) - (\frac{w_j}{b_j})(b_j - u - \hat{b}) = \hat{b}\frac{w_j}{b_j} - \hat{w} + (\frac{w_j}{b_j} - 1)u \ge \hat{b}\frac{w_j}{b_j} - \hat{w}$, 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 payoff is:

$$\mathbb{E}[K_t | K_1, \ldots, K_{t-1}] = \frac{\hat{b}}{\hat{w}} \max(0, \frac{w_j}{b_j} - \max(0, \hat{b}\frac{w_j}{b_j} - \hat{w})) \underset{\text{by Lemma 5.4}}{\le} 1$$

(Note: when the level is ℓ and $b_j = w_j$, we in fact we have $K_t = 1$ always; but we do not need this stronger fact.)

If the level is 1, then let $J \subseteq [16r]$ give the set of probed subtree positions, and H give the set of revealed subtree positions; since there are $\le r$ queries, $|J|, |H|$ are both $\le r$, and the probability of a query in an unexplored region to hit is $\le \frac{b_1}{16r - |J \cup H|} \le \frac{b_1}{14r}$. There is no charge scheme, so

$$\mathbb{E}[K_t | K_1, \ldots, K_{t-1}] \le \frac{b_1}{14r} \cdot \prod_{j=2}^{\ell} b_j$$

Thus in all cases, $\mathbb{E}[K_t | K_1, \ldots, K_{t-1}] \le \max(1, \prod_{j=1}^{\ell} b_j / 14r)$.

Note: the total charge deposited on mid-level nodes is $\le r$. The algorithm is guaranteed to succeed if the total number of leaves killed is less than the total number of leaves; i.e, if $\sum_{i \in [r]} K_t + r \le \prod_{j=1}^{\ell} b_j$. Note that by Lemma 5.2 and the definition of b_1, $\prod_{j=1}^{\ell} b_j \ge b_1 r/\alpha \ge 8r$.

Consequently,

$$r+7\mathbb{E}[\sum_{t=1}^{r} K_t] \le 8\max(r, \frac{1}{14}\prod_{j=1}^{\ell} b_j) \le 8\max(\frac{1}{8}, \frac{1}{14})\prod_{j=1}^{\ell} b_j \le \prod_{j=1}^{\ell} b_j$$

Now let $D_t = K_t / \prod_{j=2}^{\ell} b_j$, so that each $D_t \in [0,1]$. Writing events in terms of D_t lets us use Lemma 3.1 to bound the probability that too many leaves are killed:

$$\begin{aligned}
\Pr\left[r + \sum_{t\in[r]} K_t \ge \prod_{i=1}^{\ell} b_i\right] &\le \Pr\left[\sum_{t\in[r]} K_t \ge 7\max(r, \frac{1}{14}\prod_{j=1}^{\ell} b_j)\right] \\
&\le \Pr\left[\sum_{t\in[r]} D_t \ge 7\max(r/\prod_{j=2}^{\ell} b_j, \frac{b_1}{14})\right] \\
&\le \exp(-\frac{6^2}{2+6}\max(r/\prod_{j=2}^{\ell} b_j, \frac{b_1}{14})) \\
&\le \exp(-\frac{9b_1}{28}) \le 2^{b_1 \frac{9}{28\ln 2}} \le 2^{\lceil 3\log 1/\delta\rceil \frac{9}{28\ln 2}} \le \delta\,. \qquad \square
\end{aligned}$$

In the preceding proof, we used the following:

Lemma 5.4. *Let $\hat{b}, \hat{w}, b, w$ be positive, and $\hat{b} \ge 1$. Then:*

$$\frac{\hat{b}}{\hat{w}}\left(\max(0, \frac{w}{b} - \max(\hat{b}\frac{w}{b} - \hat{w}, 0))\right) \le 1$$

Proof of Lemma 5.4. If $\frac{\hat{b}}{\hat{w}} \le \frac{b}{w}$, then:

$$\frac{\hat{b}}{\hat{w}}\left(\max(0, \frac{w}{b} - \max(\hat{b}\frac{w}{b} - \hat{w}, 0))\right) \le \frac{\hat{b}}{\hat{w}}\frac{w}{b} \le 1\,.$$

Otherwise, $\frac{\hat{b}}{\hat{w}} \ge \frac{b}{w}$, and:

$$\begin{aligned}
&\frac{\hat{b}}{\hat{w}}\left(\max(0, \frac{w}{b} - \max(\hat{b}\frac{w}{b} - \hat{w}, 0))\right) \\
&\le \frac{\hat{b}}{\hat{w}}(\frac{w}{b} - (\hat{b}\frac{w}{b} - \hat{w})) == \frac{\hat{b}}{\hat{w}}(\frac{w}{b} - \hat{b}(\frac{w}{b} - \frac{\hat{w}}{\hat{b}})) \\
&\le \frac{\hat{b}}{\hat{w}}(\frac{w}{b} - 1(\frac{w}{b} - \frac{\hat{w}}{\hat{b}})) = \frac{\hat{b}}{\hat{w}}\frac{\hat{w}}{\hat{b}} = 1\,. \qquad \text{since } \hat{b} \ge 1 \text{ and } \frac{w}{b} - \frac{\hat{w}}{\hat{b}} \ge 0 \qquad \square
\end{aligned}$$

Lemma 5.5. *Listing 2 uses*

$$O\left(\left\lceil \frac{(4r)^{2/(\ell-1)}}{(n/4)^{2/(\ell(\ell-1))}} \right\rceil (\log r)^2 + \min(r, \log 1/\delta)\log r\right) \tag{12}$$

bits of space, where $\ell = \min\left(\lceil \log r \rceil, \left\lfloor 2\frac{\log(n/4)}{\log(16r)} \right\rfloor\right)$*. A weaker upper bound on this is:*

$$O(r^{2\frac{\log r}{\log n}}(\log r)^2 + \min(r, \log 1/\delta)\log r)$$

The proof of Lemma 5.5 is a straightforward calculation, which we defer to Appendix A.4.

If $r \geq 4$ and $r \leq n/64$, then Lemma 5.5 and Lemma 5.3 together show that Listing 2 has error $\leq \delta$ and space usage as bounded by Eq. 12. To handle the cases where $r < 4$ and $r > n/64$, one can instead use the simple deterministic algorithm for $\textsc{mif}(n,r)$ from [Sto23], using only r bits of space. As this is in fact less than the space upper bound from Eq. 12, it follows that Eq. 12 gives an upper bound on the space needed for a random tape, adversarially robust $\textsc{mif}(n,r)$ algorithm for any setting of parameters. Formally:

Theorem 1.3. *There is a family of adversarially robust random tape algorithms, where for* $\textsc{mif}(n,r)$ *the corresponding algorithm has* $\leq \delta$ *error and uses*

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

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

Remark. Listing 2 does not use the most optimal assignment of the parameters $b_\ell, \ldots, b_2$; constant-factor improvements in space usage are possible if one sets $b_\ell, \ldots, b_2$ to be roughly in an increasing arithmetic sequence, but this would make the analysis more painful.

Remark. In exchange for a constant factor space increase, one can adapt Listing 2 to produce an increasing sequence of output values. Similar adjustments can be performed for other MIF algorithms.

6 Pseudo-deterministic lower bound

In this section, we prove a space lower bound for pseudo-deterministic streaming algorithms; in particular, for the most general (random oracle) type of them. Let $\mathcal{A}$ be a random oracle pseudo-deterministic algorithm for $\textsc{mif}(n,r)$ using z bits of state, which has worst case failure probability $\delta \leq \frac{1}{3}$. Let $\Pi : [n]^r \to [n]$ be the function giving the canonical output of $\mathcal{A}$ after processing a stream of length r, and let $S = \Pi([n]^t)$ be the set of all canonical outputs. Clearly $|S| \leq n$, and the later Lemma 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 $z \leq \frac{r}{18p}$. The threshold $\frac{r}{18p}$, 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 $z \leq \frac{r}{18p}$, unless stated otherwise.

Let

$$p = \left\lceil \sqrt{\frac{5r\log(64|S|)}{3z\log 1/(2\delta)}} \right\rceil \qquad \text{and} \qquad \ell = \left\lfloor \frac{r}{18zp} \right\rfloor \le \left\lfloor \frac{r}{2\lceil 4\ln 2(zp+2)\rceil} \right\rfloor \tag{13}$$

$$t_\ell = \ldots = t_2 = \lceil 4\ln 2(zp+2)\rceil \qquad \text{and} \qquad t_1 = r - \sum_{k=2}^{\ell} t_k \tag{14}$$

$$w_1 = t_1 + 1 \qquad \text{and} \qquad \forall i \ge 2, w_i = \left\lceil \frac{t_i}{2\ln 2(zp+2)} w_{i-1} \right\rceil \tag{15}$$

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 3.3, 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, $\le \varepsilon$, where $\varepsilon \le (2\delta)^{p/30}$. Using the above definitions, we have:

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

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

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 $[n]^r \to [n]$, 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:

$$\forall x \in [n]^r : \Pr_{A\sim\mathcal{D}}[A(x) \ne \Pi(x)] \le \varepsilon \tag{16}$$

We describe a randomized algorithm FINDCOMMONOUTPUTS (short: FCO) which, when run on Π and a random matrix $C \in [1,2)^{\ell\times\mathbb{N}}$, 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 Listing 3. 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.

Listing 3 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.

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

Lemma 6.1. *Let $k \in \{1,\ldots,\ell\}$, and $x \in [n]^{t_\ell+\ldots+t_{k+1}}$.*

$$\Pr_{A\sim\mathcal{D},C\sim[1,2)^{\ell\times\mathbb{N}}}[\text{FCO}(A_x,C,k) = \text{FCO}(\Pi_x,C,k)] \ge 1-\varepsilon_k$$

Listing 3 The procedure to compute a set for Lemma 6.1

Let $t_1, \ldots, t_\ell$, $w_1, \ldots, w_\ell$ be integer parameters, and S the set of valid outputs

FINDCOMMONOUTPUTS(B, C, k) ▷ *A.k.a.* FCO(B, C, k)

▷ *B is a function from $[n]^{t_k+\cdots+t_1}$ to $[n]$*
▷ *C is a vector in $[1,2)^{\ell\times\mathbb{N}}$*
▷ *The output will be a subset of S of size w_k*
if $k = 1$ **then**
  $e_0 \leftarrow B((1,1,1,\ldots,1))$
  for i in $1,\ldots,t_1$ **do**
    $e_i \leftarrow B((e_0,\ldots,e_{i-1},1,\ldots,1))$
  return $\{e_0, e_1, \ldots, e_{t_1}\}$
else
  for each $y \in \text{SORT}\binom{S}{t_k}$ **do**
    ▷ *Recall B_y is notation for the partial application of B with prefix y*
    $T_{B,y} \leftarrow$ FINDCOMMONOUTPUTS$(B_y, C, k-1)$ ▷ *note $|T_{B,y}| = w_{k-1}$*
  $Q_0 \leftarrow T_{B,(1,2,\ldots,t_k)}$ ▷ *choice of seed vector is arbitrary*
  for h in $1,2,\ldots,\left\lfloor \frac{4w_k}{w_{k-1}} \right\rfloor - 1$ **do**
    for each $j \in S$ **do**
      $f_j^{(h)} \leftarrow \left| \left\{ y \in \text{SORT}\binom{Q_{h-1}}{t_k} : j \in T_{B,y} \right\} \right|$
    $P_h \leftarrow \left\{ j \in S : f_j^{(h)} \geq \frac{C_{k,h} w_{k-1}}{16|S|} \left| \binom{Q_{h-1}}{t_k} \right| \right\}$
    $Q_h \leftarrow Q_{h-1} \cup P_h$
    if $|Q_h| \geq w_k$ **then**
      return the w_k smallest elements in Q_h
  return arbitrary subset of S of size w_k (failure)

Furthermore, $\text{FCO}(\Pi_x, C, k)$ *will be disjoint from* x *and a subset of* S*, and* $\text{FCO}(\cdot,\cdot,k)$ *will always output some set of size* w_k.

We now prove Lemma 6.1, by induction on k. First, we prove the case $k = 1$:

Lemma 6.2. *Lemma 6.1 holds for* $k = 1$.

Proof of Lemma 6.2. Let $e_0, \ldots, e_{t_1}$ be the values of the variables on Lines 5 to 7 of Listing 3 when $\text{FCO}(\Pi, C, 1)$ is called; note that these do not depend on C. Define for $i \in \{0, \ldots, t_1\}$ vectors $s_i = (e_0, \ldots, e_{i-1}, 1, \ldots, 1)$, so that $s_0 = (1, 1, \ldots, 1)$, and $s_{t_1} = (e_0, \ldots, e_{t_1-1})$. Then if for all $i \in \{0, \ldots, t_1\}$, $A_x(s_i) = \Pi_x(s_i)$, the value of $\text{FCO}(A_x, C, 1)$ will exactly match $\text{FCO}(\Pi_x, C, 1)$. By a union bound,

$$\Pr_{A \sim \mathcal{D}, C}[\text{FCO}(A_x, C, k) \neq \text{FCO}(\Pi_x, C, k)] \leq \sum_{i=0}^{t_1} \Pr_{A \sim \mathcal{D}}[A_x(s_i) \neq \Pi_x(s_i)] \underset{\text{Eq. (16)}}{\leq} (t_1 + 1)\varepsilon = \varepsilon_1$$

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

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

Lemma 6.3. *Let* $x \in [n]^{t_\ell + \cdots + t_{k+1}}$. *When computing* $\text{FCO}(\Pi_x, C, k)$*, 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 20, not Line 21.*

Proof of Lemma 6.3. Assume for sake of contradiction that $|Q_{h-1}| < w_k$ and $|P_h \setminus Q_h| \leq \left\lfloor \frac{1}{4} w_{k-1} \right\rfloor$. Then we can use the algorithm $\mathcal{A}$ to implement a protocol for the one-way communication problem $\text{AVOID}(|Q_{h-1}|, t_k, \left\lceil \frac{1}{2} w_{k-1} \right\rceil)$, 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 $|W| = t_k$, she constructs a sequence $v = x.\text{SORT}(W)$ in $[n]^{t_\ell + \cdots + t_k}$. 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 the public randomness, they 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 $\tilde{A}_{x.\text{SORT}(W)} : [n]^{t_{k-1} + \cdots + t_1} \to [n]$; Bob then samples a random $C \in [1,2)^{\ell \times \mathbb{N}}$, and computes $V = \text{FCO}(\tilde{A}_{x.\text{SORT}(W)}, C, k-1)$, which is a subset of S. If $|V \cap Q_{h-1}| \geq \left\lceil \frac{1}{2} w_{k-1} \right\rceil)$, Bob outputs the smallest $\left\lceil \frac{1}{2} w_{k-1} \right\rceil$ entries of $V \cap Q_{h-1}$. Otherwise, Bob outputs an arbitrary set of size $\left\lceil \frac{1}{2} w_{k-1} \right\rceil$.

First, we observe that for any value of $\text{SORT}(W)$, the distribution of $\tilde{A}_{x.\text{SORT}(W)}$ is exactly the same as the distribution of $A_{x.\text{SORT}(W)}$, 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 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 Listing 3. In particular, we have:

$$\begin{aligned} \Pr_{y,C}\left[|T_y \cap Q_{h-1}| < \left\lceil \frac{1}{2} w_{k-1} \right\rceil\right] &= \Pr_{y,C}\left[|T_y \setminus Q_{h-1}| > \left\lfloor \frac{1}{2} w_{k-1} \right\rfloor\right] \\ &\leq \Pr_{y,C}\left[|T_y \setminus P_h \setminus Q_{h-1}| > \left\lfloor \frac{1}{2} w_{k-1} \right\rfloor - \left\lfloor \frac{1}{4} w_{k-1} \right\rfloor\right] \qquad (17) \\ &\leq \Pr_{y,C}\left[|T_y \setminus P_h| \geq \frac{1}{4} w_{k-1}\right] \end{aligned}$$

(The inequality on Eq. (17) 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 3.4, 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 14 to 20, $|Q_h| = |Q_{h-1} \cup P_h| = |Q_{h-1}| + |P_h \setminus Q_{h-1}| \geq |Q_{h-1}| + \lceil \frac{1}{4} w_{k-1} \rceil$, 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. □

Lemma 6.4. *For $k > 1$, $x \in [n]^{t_\ell + \ldots + t_{k+1}}$, $\text{FCO}(\Pi_x, C, k)$ is disjoint from x and a subset of S; and for all A, C, k, $\text{FCO}(A_x, C, k)$ outputs a set of size w_k.*

Proof of Lemma 6.4. That $\text{FCO}(A_x, C, k)$ 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 20, or a set of size w_k via Step 21.

By Lemma 6.1 at $k-1$, the sets $T_{A_x,y}$ chosen on Line 12 are always subsets of S and disjoint from $x.y$, and hence disjoint from x. Per Lemma 6.3, FINDCOMMONOUTPUTS will return a subset of Q_h using Line 20, 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 $(T_{A_x,y})_{y \in \text{SORT}\binom{Q_{h'-1}}{t_k}}$, and are thereby also in S. □

Lemma 6.5. *For $k > 1$, and all $x \in [n]^{t_\ell + \ldots + t_{k+1}}$,*

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

Proof of Lemma 6.5. The proof of the lemma follows from the observation that, when computing $\text{FCO}(A_x, C, k)$, even if a fraction of the recursive calls to $\text{FCO}(A_{x.y}, C, k-1)$ produced incorrect outputs, the values for Q_0 and $(P_h)_{h \geq 1}$ will likely match those computed when $\text{FCO}(\Pi_x, C, k)$ is called.

Henceforth, we indicate variables from the computation of $\text{FCO}(\Pi_x, C, k)$ without a tilde, and variables from the computation of $\text{FCO}(A_x, C, k)$ 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

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

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

Say $\text{FCO}(\Pi_x, C, k)$ returns from the loop at iteration $h^\star$. The output of $\text{FCO}(A_x, C, k)$ will equal $\text{FCO}(\Pi_x, C, k)$ if $Q_0 = \widetilde{Q}_0$ and for all $h \in [h^\star]$, we have $P_h = \hat{P}_h$. (If this occurs, then as $Q_0 = \widetilde{Q}_0$, $\hat{P}_1 = \widetilde{P}_1$, so $Q_1 = Q_0 \cup P_1 = \widetilde{Q}_0 \cup \widetilde{P}_1 = \widetilde{Q}_1$, and as $Q_1 = \widetilde{Q}_1$, $\hat{P}_2 = \widetilde{P}_2$, and so on.) By Lemma 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 $y \in \text{SORT}\binom{Q_{h-1}}{t_k}$ 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 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 $\hat{f}_j^{(h)}$:

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

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:

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

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

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

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

Finally, we observe that Lemmas 6.2 to 6.5, together imply Lemma 6.1.

A consequence of Lemma 6.1 is that $\text{FCO}(\Pi, C, \ell)$ will output a set of size w_ℓ which is a subset of S.

Lemma 6.6. *Even if* $18zp > r$*, we have:*

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

Proof of Lemma 6.6. If, as we have assumed, $18zp \leq r$, then for each $i \geq 2$, by Eqs. (14) and (15),

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

Also, by Eqs. (13) and (14),

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

Since $w_\ell \leq |S|$, we obtain:

$$|S| \geq w_\ell \geq 2^\ell \lceil r/2\rceil \quad \Longrightarrow \quad \log\frac{2|S|}{r} \geq \ell \geq \frac{r}{36zp} \quad \Longrightarrow \quad zp \geq \frac{r}{36 \log \frac{2|S|}{r}} \tag{18}$$

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

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

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

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

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

$$z \geq \frac{r}{36 \log \frac{2|S|}{r}}$$

while otherwise

$$z \geq \frac{r\log 1/(2\delta)}{8460\log(64|S|)(\log\frac{2|S|}{r})^2}$$

Computing a common lower bound for the two cases gives:

$$z \geq \frac{r}{8460 \log \frac{2|S|}{r}} \min\left(1, \frac{\log(1/2\delta)}{\log(64|S|)\log \frac{2|S|}{r}}\right) \tag{19}$$

□

Lemma 6.7. $|S| < 2^{z+1}$.

Proof of Lemma 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 1.1. *Pseudo-deterministic δ-error random oracle algorithms for* MIF(n,r) *require*

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

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

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

Using Lemma 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 6.7, and four each of four cases corresponding to different branches of min and max, solving to find a lower bound on z. The proof is given in Appendix A.5.

Remark. For $\delta \leq 2^{-r}$, Theorem 1.1 reproduces the deterministic algorithm space lower bound for MIF(n,r) from [Sto23] within a constant factor.

6.1 Implications for adversarially robust random seed algorithms

The following result, paraphrased, from [Sto23] relates the random seed adversarially robust space complexity with the pseudo-deterministic space complexity:

Theorem 6.8 ([Sto23])**.** *Let $S^{PD}_{1/3}(n,r)$ give a space lower bound for a pseudo-deterministic algorithm for* MIF(n,r) *with error $\leq 1/3$. Then an adversarially robust random start algorithm with error $\delta \leq \frac{1}{6}$, if it uses z bits of space, must have $z \geq S^{PD}_{1/3}(n, \lfloor\frac{r}{2z+2}\rfloor)$.*

Corollary 1.2. *Adversarially robust random seed algorithms for* MIF(n,r) *with error $\leq \frac{1}{6}$ require $\Omega\big(\sqrt{r/(\log n)^3} + r^{1/5}\big)$ bits of space.*

This follows by combining Theorem 6.8 and Theorem 1.1 and performing some algebra; a proof is given in Appendix A.5.

Acknowledgements

M. S. would like to thank Jonathan Conroy for helpful feedback.

References

[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*, pages 15–27, 2022.

[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*, pages 141–153, 2023.

[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.

[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.

[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.

[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.

[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.

[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.

[Fei19] Uriel Feige. A randomized strategy in the mirror game. *arXiv preprint arXiv:1901.07809*, 2019.

[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.

[GGS23] Ofer Grossman, Meghal Gupta, and Mark Sellke. Tight space lower bound for pseudo-deterministic approximate counting. *arXiv preprint arXiv:2304.01438*, 2023.

[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.

[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.

[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.

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

[JP83] Kumar Joag-Dev and Frank Proschan. Negative association of random variables, with applications. *Ann. Stat.*, 11(1):286–295, 1983.

[JW23] Rajesh Jayaram and David P Woodruff. Towards optimal moment estimation in streaming and distributed models. *ACM Trans. Alg.*, 19(3):1–35, 2023.

[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.

[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.

[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.

[New91] Ilan Newman. Private vs. common random bits in communication complexity. *Inform. Process. Lett.*, 39(2):67–71, 1991.

[Nis90] Noam Nisan. Pseudorandom generators for space-bounded computation. In *Proc. 22nd Annual ACM Symposium on the Theory of Computing*, pages 204–212, 1990.

[Nis93] Noam Nisan. On read once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993.

[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. Full version at arXiv:2211.05170v1.

[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.

A Appendix

A.1 Explanation of Table 1

Table 1 is constructed from a number of results, and some cells, like that for random tape algorithms in the adversarial setting, are formed by combining multiple results.

Random oracle, static setting. From [Sto23], the actual algorithm space complexity is $O(\log_{n/r}(1/\delta))$ for $r < n/2$.

Random seed/tape, static setting. Explicitly storing the random list from the random oracle static setting algorithm from [Sto23] gives $O(\log_{n/r}(1/\delta)\log n)$ space.

Deterministic. The upper bound from [Sto23] is $O(\frac{r\log r}{\log n} + \sqrt{r\log r})$; the lower bound is $\Omega(\frac{r}{\log 2n/r} + \sqrt{r})$.

Adversarial setting, random oracle. The upper bound from [Sto23] is $O((1+r^2/n+\ln\frac{1}{\delta})\log r)$, while the lower bound is $\Omega(\frac{r^2}{n} + \log(1-\delta))$. One can easily prove, however, that any adversarially robust algorithm requires $\geq \log(r+1)$ bits of space, by considering the adversary that echos back to the algorithm whatever its last output was.

Adversarial setting, random tape. Uses the upper bound from Theorem 1.3, and the lower bound from Theorem 1.4. The $\Omega(\log r)$ lower bound from Lemma 3.5 is stronger than Theorem 1.4 for $r \ll n$ but hidden by the $\widetilde{\Theta}(\cdot)$.

Adversarial setting, random seed. The lower bound from Corollary 1.2 provides an $r^{\Omega(1)}$ term; the $\Omega(r^2/n)$ part comes from the general adversarially robust lower bound of [Sto23]. The arXiv version of [Sto23] provides an $\widetilde{O}(r^2/n+\sqrt{r})$ algorithm and explains how to improve this to $\widetilde{O}(r^2/n+\sqrt{r/\text{polylog}(n)}+r^{1/3})$ using the deterministic algorithm.

Pseudo-deterministic. Uses the lower bound from Theorem 1.1. The best known upper bound comes from the deterministic algorithm.

A.2 Proofs of useful lemmas

Here we provide proofs of the results in Section 3.2 for which we haven't found an external source.

Proof of Lemma 3.1. We only prove this elementary lemma since it's hard to find a source for. The proof is modeled off that in [Sto23], which only addresses one direction.

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

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

The innermost term $\mathbb{E}[e^{zX_t}|X_1,\ldots,X_{t-1}]$ is, by 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 bound the X_{t-1} term, and so on. Thus we continue the chain of inequalities to get:

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

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. That $p_t e^z + (1-p_t) \leq e^{p_t(e^z-1)}$ still holds for negative z, so:

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

□

Proof of Lemma 3.3. 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:

$$\begin{aligned}
\Pr[\sum_{i\in[p]} Y_i \geq \frac{1}{2}p] = \Pr[\sum_{i\in[p]} Y_i \geq (1+\alpha)\delta p] &\leq \exp\left(-((1+\alpha)\ln(1+\alpha)-\alpha)\delta p\right) \\
&\leq \exp\left(-0.073((1+\alpha)\ln(1+\alpha))\delta p\right) \qquad \text{since } \alpha \geq 1/6 \\
&\leq \exp\left(-\frac{0.073}{2\delta}\ln\frac{1}{2\delta}\delta p\right) = (2\delta)^{0.036p}
\end{aligned}$$

□

A.3 Mechanical proofs for Section 4

Proof of Lemma 4.6. As a consequence of the precondition Eq. (1), we are guaranteed that for each $k \in [\ell]$ that $\tau_\ell \geq 1$. Thus, using $x \geq 1 \implies \lfloor x \rfloor \geq x/2$,

$$\tau_k \geq \frac{1}{2}\frac{r}{h_{\max}^{k-1}}$$

Iteratively applying the upper bound on w_ℓ gives:

$$\begin{aligned}
z &\geq \frac{\tau_\ell^2}{8\ln 2}\frac{1}{w_\ell} \geq \frac{\tau_\ell^2}{8}\frac{\prod_{k=1}^{\ell-1}\tau_k}{(60z)^{\ell-1}n} \\
&\geq \frac{r^{\ell+1}}{8\cdot 2^{\ell+1}\cdot h_{\max}^{2(\ell-1)+(\ell-2)+(\ell-3)+\ldots+1+0}(60z)^{\ell-1}n} \\
&\geq \frac{r^{\ell+1}}{8\cdot 2^{\ell+1}\cdot(64\ell z)^{(\ell^2+\ell-2)/2}(60z)^{\ell-1}n} \qquad \text{by def of } h_{\max} \\
&\geq \frac{r^{\ell+1}}{8\cdot 2^{\ell+1}\cdot(64\ell z)^{(\ell^2+\ell-2)/2}(64z)^{\ell-1}n} \qquad \text{since } 60 \leq 64
\end{aligned}$$

Rearranging to put all z terms on the left gives:

$$z(64z)^{\frac{\ell^2+3\ell-4}{2}} \geq \frac{r^{\ell+1}}{16\cdot 2^\ell \ell^{(\ell^2+\ell-2)/2}n}$$

which implies

$$z \geq \frac{1}{64}\left(\frac{64r^{\ell+1}}{16n}\right)^{\frac{2}{\ell^2+3\ell-2}}\frac{1}{2^{\ell\cdot\frac{2}{\ell^2+3\ell-2}}\ell^{\frac{\ell^2+\ell-2}{\ell^2+3\ell-2}}} \geq \frac{1}{128\ell}\left(\frac{r^{\ell+1}}{n}\right)^{\frac{2}{\ell^2+3\ell-2}}$$

In the last inequality, we used the facts that $\frac{\ell^2+\ell-2}{\ell^2+3\ell-2} \leq 1$ and that $\frac{2\ell}{\ell^2+3\ell-2} \leq 1$ for $\ell \geq 1$. □

Proof of Lemma 4.7. The left branch of the $\min(\cdot,\cdot)$ terms in Eq. 8 is actually unnecessary. For any integer $\lambda \geq 2$, say that

$$\frac{1}{\lambda}\left(\frac{r^{\lambda+1}}{n}\right)^{\frac{2}{\lambda^2+3\lambda-2}} = \max_{\ell\in\mathbb{N}} \frac{1}{\ell}\left(\frac{r^{\ell+1}}{n}\right)^{\frac{2}{\ell^2+3\ell-2}}$$

Then in particular,

$$\frac{1}{\lambda}\left(\frac{r^{\lambda+1}}{n}\right)^{\frac{2}{\lambda^2+3\lambda-2}} \geq \frac{1}{\lambda-1}\left(\frac{r^{(\lambda-1)+1}}{n}\right)^{\frac{2}{(\lambda-1)^2+3(\lambda-1)-2}} \geq \frac{1}{\lambda}\left(\frac{r^{\lambda}}{n}\right)^{\frac{2}{\lambda^2+\lambda-4}}$$

which implies

$$n^{2\lambda+2} = n^{(\lambda^2+3\lambda-2)-(\lambda^2+\lambda-4)} \geq r^{\lambda\cdot(\lambda^2+3\lambda-2)-(\lambda+1)\cdot(\lambda^2+\lambda-4)} = r^{\lambda^2+\lambda+4}$$

hence we have $r \leq n^{\frac{2\lambda+2}{\lambda^2+\lambda+4}}$.

On the other hand, we have

$$\frac{1}{\lambda}r^{1/\lambda} \geq \frac{1}{\lambda}\left(\frac{r^{\lambda+1}}{n}\right)^{\frac{2}{\lambda^2+3\lambda-2}} \qquad \Longleftrightarrow \qquad n^{\lambda} \leq r^{(\lambda+1)\lambda-\frac{\lambda^2+3\lambda-2}{2}} = r^{\frac{\lambda^2-\lambda+2}{2}}$$

so the left branch of the $\min(\cdot,\cdot)$ in Eq. 8 is only smaller when $r \geq n^{\frac{2\lambda}{\lambda^2-\lambda+2}}$. 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. 8 is only smaller than the right when the entire term is not the maximum. Thus

$$\text{LHS} \geq \max_{\ell\in\mathbb{N}} \frac{1}{128\ell}\left(\frac{r^{\ell+1}}{n}\right)^{\frac{2}{\ell^2+3\ell-2}} \tag{20}$$

To get a looser but more easily comprehensible lower bound,[14] we observe that since $(\ell+1)^2 \geq$

[14]The constant $\frac{1}{4}$ is not the tightest possible; to get a better constant, one could use the convexity of Eq. 20 and compute the constant by taking the minimum value over all "corner" points where two adjacent terms in the $\max_\ell$ are equal.

$\frac{\ell^2+3\ell-2}{2}$,

$$\max_{\ell\in\mathbb{N}} \frac{1}{128\ell}\left(\frac{r^{\ell+1}}{n}\right)^{\frac{2}{\ell^2+3\ell-2}} \geq \max_{\ell\in\mathbb{N}} \frac{1}{128\ell}\left(\frac{r^{\ell+1}}{n}\right)^{\frac{1}{(\ell+1)^2}}$$
$$\geq \left[\frac{1}{128\ell}\left(\frac{r^{\ell+1}}{n}\right)^{\frac{1}{(\ell+1)^2}}\right]_{\ell=\left\lceil 2\frac{\log n}{\log r}\right\rceil-1}$$
$$\geq \left[\frac{1}{128\ell}n^{\frac{1}{(\ell+1)^2}}\right]_{\ell=\left\lceil 2\frac{\log n}{\log r}\right\rceil-1}$$
$$\geq \frac{\log r}{256\log n}n^{\left(\frac{\log r}{2\log n}\right)^2} = \frac{\log r}{256\log n}r^{\frac{\log r}{4\log n}}$$

□

A.4 Mechanical proofs for Section 5

Proof of Lemma 5.2. First, we handle the case where $\lceil\log r\rceil < \left\lfloor 2\frac{\log(n/4)}{\log 16r}\right\rfloor$. Then $\ell = \lceil\log r\rceil \leq \left\lfloor 2\frac{\log(n/4)}{\log 16r}\right\rfloor - 1$, and $\alpha = 2$. Note that $\frac{r}{2^{\ell-1}} \geq 1$ since $2^{\ell-1} \leq 2^{\lceil r\rceil-1} \leq 2r/2 = r$.

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

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

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

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

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

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

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

We now prove Eq. (9). Because $\prod_{i=2}^{\ell-1} b_i \geq \alpha^{\ell-2}$,

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

For the Eq. (11), we observe that

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

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

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

Finally, we prove Eq. (11). As noted above,

$$b_\ell \leq \frac{2r}{\alpha^{\ell-1}} \leq \frac{4r}{\alpha \prod_{i=2}^{\ell-1} b_j}$$

Applying this fact to bound the left hand side of Eq. (11) gives:

$$\begin{aligned}
\prod_{i\in[\ell]} w_i &= (16r) \prod_{i=2}^{\ell} \prod_{j=i}^{\ell} b_j = 16r (b_\ell)^{\ell} \prod_{i=2}^{\ell-1} \prod_{j=i}^{\ell-1} b_j \\
&\leq 16r \left(\frac{4r}{\alpha \prod_{i=2}^{\ell-1} b_j}\right)^{\ell-1} \prod_{i=2}^{\ell-1} \prod_{j=i}^{\ell-1} b_j \\
&\leq \frac{4 \cdot (4r)^{\ell}}{\alpha^{\ell-1}} \frac{1}{\prod_{i=2}^{\ell-1} \prod_{j=2}^{i-1} b_j} \\
&\leq \frac{4 \cdot (4r)^{\ell}}{\alpha^{\ell-1}} \frac{1}{\alpha^{(\ell-1)(\ell-2)/2}} \qquad \text{since } \prod_{j=2}^{\ell-1} b_j \geq \alpha^{\ell-2} \text{ and } b_2 \geq b_3 \geq \ldots \geq b_{\ell-1} \\
&= \frac{4 \cdot (4r)^{\ell}}{\alpha^{\ell(\ell-1)/2}} = \frac{4 \cdot (4r)^{\ell}}{\frac{(4r)^{\ell}}{n/4}} = n.
\end{aligned}$$

□

Proof of Lemma 5.5. Listing 2 only stores two types of data: for each $i \in [\ell]$, 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:

$$\sum_{i \in [\ell]} b_i \log(2w_i) \leq b_1 \log(2w_1) + \sum_{i=2}^{\ell} b_i \log(2w_i)$$
$$\leq b_1 \log(32r) + \sum_{i=2}^{\ell} b_i \log(2\prod_{j=i}^{\ell} b_i) \leq \sum_{i=1}^{\ell} b_i \log(32r)$$

since by Eq. 10, $\prod_{j=i}^{\ell} b_i \leq 4r$.

We now observe that $b_\ell \leq \lceil \alpha \rceil$. If $\ell = \lceil \log r \rceil$, then $\alpha = b_2 = \ldots = b_{\ell-1} = 2$ and $b_\ell = \lceil r/\alpha^{\ell-1} \rceil \leq 2$. On the other hand, if $\ell = \left\lfloor 2\frac{\log(n/4)}{\log(16r)} \right\rfloor$, then $\alpha = \frac{(4r)^{2/(\ell-1)}}{(n/4)^{2/(\ell(\ell-1))}}$. We have:

$$(4r)^{\ell+1} = (4r)^{\left\lfloor 2\frac{\log(n/4)}{\log(16r)} \right\rfloor + 1} \geq (4r)^{2\frac{\log(n/4)}{\log(16r)}} = (n/4)^2$$

which implies

$$\alpha = \frac{(4r)^{2/(\ell-1)}}{(n/4)^{2/(\ell(\ell-1))}} \geq \frac{(4r)^{2/(\ell-1)}}{(4r)^{(\ell+1)/(\ell(\ell-1))}} = (4r)^{(2-\frac{\ell+1}{\ell})\cdot\frac{1}{\ell-1}} = (4r)^{\frac{1}{\ell}}$$

Consequently,

$$b_\ell = \left\lceil \frac{r}{\alpha^{\ell-1}} \right\rceil = \left\lceil \frac{1}{4} \frac{4r}{\alpha^{\ell-1}} \right\rceil \leq \left\lceil \frac{1}{4}(4r)^{1/\ell} \right\rceil \leq (4r)^{1/\ell} \leq \alpha$$

With the bound on b_ℓ, and the fact that $\alpha \geq 1$ in both cases, and that $\ell \leq \lceil \log r \rceil$ we obtain:

$$\sum_{i \in [\ell]} b_i \leq \min(r+1, \lceil 8\alpha \rceil + \lceil 3\log 1/\delta \rceil) + (\ell - 1)\lceil \alpha \rceil$$
$$\leq \min(r, \lceil 3\log 1/\delta \rceil) + (7+\ell)2\alpha$$
$$\leq \min(r, \lceil 3\log 1/\delta \rceil) + 32\log r \left\lceil \frac{(4r)^{2/(\ell-1)}}{(n/4)^{2/(\ell(\ell-1))}} \right\rceil$$

Multiplying this last quantity by $\log(32r)$ gives a space bound.

To obtain a much weaker, but somewhat more comprehensible upper bound on α, we note that:

$$\max_{\lambda \in \mathbb{N}} \log \frac{(4r)^{2/(\lambda-1)}}{(n/4)^{2/(\lambda(\lambda-1))}} \leq \log \max_{\lambda \in \mathbb{R} \cap [2,\infty)} \left(\frac{2}{\lambda - 1} \log(4r) - \frac{2}{\lambda(\lambda-1)} \log(n/4) \right)$$
$$\leq \log 2 \max_{\lambda \in \mathbb{R} \cap [2,\infty)} \left(\frac{2}{\lambda} \log(4r) - \frac{2}{\lambda^2} \log(n/4) \right)$$
$$\leq \log 4 \left[\frac{1}{\lambda} \log(4r) - \frac{1}{\lambda^2} \log(n/4) \right]_{\lambda = \frac{2\log(n/4)}{\log(4r)}}$$
$$= \log 4 \left(\frac{\log(4r)^2}{2\log(n/4)} - \frac{\log(4r)^2}{4\log(n/4)} \right) = \log \frac{2\log(4r)^2}{\log(n/4)}$$

Thus (introducing a multiplicative factor of 2 to account for the $\alpha = 2$ case):

$$\alpha \leq 2(4r)^{\frac{2\log(4r)}{\log(n/4)}}$$

□

A.5 Mechanical proofs for Section 6

Proof of Theorem 1.1. By Lemma 6.7, $|S| \leq \min(n, 2^{z+1})$. We thus have both $|S| \leq n$ and $|S| \leq 2^{z+1} \leq 4^z$. Applying the first upper bound for $|S|$ to the result of Lemma 6.6 gives:

$$z \geq \frac{r}{8460 \log \frac{2n}{r}} \min\left(1, \frac{\log(1/2\delta)}{\log(64n)\log\frac{2n}{r}}\right) \tag{21}$$

Applying the second upper bound for $|S|$ gives:

$$\begin{aligned} z &\geq \frac{r}{8460 \log \frac{2 \cdot 4^z}{r}} \min\left(1, \frac{\log(1/2\delta)}{\log(64 \cdot 4^z)\log\frac{2 \cdot 4^z}{r}}\right) \\ &= \frac{r}{8460 \cdot 3z} \min\left(1, \frac{\log(1/2\delta)}{8z \cdot 3z}\right) \end{aligned}$$

If the left branch of the min holds, then

$$z^2 \geq \frac{r}{25380} \qquad \Longrightarrow \qquad z \geq \sqrt{\frac{r}{25380}}$$

while if the right branch holds:

$$z^4 \geq \frac{r\log(1/2\delta)}{609120} \qquad \Longrightarrow \qquad z \geq \left(\frac{r\log(1/2\delta)}{609120}\right)^{1/4}$$

Thus

$$z \geq \min\left(\sqrt{\frac{r}{25380}}, \left(\frac{r\log(1/2\delta)}{609120}\right)^{1/4}\right)$$

Combining this with Eq. 21 gives,

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

□

Proof of Corollary 1.2. The lower bound from Theorem 1.1 for $\textsc{mif}(n,t)$ with error $\delta = 1/3$, showing constants, is:

$$\max\left(\frac{t\log(3/2)}{8460(\log\frac{2n}{t})^2\log(64n)}, \left(\frac{t\log(3/2)}{609120}\right)^{1/4}\right) \tag{22}$$

If $z \geq (r-1)/2$, then we tautologically have a lower bound of $(r-1)/2$. Otherwise, we have $2z+2 \leq r$. Applying Theorem 6.8 gives, for the left branch of the max in Eq. 22, with $t = \left\lfloor \frac{r}{2z+2} \right\rfloor \geq \frac{1}{8z}$:

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

which implies

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

For the right branch of Eq. 22, we obtain:

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

which implies:

$$z^{5/4} \geq \left(\frac{r}{9745920}\right)^{1/4} \qquad \Longrightarrow z \geq \left(\frac{r}{9745920}\right)^{1/5}$$

Combining the two lower bounds, gives:

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

This lower bound is everywhere smaller than $(r-1)/2$, so it is compatible with the case in which $z \geq (r-1)/2$. □