


Finding Missing Items Requires Strong Forms of Randomness

Amit Chakrabarti 

Department of Computer Science, Dartmouth College, Hanover, NH, USA

Manuel Stoeckl 

Department of Computer Science, Dartmouth College, Hanover, NH, USA

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 as a function of 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 ℓ with elements in $\{1, \dots, n\}$, and $\leq 1/\text{poly}(\ell)$ error, we show that when $\ell = O(2^{\sqrt{\log n}})$, “random seed” adversarially robust algorithms, which only use randomness at initialization, require $\ell^{\Omega(1)}$ bits of space, while “random tape” adversarially robust algorithms, which may make random decisions at any time, may use $O(\text{polylog}(\ell))$ random bits. When ℓ is between $n^{\Omega(1)}$ and $O(\sqrt{n})$, “random tape” adversarially robust algorithms need $\ell^{\Omega(1)}$ space, while “random oracle” adversarially robust algorithms, which can read from a long random string for free, may use $O(\text{polylog}(\ell))$ 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.

2012 ACM Subject Classification Theory of computation \rightarrow Sketching and sampling; Theory of computation \rightarrow Lower bounds and information complexity; Theory of computation \rightarrow Pseudorandomness and derandomization

Keywords and phrases Data streaming, lower bounds, space complexity, adversarial robustness, derandomization, sketching, sampling

Digital Object Identifier 10.4230/LIPIcs.CCC.2024.28

Related Version *Full Version:* <https://arxiv.org/abs/2310.03634> [9]

Extended in Chapter 3 of: <https://digitalcommons.dartmouth.edu/dissertations/229/> [25]

Funding Supported in part by the National Science Foundation under Award 2006589.

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 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 [16]. 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



© Amit Chakrabarti and Manuel Stoeckl;
licensed under Creative Commons License CC-BY 4.0
39th Computational Complexity Conference (CCC 2024).

Editor: Rahul Santhanam; Article No. 28; pp. 28:1–28:20
Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



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, [16] 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 [21] 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.

Some recent works have studied streaming algorithms in a setting where the input to the algorithm can be adaptively (and adversarially) chosen based on its past outputs. 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 [6], sketch switching using independent sub-instances of an underlying classic algorithm [5], rounding outputs to limit the number of computation paths [5], and differential privacy to safely aggregate classic algorithm sub-instances [15]. Mostly, 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) [8, 2], and one for the MISSINGITEMFINDING problem [24], 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 $\text{MIF}(n, \ell)$, the input is a stream (e_1, \dots, e_ℓ) of ℓ integers, not necessarily distinct, with each $e_i \in \{1, \dots, n\}$, where $1 \leq \ell \leq n$. The goal is as follows: having received the i th integer, output a number v in $\{1, \dots, n\} \setminus \{e_1, \dots, e_i\}$. We will be mostly interested in the setting $\ell = o(n)$, so the “trivial” upper bound on the space complexity of $\text{MIF}(n, \ell)$ is $O(\ell \log n)$, achieved by the deterministic algorithm that simply stores the input stream as is.

1.1 Groundwork for Our Results

To state our results about MIF, we need to introduce some key terminology. Notice that MIF is a *tracking problem*: an output is required after reading each input. Thus, we view streaming algorithms as generalizations of finite state (Moore-type) machines. An algorithm \mathcal{A} has a finite set of states Σ (leading to a space cost of $\log_2 |\Sigma|$), a finite input set \mathcal{I} , and a finite output set \mathcal{O} . It has a transition function $T: \Sigma \times \mathcal{I} \times \mathcal{R} \rightarrow \Sigma$ indicating the state to switch to after receiving an input, plus an output function $\gamma: \Sigma \times \mathcal{R} \rightarrow \mathcal{O}$ indicating the output produced upon reaching a state. How the final parameter (in \mathcal{R}) of T and γ is used depends on the type of randomness. We consider four cases, leading to four different models of streaming computation.

- *Deterministic.* The initial state of the algorithm is a fixed element of Σ , and T and γ are deterministic (they do not depend on the parameter in \mathcal{R}).
- *Random seed.* The initial state is drawn from a distribution \mathcal{D} over Σ , and T and γ are deterministic. This models the situation that all random bits used count towards the algorithm's space cost.
- *Random tape.* The initial state is drawn from a distribution \mathcal{D} over Σ . The space \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 $T(e, \sigma, \rho)$. However, γ is deterministic. This models the situation that the algorithm can make random decisions at any time, but it cannot remember past random decisions without recording them (which would add to its space cost).
- *Random oracle.* The initial state is fixed; \mathcal{R} is a sample space. A specific $R \in \mathcal{R}$ is drawn at the start of the algorithm and stays the same over its lifetime. When the algorithm is at state σ and receives input e , its next state is $T(e, \sigma, R)$. The output given at state σ is $\gamma(\sigma, R)$. This models the situation that random bits are essentially “free” to the algorithm; it can read from a long random string which doesn't count toward its space cost and which remains consistent over its lifetime. A random oracle algorithm can be interpreted as choosing a random deterministic algorithm, indexed by R , from some family.

These models form a rough hierarchy; they have been presented in (almost) increasing order of power. 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. Every z -bit random tape algorithm has a corresponding $(z + \log \ell)$ -bit random oracle algorithm – the added space cost is because for a random oracle algorithm to emulate a random tape algorithm, it must have a way to get “fresh” randomness on each turn.¹

Streaming algorithms are also classified by the kind of correctness guarantee they provide. Recall that we focus on “tracking” algorithms [5]; they present an output after reading each input item and this *entire sequence* of outputs must be correct. Here are three possible meanings of the statement “algorithm \mathcal{A} is δ -error” (we assume that \mathcal{A} handles streams of length ℓ with elements in \mathcal{I} and has outputs in \mathcal{O}):

- *Static setting.* For all inputs $\tau \in \mathcal{I}^\ell$, running \mathcal{A} on τ produces incorrect output with probability $\leq \delta$.

¹ 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 position in the stream for free; both are reasonable assumptions in practice.

- *Adversarial setting.* For all (computationally unbounded) adaptive adversaries α (i.e., for all functions $\alpha: \mathcal{O}^* \rightarrow \mathcal{I}$),² running \mathcal{A} against α will produce incorrect output with probability $\leq \delta$.
- *Pseudo-deterministic setting.* There exists a canonical output function $f: \mathcal{I}^* \rightarrow \mathcal{O}$ producing all correct outputs so that, for each $\tau \in \mathcal{I}^\ell$, $\mathcal{A}(\tau)$ fails to output $f(\tau)$ 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 [21], 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 $O(\log \ell + \log \log |\mathcal{I}| + \log \frac{1}{\varepsilon \delta})$ bits of space. However, the resulting algorithm is non-constructive.

1.2 Our Results

As context for our results, we remind the reader that it’s trivial to solve $\text{MIF}(n, \ell)$ in $O(\ell \log n)$ space deterministically (somewhat better deterministic bounds were obtained in [24]). Moving to randomized algorithms, [24] gave a space bound of $O(\log^2 n)$ for $\ell \leq n/2$ in the static setting, and a bound of $\tilde{O}(\ell^2/n + 1)$ ³ in the adversarial setting, using a random *oracle*. The immediate takeaway is that, given access to a deep pool of randomness (i.e., an oracle), MIF becomes easy in the static setting for essentially the full range of stream lengths ℓ and remains easy even against an adversary for lengths $\ell \leq \sqrt{n}$.

The main results of this paper consist of three new lower bounds and one new upper bound on the space complexity of $\text{MIF}(n, \ell)$. Stating the bounds in their strongest forms leads to complicated expressions; therefore, we first present some easier-to-read takeaways from these bounds that carry important conceptual messages. In the lower bounds below, the error level should be thought of as $\delta = 1/n^2$.

► **Result 1.** *At $\ell = \sqrt{n}$, adversarially robust random tape algorithms for $\text{MIF}(n, \ell)$ require $\Omega(\ell^{1/4})$ bits of space. More generally, for every constant $\alpha \in (0, 1)$, there is a constant $\beta \in (0, 1)$ such that at $\ell = \Omega(n^\alpha)$, the space requirement is $\Omega(\ell^\beta)$, in the adversarially robust random tape setting.*

This shows that MIF remains hard, even for modest values of ℓ , if we must be robust while using only a random *tape*, i.e., if there is a cost to storing random bits we want to reuse – a very reasonable requirement for a practical algorithm. The above result is an exponential separation between the random tape and random oracle models.

The random *seed* model places an even greater restriction on an algorithm: besides counting towards storage cost, random bits are available only at initialization and not on the fly. Many actual randomized algorithms, including streaming ones, are structured this way, making it a natural model to study. We obtain the following result.

► **Result 2.** *Adversarially robust random seed algorithms for $\text{MIF}(n, \ell)$ require $\tilde{\Omega}(\sqrt{\ell})$ bits of space.*

² By the minimax theorem, it suffices to consider deterministic adversaries.

³ The notations $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ hide factors polylogarithmic in n and ℓ .

■ **Table 1** Bounds for the space complexity of $\text{MIF}(n, \ell)$, from this and prior work. To keep expressions simple, these bounds are evaluated at error level $\delta = 1/n^2$, when applicable. (†) indicates that the precise results are stronger.

Setting	Type	Bound	Reference
Static	Random seed	$O((\log n)^2)$ if $\ell \leq n/2$	[24]
Adversarial	Random oracle	$O((\frac{\ell^2}{n} + \log n) \log n)$	[24]
		$\Omega(\frac{\ell^2}{n})$	[24]
Adversarial	Random tape	$O(\ell^{\log_n \ell} (\log \ell)^2 + \log \ell \cdot \log n)$ †	Theorem 1
		$\Omega(\frac{\log \ell}{\log n} \ell^{\frac{15}{32} \log_n \ell})$ †	Theorem 8
Adversarial	Random seed	$O((\frac{\ell^2}{n} + \sqrt{\ell} + \log n) \log n)$	[24] ^{a)}
		$\Omega(\frac{\ell^2}{n} + \sqrt{\frac{\ell}{(\log n)^3}} + \ell^{1/5})$	Theorem 10
Pseudo-deterministic	Random oracle	$\Omega(\frac{\ell}{(\log(2n/\ell))^2} + (\ell \log n)^{1/4})$	Theorem 16
Static	Deterministic	$\Omega(\frac{\ell}{\log(2n/\ell)} + \sqrt{\ell})$	[24]
		$O(\frac{\ell \log \ell}{\log n} + \sqrt{\ell \log \ell})$	[24]

a) The random seed algorithm for the adversarial setting is given in the arXiv version of [24].

Consider the two results above as ℓ decreases from \sqrt{n} to $\Theta(1)$. The bound in Result 2 stays interesting even when $\ell = n^{o(1)}$, so long as $\ell \geq (\log n)^C$ for a suitable constant C (in fact, the full version of the result is good for even smaller ℓ). In contrast, the bound in Result 1 peters out at much larger values of ℓ . There is a very good reason: MIF starts to become “easy,” even under a random-tape restriction, once ℓ decreases to sub-polynomial in n . Specifically, we obtain the following *upper* bound.

► **Result 3.** *There is an adversarially robust random tape algorithm for $\text{MIF}(n, \ell)$ that, in the regime $\ell = O(2^{\sqrt{\log n}})$, uses $O(\log \ell \cdot \log n)$ bits of space.*

Notice that at $\ell = \Theta(2^{(\log n)^{1/C}})$, where $C \geq 2$ is a constant, the bound in Result 3 is polylogarithmic in ℓ . Combined with the lower bound in Result 2, we have another exponential separation, between the random seed and random tape models.

The proof of Result 2 uses a reduction, given in prior work [24], that converts a space lower bound in the pseudo-deterministic setting to a related bound in the random-seed setting. A pseudo-deterministic algorithm is allowed to use randomness (which, due to Newman’s theorem, might as well be of the oracle kind) but must, with high probability, map each input to a *fixed* output, just as a deterministic algorithm would. This strong property makes the algorithm adversarially robust, because the adversary has nothing to learn from observing its outputs. Thanks to the [24] reduction, the main action in the proof of Result 2 is the following new lower bound we give.

► **Result 4.** *Pseudo-deterministic random oracle algorithms for $\text{MIF}(n, \ell)$ require $\tilde{\Omega}(\ell)$ bits of space.*

These separations rule out the possibility of a way to convert an adversarially robust random oracle algorithm to use only a random *seed* or even a random *tape*, with only minor (e.g., a $\text{polylog}(\ell, n)$ factor) overhead. In contrast, as we noted earlier, such a conversion is routine in the static setting, due to Newman’s theorem [21]. The separation between random

oracle and random tape settings shows that MISSINGITEMFINDING is a problem for which much lower space usage is possible if one’s adversaries are computationally bounded (in which case a pseudo-random generator can emulate a random oracle.)

Table 1 shows more detailed versions of the above results as well as salient results from earlier work, summarizing the state of the art for the space complexity of MIF(n, ℓ). The fully detailed versions of our results, showing the dependence of the bounds on the error probability, appear in later sections of the paper, as indicated in the table.

1.3 Related Work

We briefly survey related work. An influential early work [14] considered adaptive adversaries for *linear* sketches. The adversarial setting was formally introduced by [5], 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 obtained slightly better upper bounds by using a random oracle, although later work [26] removed this need. [6] observed that in sampling-based streaming algorithms, increasing the sample size is often all that is needed to make an algorithm adversarially robust. [15] described how to use differential privacy techniques as a more efficient alternative to sketch-switching, and [4] used 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. [18] described a problem (of approximating a certain real-valued function) that requires exponentially more space in the adversarial setting than in the static setting. [8], in a brief comment, observed a similar separation for a simple problem along the lines of MIF. They also proved lower bounds for adversarially robust coloring algorithms for graph edge-insertion streams. [24] considered the MIF problem as defined here and, among upper and lower bounds in a number of models, described an adversarially robust algorithm for MIF that requires a random oracle; they asked whether a random oracle is *necessary* for space-efficient algorithms.

The *white-box* adversarial setting [1] is similar to the adversarial setting we study, with the adversary having the additional power of seeing the internal state of the algorithm, including (if used) the random oracle. [24] proved an $\Omega(\ell/\text{polylog}(n))$ lower bound for MIF(n, ℓ) for random tape algorithms in this setting, suggesting that any more efficient algorithm for MIF must conceal some part of its internal state. Pseudo-deterministic streaming algorithms were introduced by [12], who gave lower bounds for a few problems. [7, 13] gave lower bounds for pseudo-deterministic algorithms that approximately count the number of stream elements. The latter shows they require $\Omega(\log m)$ space, where m is the stream length; in contrast, in the static setting, Morris’s counter algorithm⁴ uses only $O(\log \log m)$ space.

While it is not posed as a streaming task, the *mirror game* introduced by [11] 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). [11] showed that if Alice has $o(n)$ bits of memory and plays a deterministic strategy, Bob can always win. Later, [10, 20] showed that if Alice has access to a random oracle, she can tie-or-win w.h.p. using only

⁴ Morris’s is a “random tape” algorithm; “random seed” algorithms for counting aren’t better than deterministic ones.

$O(\text{polylog}(n))$ space. A major open question here is how much space Alice needs when she does not have a random oracle. [19] did not resolve this, but showed 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, [16] 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 [22] 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 [17] 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 [3], in which one assumes that *all* agents have access to the random oracle. [1], when defining white-box adversaries, also assumed that they can see the same random oracle as the algorithm; and, for one task, obtained 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. [23] showed that logspace Turing machines with a multiple-access random tape can (with zero error) decide languages that logspace Turing machines with a read-once random tape decide only with bounded two-sided error. This type of separation does not hold for *time* complexity classes.

For a more detailed history and survey of problems related to MISSINGITEMFINDING, we direct the reader to [24].

2 Organization of This Extended Abstract

What follows is an extended abstract of our paper, which omits formal proofs of our results. Instead, we give a technical overview of each result, followed by selected details of its proof. The full paper contains all remaining details and formal proofs.

2.1 Notation

Throughout this paper, $\log x = \log_2 x$, while $\ln x = \log_e x$. The set \mathbb{N} consists of all positive integers; $[k] := \{1, 2, \dots, k\}$; and $[a, b)$ is a half open interval of real numbers. For a condition or event E , the symbol $\mathbb{1}_E$ takes the value 1 if E occurs and 0 otherwise. The sequence (stream) obtained by concatenating sequences a and b , in that order, is denoted $a \circ b$. For a set S of elements in a totally ordered universe, $\text{SORT}(S)$ denotes the sequence of elements of S in increasing order; $\binom{S}{k}$ is the set of k -element subsets of S ; and $\text{SEQS}(S, k) = \{\text{SORT}(Y) : Y \in \binom{S}{k}\}$. We sometimes extend set-theoretic notation to vectors and sequences; e.g., for $y \in [n]^t$, write $y \subseteq S$ to mean that $\forall i \in [t] : y_i \in S$. For a set X , $\Delta[X]$ denotes the set of probability distributions over X , while $A \sim X$ indicates that A is chosen uniformly at random from X .

2.2 Preliminary Remarks

The proofs of Results 1, 3, and 4 are all significant generalizations of existing proofs from [24] which handled different (and more tractable) models. The proof of Result 2 consists of applying a reduction from [24] to the lower bound given by Result 4. As we explain our techniques, we will summarize the relevant “basic” proofs from [24], which will clarify the enhancements needed to obtain our results.

Space complexity lower bounds in streaming models are often proved via communication complexity. This meta-technique is unavailable to us, because the setup of communication complexity blurs the distinctions between random seed, random tape, and random oracle models and our results are all about these distinctions. Instead, to prove Result 1, we design a suitable strategy for the stream-generating adversary that exploits the algorithm’s random-tape limitation by learning enough about its internal state. Our adversary uses a nontrivially recursive construction. To properly appreciate it, it is important to understand what streaming-algorithmic techniques the adversary must contend with. Therefore, we shall discuss our *upper* bound result first.

3 Random Tape Upper Bound (Result 3)

The adversarially robust random tape algorithm for $\text{MIF}(n, \ell)$ can be seen as a generalization of the random oracle and random seed algorithms.

The random oracle algorithm and its adversaries. The random oracle algorithm for $\text{MIF}(n, \ell)$ from [24] has the following structure. It interprets its oracle random string as a uniformly random sequence L containing $\ell + 1$ distinct elements in $[n]$. As it reads its input, it keeps track of which elements in L were in the input stream so far (were “covered”). It reports as its output the first uncovered element of L . Because L comes from the oracle, the space cost of the algorithm is just the cost of keeping track of the set J of covered positions in L . We will explain why that can be done using only $O((\ell^2/n + 1) \log \ell)$ space, in expectation.

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

Delaying the echo strategy. If we implemented the above random oracle algorithm as a random seed algorithm, we would need $\Omega(\ell)$ bits of space, just to store the random list L . But why does L need to have length $\ell + 1$? This length is needed for the algorithm to be resilient to the echo strategy, which covers one new element of L on every step; if L were shorter, the echo strategy could entirely cover it, making the algorithm run out of possible values to output. The random seed algorithm for $\text{MIF}(n, \ell)$ works by making the echo strategy less effective, ensuring that multiple inputs are needed for it to cover another element of L . It does this by partitioning $[n]$ into $\Theta(\ell)$ disjoint subsets (“blocks”) of size $\Theta(n/\ell)$, and then taking L to be a random list of blocks (rather than a random list of elements of $[n]$). We will now say that a block is “covered” if *any* element of that block was an input. Instead of outputting the first uncovered element in L , the algorithm will run a deterministic algorithm for MIF *inside* the block corresponding to the first uncovered block of L , and report outputs from that; and will only move on to the next uncovered block when the nested algorithm

stops. See Algorithm 1 for the details of this design. Because the analogue of the echo strategy now requires many more inputs to cover a block, we can make the list L shorter. This change will not make the random strategy much more effective.⁵ The minimum length of L is constrained by the $O(n/\ell)$ block sizes, which limit the number of outputs that the nested algorithm can make; as a result, one must have $L = \Omega(\ell^2/n)$. In the end, after balancing the length of the list with the cost of the nested algorithm, the optimal list length for the random seed algorithm will be $O(\ell^2/n + \sqrt{\ell})$.

■ **Algorithm 1** Example: recursive construction for a random tape $\text{MIF}(n, \ell)$ algorithm.

Parameter: $t \in [\Omega(\ell^2/n), \ell]$ is the number of parts into which the input stream is split

Initialization:

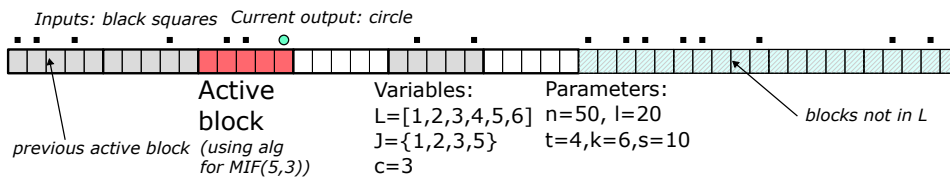
- 1: Let $k = O(t)$, $s = O(\ell)$, and B_1, \dots, B_s be a partition of $[n]$ into s equal “blocks”
- 2: $L \leftarrow$ uniformly randomly chosen sequence of k distinct elements of $[s]$
- 3: $J \leftarrow \emptyset$, is a subset of $[k]$ \triangleright a set marking which blocks of L have been covered
- 4: $c \leftarrow 1$ \triangleright the current active block
- 5: $A \leftarrow$ instance of algorithm \mathcal{A} solving $\text{MIF}(n/s, \lceil \ell/t \rceil)$

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

- 6: Let h be the block containing a , and x the rank of a in B_h
- 7: **if** $h \in L$ **then**
- 8: Add j to J , where $L_j = h$ \triangleright Mark list element containing h as used
- 9: **if** $h = L_c$ **then**
- 10: $A.\text{UPDATE}(x)$
- 11: **if** A is out of space **then** \triangleright This requires that $A.\text{UPDATE}()$ be called $\geq \lceil \ell/t \rceil$ times
- 12: $c \leftarrow$ least integer which is $> c$ and not in J \triangleright This line may abort if $J = [k]$
- 13: $A \leftarrow$ new instance of algorithm \mathcal{A} \triangleright Using new random bits, if \mathcal{A} is randomized

Output $\rightarrow [n]$:

- 14: Let $x \in [n/s]$ be the output of A
- 15: **return** x th entry of block B_c



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

The recursive random tape algorithm. The random seed algorithm for $\text{MIF}(n, \ell)$ used the construction of Algorithm 1 to build on top of an “inner” deterministic algorithm.⁶ To get an efficient random tape algorithm, we can recursively apply the construction of Algorithm 1

⁵ The fact that $[n]$ is split into $\Omega(\ell)$ blocks is enough to mitigate the random strategy; with ℓ guesses, the adversary is unlikely to guess more than a constant fraction of the elements in L .

⁶ The construction uses randomness in two places: when initializing the random sequence L , and (possibly) each time the inner algorithm is initialized. For the random seed model, every “inner” initialization

28:10 Finding Missing Items Requires Strong Forms of Randomness

$d - 1$ times, for $d = O(\min(\log \ell, \log n / \log \ell))$; at the end of this recursion, we can use a simple deterministic algorithm for MIF. The optimal lengths of the random lists used at each level of the recursion are determined by balancing the costs of the different recursion levels. We end up choosing list lengths that all bounded by a quantity which lies between $O(\ell^{1/d})$ and $O(\ell^{1/(d-1)})$.

In the extreme case where $d = \Theta(\log \ell)$ and the required error level δ is constant, our recursive algorithm may have a stack of random lists, each of length 2, and every time a level of the algorithm completes (i.e., all blocks of a list have been used), it will make a new instance of that level. That is, some large uncovered block will be split into many smaller blocks, and the algorithm will randomly pick two of them for the new instance's list. Because the lists are all short, the algorithm will not need to remember many random bits at a given time; in exchange, for this regime it needs a very large ($n = \ell^{\Omega(d)}$) number of possible outputs and will frequently need to sample new random lists.

The final version of our algorithm is given in the full version of the paper. It looks somewhat different from the recursive construction in Algorithm 1, because we have unraveled the recursive framing to allow for a simpler error analysis that must only bound the probability of a single “bad event.” The resulting space bound is:

► **Theorem 1.** *There is a family of adversarially robust random tape algorithms, where for $\text{MIF}(n, \ell)$ the corresponding algorithm has $\leq \delta$ error and uses*

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

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

4 Random Tape Lower Bound (Result 1)

The AVOID problem. At the core of many of the MIF lower bounds is the SUBSET AVOIDANCE communication problem, introduced in [8]. Here we have two players, Alice and Bob, and a known universe $[m]$: Alice has a set $A \subseteq [m]$ of size a , and should send a message (as short as possible) to Bob, who should use the message to output a set $B \subseteq [m]$ of size b which is disjoint from A . Henceforth, we'll call this problem $\text{AVOID}(m, a, b)$. [8] showed that both deterministic and constant-error randomized one-way protocols for this problem require $\Omega(ab/m)$ bits of communication. An adversarially robust z -space algorithm for $\text{MIF}(m, a + b)$ can be used as a subroutine to implement a z -bit one-way protocol for $\text{AVOID}(m, a, b)$, thereby proving $z = \Omega(ab/m)$. This immediately gives us an $\Omega(\ell^2/n)$ space lower bound for $\text{MIF}(n, \ell)$, which, as we have seen, is near-optimal in the robust, random oracle setting.

► **Lemma 2** (Adversarially robust random oracle lower bound, from [24]). *Any random oracle (or random seed) algorithm which solves $\text{MIF}(n, \ell)$ in the adversarial setting with total error $\leq \delta$ requires $\Omega(\ell^2/n + \log(1 - \delta))$ bits of space.*

The random tape lower bound. To prove stronger lower bounds that exploit the random tape limitation of the algorithm, we need a more sophisticated use of AVOID. Fix an adversarially robust, random tape, z -space algorithm \mathcal{A} for $\text{MIF}(n, \ell)$. Roughly speaking,

would require a corresponding set of random bits, which are counted toward the space cost of the algorithm. Using a deterministic inner algorithm avoids this cost.

while the random oracle argument used \mathcal{A} to produce an AVOID protocol at the particular scale $a = b = \ell$, for the fixed universe $[n]$, our random tape argument will “probe” \mathcal{A} in a recursive fashion – reminiscent of the recursion in our random tape upper bound – to identify a suitable scale and sub-universe at which an AVOID protocol can be produced. This probing will itself invoke the AVOID lower bound to say that if an AVOID(m, a, b) protocol is built out of a z -space streaming algorithm where $z \ll a$, then B must be small, with size $b = O((z/a)m)$.

We will focus on the regime where $\delta = O(1/n)$. This error level requires a measure of structure from the algorithm: it cannot just pick a random output each step, because that would risk colliding with an earlier input with $\geq 1/n$ probability. Our recursive argument works by writing z , the space usage of \mathcal{A} , as a function of a space lower bound for MIF(w, t), where $w = \Theta(zn/\ell)$ and $t = \Theta(n/z)$. For small enough z , $t^2/w \gg \ell^2/n$, so by repeating this reduction step a few times we can increase the ratio of the stream length to the input domain size until we can apply the simple $\Omega(\hat{\ell}^2/\hat{n})$ lower bound for MIF($\hat{n}, \hat{\ell}$). With the right number of reduction steps, one obtains the lower bound formula of Theorem 8, of which Result 1 is a special case.

The reduction. The reduction step argues that the MIF(n, ℓ) algorithm \mathcal{A} “contains” a z -space algorithm for MIF(w, t), which, on being given any $t = O(\ell/z)$ items in a certain sub-universe $W \subseteq [n]$ of size $w = O(zn/\ell)$, will repeatedly produce missing items *from that sub-universe*. That such a set W *exists* can be seen as a consequence of the lower bound for AVOID: if \mathcal{A} receives a random sorted subset S of $\ell/2$ elements in $[n]$, then because there are $\binom{n}{\ell/2}$ possible subsets, most of the 2^z states of \mathcal{A} will need to be “good” for $\Omega(2^{-z} \binom{n}{\ell/2})$ different subsets. In particular, upon reaching a given state σ , for \mathcal{A} to solve MIF with error probability $O(1/n)$, its outputs henceforth – for the next $\ell/2$ items in the stream – must avoid most of the sets of inputs that could have led it to σ . We will prove by a counting argument that after the random sequence S is sent, each state σ has an associated set H_σ of possible “safe” outputs which are unlikely to collide with the inputs from S , and that $|H_\sigma|$ is typically $O(zn/\ell)$. Thus, for a typical state σ , starting \mathcal{A} from σ causes its next $\ell/2$ outputs to be inside H_σ , w.h.p.; in other words, \mathcal{A} contains a “sub-algorithm” solving MIF($O(zn/\ell), \ell/2$) on the set $W = H_\sigma$.

However, even though there exists a set W on which \mathcal{A} will concentrate its outputs, it may not be possible for an adversary to find it. In particular, had \mathcal{A} been a random *oracle* algorithm, each setting of the random string might lead to a different value for W , making W practically unguessable. But \mathcal{A} is in fact a random *tape* algorithm, so we can execute the following strategy.

In our core lemma, Lemma 5, we design an adversary (Adversary 2) that can with $\Omega(1)$ probability identify a set W of size $\Theta(zn/\ell)$ for which the next $\Theta(\ell/z)$ outputs of \mathcal{A} will be contained in W , with $\Omega(1)$ probability, no matter what inputs the adversary sends next. In other words, our adversary will identify a part of the stream and a sub-universe of $[n]$ where the algorithm solves MIF($\Theta(zn/\ell), \Theta(\ell/z)$). The general strategy is to use an iterative search based on a win-win argument. First, the adversary will send a stream comprising a random subset S of size $\ell/2$ to \mathcal{A} , to ensure that henceforth its outputs are contained in some (unknown) set H_ρ , where ρ is the (unknown) state reached by \mathcal{A} just after processing S . Because \mathcal{A} has $\leq 2^z$ states, from the adversary’s perspective there are $\leq 2^z$ possible candidates for H_ρ . Then, the adversary conceptually divides the rest of the stream to be fed to \mathcal{A} into $O(z)$ phases, each consisting of $t = O(\ell/z)$ stream items. In each phase, one of the following things happens.

1. There exists a “sub-adversary” (function to choose the t items constituting the phase, one by one) which will probably make \mathcal{A} output an item that rules out a constant fraction of the candidate values for H_ρ (output i rules out set J if $i \notin J$). The adversary then runs this sub-adversary.
2. No matter how the adversary picks the t inputs for this phase, there will be a set W (roughly, an “average” of the remaining candidate sets) that probably contains the corresponding t outputs of \mathcal{A} .

As the set of candidate sets can only shrink by a constant fraction $O(z)$ times, the first case can only happen $O(z)$ times, with high probability. Thus, eventually, the adversary will identify the set W that it seeks. Once it has done so, it will run the optimal adversary for $\text{MIF}(\Theta(zn/\ell), \Theta(\ell/z))$. This essentially reduces the lower bound for $\text{MIF}(\ell, n)$ to that for $\text{MIF}(\Theta(zn/\ell), \Theta(\ell/z))$.

4.1 Technical Details

Types of error. One subtlety is that we will need to carefully account for the probability that \mathcal{A} , over the next $\Theta(\ell/z)$ stream items, produces outputs outside W . This will require us to distinguish between two types of “errors” for the algorithm over those next $\Theta(\ell/z)$ items: an $O(1)$ chance of producing an output outside W , and a smaller chance of making a mistake per the definition of MIF, i.e., outputting an item that was not missing (cf. Definition 3).

► **Definition 3.** *An algorithm \mathcal{A} for $\text{MIF}(n, \ell)$ can fail in either of two ways. It may make an incorrect output, or mistake, if outputs an element that was already in the input stream. It may also abort, by outputting a special value \perp (or some other value which is not a possible input for MIF).*

This distinction is useful because, if we take an algorithm for $\text{MIF}(n, \ell)$, conditioned on producing some initial transcript of outputs in response to an input sequence, we may obtain an algorithm for $\text{MIF}(|W|, t)$ for some $t \leq \ell$ and $W \subseteq [n]$; the probability that the algorithm “aborts” (produces an output outside of W) can be much larger than the probability that the algorithm makes an incorrect output (output in W that collides with an earlier input). In the following proofs the algorithm aborting will be bad for the adversary, and making a mistake will be good.

For integers n, ℓ, z with $1 \leq \ell < n$, and $\gamma \in [0, 1]$, let $\text{Algs}(n, \ell, z, \gamma)$ be the set of all z -bit *random tape* algorithms for $\text{MIF}(n, \ell)$ which on *any* adversary abort with probability $\leq \gamma$. Define $\Delta(n, \ell, \gamma, z) := \min\{\delta(\mathcal{A}, n, \ell) : \mathcal{A} \in \text{Algs}(n, \ell, z, \gamma)\}$, where $\delta(\mathcal{A}, n, \ell)$ is the maximum probability, over all possible adversaries, that \mathcal{A} makes an incorrect output. As a consequence of the definition, $\Delta(n, \ell, \gamma, z)$ is non-increasing in γ and z .

Using this new notation, the $\Omega(\ell^2/n)$ lower bound for adversarially robust streaming algorithms from [24] (cf. Lemma 2) tells us:

► **Lemma 4.** *Random tape algorithms for $\text{MIF}(n, \ell)$ that do not abort often have high error if they use too little space: concretely,*

$$\Delta(n, \ell, \gamma, z) \geq \frac{1}{4} \mathbb{1}_{z \leq \ell^2 / (16n \ln 2)} \mathbb{1}_{\gamma \leq 1/2}. \quad (1)$$

Induction lemma. Our proof of Result 1 is inductive, with the above lemma being the base case. The induction step consists of a reduction, using an adaptive adversary described in Adversary 2 to prove a lower bound on the mistake probability. The next lemma formalizes the induction step.

► **Lemma 5.** Let $1 \leq \ell < n$ and z be integers, and $\gamma \in [0, \frac{1}{2}]$. Let k be an integer parameter for which $z \geq 2 \log(32k)$. Define, matching definitions in Adversary 2,

$$w = 2 \left\lfloor 32 \frac{zn}{\ell} \right\rfloor \quad \text{and} \quad t = \left\lfloor \frac{\ell}{64zk} \right\rfloor.$$

If $t < w$, then there is a distribution $\mu \in \Delta[0, 1]$ for which $\mathbb{E}_{G \sim \mu} G \leq \gamma + \frac{1}{4k}$ and

$$\Delta(n, \ell, \gamma, z) \geq \min \left(\frac{\ell}{2^7 nk}, \left(\frac{1}{2} - \frac{1}{4k} \right) \mathbb{E}_{G \sim \mu} \Delta(w, t, G, z) \right). \quad (2)$$

The adversary. The adversary (Adversary 2) used for Lemma 5 is rather complicated, and requires some additional definitions.

► **Definition 6.** Say \mathcal{A} is a random tape algorithm whose states are given by the set Σ , and Q is a subset of Σ , where each state in Q 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 sequences in $[n]^*$ of length between 0 and $t - 1$, inclusive, to values in $[n]$.) For any state $\sigma \in \Sigma$ of \mathcal{A} , let $G(\sigma, \Upsilon)$ be the random variable in $[n]^t$ which gives the output if we run \mathcal{A} , starting at state σ , against the adversary Υ . (If after processing a few inputs, the algorithm has output sequence $v \in [n]^*$, its next input will be $\Upsilon(v)$.) We define an adversary to be α -splitting for Q against a distribution $\mathcal{D} \in \Delta[\Sigma]$ if, when we choose a random state S from \mathcal{D} ,

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

When we run Adversary 2 against an algorithm \mathcal{A} , let ρ be the state of \mathcal{A} after v is sent. The proof of Lemma 5 is long and requires that we consider the probabilities of the following events:

- B_{REPEAT} occurs if \mathcal{A} produces an output in $[n] \setminus H_\rho$
- B_{BIG} occurs if the state ρ has $|H_\rho| > \frac{1}{2}w$
- $B_{\text{INCOMPLETE}}$ occurs if the adversary aborts without executing Line 18
- B_{ABORT} occurs if \mathcal{A} aborts *before* the adversary reaches Line 18
- R_{ABORT} occurs if \mathcal{A} “aborts” (either for real, or by making an output outside W') *while* the adversary is executing Line 18
- R_{ERROR} occurs if \mathcal{A} produces an incorrect output *while* the adversary is executing Line 18

Calculations. By repeatedly applying Lemma 5, we obtain the following:

► **Lemma 7.** Let $1 \leq \ell < n$. For any integer $k \geq 1$, say that z is an integer satisfying $z \leq \frac{1}{64k} \ell^{1/k}$. Then:

$$\Delta(n, \ell, 0, z) > \min \left(\frac{\ell}{2^{10} nk}, \frac{1}{2^{k+5}} \mathbb{1}_{z \leq L} \right) \quad \text{where} \quad L = \frac{1}{64k} \left(\frac{\ell^{k+1}}{n} \right)^{\frac{2}{k^2+3k-2}}. \quad (3)$$

Consequently, algorithms for MIF with $\leq \min(\frac{\ell}{2^{10} nk}, 2^{-(k+5)})$ error require $> L$ bits of space.

Lemma 7 implies a lower bound on z for z -bit algorithms with $< \frac{\ell}{2^{10} nk}$ error probability. Choosing the value of k which maximizes the lower bound on z , and doing some additional calculations, gives the following theorem:

⁶ For any sequence $v \in \text{SEQS}([n], \lceil \ell/2 \rceil)$, $P(v)(\sigma) = \Pr[\text{the state of } \mathcal{A} \text{ just after receiving } v \text{ is } \sigma]$

28:14 Finding Missing Items Requires Strong Forms of Randomness

■ **Adversary 2** An adversary for a random tape MIF(n, ℓ) algorithm, with parameter k .

Let: $w = 2 \lfloor 32 \frac{zn}{\ell} \rfloor$, $h_{\max} = 32zk$, and $t = \lfloor \frac{\ell}{2h_{\max}} \rfloor$

ADVERSARY

- 1: $v \leftarrow$ a uniformly random sequence in $\text{SEQS}([n], \lceil \ell/2 \rceil)$.
- 2: **send** v to the algorithm
- 3: Let \mathcal{G} be a distribution over functions of type $\text{SEQS}([n], \lceil \ell/2 \rceil) \rightarrow \Sigma$, so that when $F \sim \mathcal{G}$, the distribution of $F(v)$ equals the distribution of current algorithm states
- 4: Compute, for all $\sigma \in \Sigma$,

$$H_\sigma := \left\{ i \in [n] : \Pr_{X \sim \text{SEQS}([n], \lceil \ell/2 \rceil), F \sim \mathcal{G}} [i \in X \mid F(X) = \sigma] \leq \frac{\lceil \ell/2 \rceil}{4n} \right\}$$

- 5: Let $Q_0 = \{\sigma \in \Sigma : |H_\sigma| \leq \frac{1}{2}w\}$ ▷ *Have a $\geq 1 - 1/16k$ chance current alg. state is in Q_0*
- 6: **for** h in $1, \dots, h_{\max}$ **do**
- 7: Let \mathcal{D} be the distribution over alg. states conditioned on the transcript so far
- 8: **if** \exists a $1/(8k)$ -splitting t -length deterministic adversary Υ for Q_{h-1} given \mathcal{D} **then**
- 9: **run** Υ against the algorithm, and let $y \in [n]^t$ be the output
- 10: $Q_h \leftarrow \{\sigma \in Q_{h-1} : y \subseteq H_\sigma\}$ ▷ *Have a $\geq 1/(8k)$ chance that $|Q_h| \leq \frac{1}{2}|Q_{h-1}|$*
- 11: **if** $Q_h = \emptyset$ **then abort**
- 12: **else**
- 13: $W \leftarrow \{i \in [n] : |\{\sigma \in Q_{h-1} : i \in H_\sigma\}| \geq \frac{1}{2}|Q_{h-1}|\}$.
- 14: Let $W' \leftarrow W$ plus $w - |W|$ padding elements
- 15: Define sub-algorithm \mathcal{B} to behave like the given algorithm, conditioned on the exact transcript of inputs and outputs made so far
- 16: Let Ξ be an adversary maximizing the probability that \mathcal{B} makes an incorrect output. (This can be computed using brute-force search.)
- 17: ▷ *If \mathcal{B} produces an output outside of W' , we interpret this as \mathcal{B} having aborted, not as having made a mistake*
- 18: **run** adversary Ξ , sending t inputs in W'
- 19: **return**
- 20: **abort**

► **Theorem 8.** *Random tape δ -error adversarially robust algorithms for MIF(n, ℓ) require*

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

bits of space, for $\delta \leq \frac{1}{2^{10n}}$.

► **Remark.** The adversary of Adversary 2 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 MIF(n, ℓ) from [24] using a pseudo-random generator that fools all polynomial-time adversaries. One can also prove by minimax theorem that universal adversaries for (random tape or otherwise) MIF(n, ℓ) algorithms can not be used to prove any stronger lower bounds than the one for random oracle algorithms.

5 Random Seed Lower Bound (Result 2)

The adversary constructed above for our random tape lower bound can be seen as a significant generalization of the adversary used by [24] to prove a random seed lower bound conditioned on a (then conjectured) pseudo-deterministic lower bound. Indeed, [24]’s adversary against a z -space algorithm \mathcal{A} also proceeds in a number of phases, each of length $t = \Theta(\ell/z)$. In each step, either (1) it can learn some new information about the initial state of \mathcal{A} (the “random seed”), by sending \mathcal{A} a specific stream of inputs in $[n]^t$, looking at the resulting output, and ruling out the seed values that could not have produced the output; or (2) it cannot learn much information, because for any possible input stream in $[n]^t$, \mathcal{A} has an output that it produces with constant probability. Each time the adversary follows the case (1), a constant fraction of the $\leq 2^z$ seed values are ruled out. Therefore, either within $O(z)$ steps the adversary will exactly learn the seed, at which point it can perfectly predict \mathcal{A} ’s behavior, which lands us in case (2); or \mathcal{A} will not reveal much information about the seed in a given phase, which also puts us in case (2). Because case (2) means that \mathcal{A} behaves pseudo-deterministically, \mathcal{A} must use enough space to pseudo-deterministically solve $\text{MIF}(n, t)$.

► **Theorem 9** (from [24]). *Let $S_{1/3}^{PD}(n, \ell)$ give a space lower bound for a pseudo-deterministic algorithm for $\text{MIF}(n, \ell)$ with error $\leq 1/3$. Then an adversarially robust random seed algorithm with error $\delta \leq \frac{1}{6}$, if it uses z bits of space, must have $z \geq S_{1/3}^{PD}(n, \lfloor \frac{\ell}{2z+2} \rfloor)$.*

Thus, Result 2 follows as a corollary of Result 4, which we discuss next. More specifically, Theorem 10 follows by combining Theorem 9 with the pseudo-deterministic lower bound, and also applying Lemma 2, which is stronger in the regime $\ell \geq n^{2/3}$.

► **Theorem 10.** *Adversarially robust random seed algorithms for $\text{MIF}(n, \ell)$ with error $\leq \frac{1}{6}$ require space:*

$$\Omega\left(\frac{\ell^2}{n} + \sqrt{\ell/(\log n)^3} + \ell^{1/5}\right).$$

6 Pseudo-Deterministic Lower Bound (Result 4)

This proof generalizes [24]’s space lower bound for *deterministic* $\text{MIF}(n, \ell)$ algorithms, which we briefly explain. Fix a deterministic $\text{MIF}(n, \ell)$ algorithm \mathcal{A} that uses z bits of space. For each stream τ with length $|\tau| \leq \ell$, define F_τ to be the set of *all possible outputs* of \mathcal{A} corresponding to length- ℓ streams that have τ as a prefix. Let ρ be a stream such that $|\tau| + |\rho| \leq \ell$. Then, by definition, $F_{\tau \circ \rho} \subseteq F_\tau$; whereas, by the correctness of \mathcal{A} , $F_{\tau \circ \rho} \cap \rho = \emptyset$. Now consider the AVOID problem over the universe F_τ , for a fixed τ : if Alice gets $\rho \subseteq F_\tau$ as an input, she could send Bob the state σ of \mathcal{A} upon processing $\tau \circ \rho$, whereupon Bob could determine $F_{\tau \circ \rho}$ (by repeatedly running \mathcal{A} ’s state machine starting at σ), which would be a valid output.

Let us restrict this scenario to suffixes ρ of some fixed length t ; we’ll soon determine a useful value for t . By the above observations, were it the case that

$$\exists \tau \in [n]^{\leq \ell-t} \forall \rho \in [n]^t: |F_{\tau \circ \rho}| \geq \frac{1}{2}|F_\tau|, \quad (4)$$

we would have a z -bit protocol for AVOID($|F_\tau|, t, \frac{1}{2}|F_\tau|$). By [8]’s lower bound for AVOID, we would have $z \geq Ct$ for a universal constant C . On the other hand, if the opposite were true, i.e.,

28:16 Finding Missing Items Requires Strong Forms of Randomness

$$\forall \tau \in [n]^{\leq \ell-t} \exists \rho \in [n]^t: |F_{\tau \circ \rho}| < \frac{1}{2}|F_{\tau}|, \quad (5)$$

then, starting from the empty stream ϵ , we could add a sequence of length- ℓ suffixes ρ_1, \dots, ρ_d (where $d \leq \lfloor \ell/t \rfloor$) such that $|F_{\rho_1 \circ \dots \circ \rho_s}| < 2^{-d}|F_{\epsilon}| \leq 2^{-d}n$. Since \mathcal{A} must produce *some* output at time ℓ , this would be a contradiction for $d \geq \log n$. Thus, for a setting of $t = \Theta(\ell/\log n)$, situation (4) must occur, implying a lower bound of $z = \Omega(\ell/\log n)$.

Relaxing “all outputs” to “common outputs”. Examining the above argument closely shows where it fails for pseudo-deterministic algorithms. In constructing an AVOID protocol above, we needed the key property that F_{τ} can be determined from just the *state* of \mathcal{A} upon processing τ . For pseudo-deterministic algorithms, if we simply define F'_{τ} to be “the set of all *canonical* outputs at time ℓ for continuations of τ ,” we cannot carry out the above proof plan because this F'_{τ} cannot be computed reliably from a single state: given a random state σ associated to τ , on average a δ fraction of the outputs might be incorrect and have arbitrary values; even a single bad output could corrupt the union calculation!

To work around this issue, we replace F_{τ} with a more elaborate recursive procedure FINDCOMMONOUTPUTS, (or FCO for short) that computes the “most common outputs” at time ℓ for a certain distribution over continuations of τ . To explain this, let us imagine positions 1 through ℓ in the input stream as being divided into d contiguous “time intervals.” In the deterministic proof, these intervals were of length t each. Given a stream τ that occupies the first $d - k$ of these intervals, F_{τ} can be thought of as the output of a procedure FINDALLOUTPUTS (or FAO for short) where $\text{FAO}(\mathcal{A}, \tau, k)$ operates as follows: for each setting ρ of the $(d - k + 1)$ th time interval, call $\text{FAO}(\mathcal{A}, \tau \circ \rho, k - 1)$ and return the union of the sets so obtained. In the base case, $\text{FAO}(\mathcal{A}, \tau, 0)$ takes a stream $\tau \in [n]^{\ell}$ and returns the singleton set $\{\mathcal{A}(\tau)\}$. The deterministic argument amounts to showing that, with interval lengths $t = \Theta(z)$, the set $\text{FAO}(\mathcal{A}, \tau, k)$ has cardinality $\geq 2^k$; since $\text{FAO}(\mathcal{A}, \epsilon, d)$ has cardinality $\leq n$, this bounds $d \leq \log n$, which lower-bounds z .

For our pseudo-deterministic setting, we use time intervals as above and we design an analogous procedure $\text{FCO}(B, C, \tau, k)$ that operates on a function $B: [n]^{\ell} \rightarrow [n]$ (roughly corresponding to a MIF algorithm), a matrix C of random thresholds,⁷ and a stream τ of length $\leq \ell$ that occupies the first $d - k$ time intervals. The recursive structure of $\text{FCO}(B, C, \tau, k)$ is similar to FAO, but crucially, the sets computed by the recursive calls $\text{FCO}(B, C, \tau \circ \rho, k - 1)$ are used differently. Instead of simply returning their union, we use these sets to collect statistics about the outputs in $[n]$ and return only those that are sufficiently common. The thresholds in C control the meaning of “sufficiently common.”

The function B provided to FCO can be either the canonical output function Π of the given pseudo-deterministic algorithm \mathcal{B} or a deterministic algorithm $A \sim \mathcal{B}$ obtained by fixing the random coins of \mathcal{B} . We will show that:

- With high probability over C and the randomness of \mathcal{B} , FCO will produce the same outputs on Π and \mathcal{B} . In other words, FCO is robust to noise (i.e., to algorithm errors).
- When applied to the canonical algorithm, the cardinalities of the sets returned by FCO will grow exponentially with k . Equivalently, similar to $|F_{\tau}|$ from the deterministic proof, the cardinality of $\text{FCO}(B, C, \tau, k)$ will shrink exponentially as the length $|\tau|$ grows. Ultimately, this is proven by implementing AVOID using FCO on the actual algorithm as a subroutine. Critically, this implementation uses the fact that the recursive calls to FCO w.h.p. produce the same output on Π and \mathcal{B} .

⁷ The use of random thresholds is a standard trick for robustly computing quantities in the presence of noise.

- The argument can be carried out with all but one of the d time intervals being of length $\approx \Theta(z)$. If z were too small, d would be large enough that for the empty stream prefix we would have $|\text{FCO}(B, C, \epsilon, d)| > n$, which contradicts $\text{FCO}(\dots) \subseteq [n]$; this lets us derive a lower bound on z .

Error amplification and the case $n \gg \ell$. One technical issue that arises is that the correctness of FCO requires \mathcal{B} 's error probability to be as small as $1/n^{\Omega(\log n)}$. Fortunately, even if the original error probability was $1/3$, we can reduce it to the required level since pseudo-deterministic algorithms allow efficient error reduction by independent repetition. A second technical point is that a z -space pseudo-deterministic algorithm can be shown to have only $O(2^z)$ possible outputs; so if $n \gg \ell$, we can sometimes obtain a stronger lower bound by pretending that n is actually $O(2^z)$. This is formalized by a simple encoding argument.

6.1 Technical Details

Pseudocode. Pseudocode for FCO is given in Procedure 3. The procedure is parameterized by the interval lengths t_d, \dots, t_1 , the set S of all possible *canonical* outputs, and a series of output sizes w_d, \dots, w_1 , where $w_i = 2^{i-1}(t_1 + 1)$.

■ **Procedure 3** The procedure to compute a set for Lemma 11.

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

```

FINDCOMMONOUTPUTS( $B, C, x, k$ ) ▷ abbreviated as  $\text{FCO}(B, C, x, k)$ 
1: ▷ Inputs: function  $B: [n]^\ell \rightarrow [n]$ , matrix  $C \in [1, 2]^{d \times N}$ , stream prefix  $x \in [n]^{t_k + \dots + t_1}$ 
2: ▷ Output: a subset of  $S$  of size  $w_k$ 
3: if  $k = 1$  then
4:    $e_0 \leftarrow B(x \circ \langle 1, 1, \dots, 1 \rangle)$ 
5:   for  $i$  in  $1, \dots, t_1$  do
6:      $e_i \leftarrow B(x \circ \langle e_0, \dots, e_{i-1}, 1, \dots, 1 \rangle)$ 
7:   if  $e_0, \dots, e_{t_1}$  are all distinct then
8:     return  $\{e_0, e_1, \dots, e_{t_1}\}$  ▷ identify  $w_1$  distinct possible outputs
9:   return arbitrary subset of  $S$  of size  $w_1$  (failure)
10: else
11:   for each  $y \in \text{SEQS}([n], t_k)$  do
12:      $T_y \leftarrow \text{FINDCOMMONOUTPUTS}(B, C, x \circ y, k - 1)$  ▷ note  $|T_y| = w_{k-1}$ 
13:    $Q_0 \leftarrow T_{\langle 1, 2, \dots, t_k \rangle}$ 
14:   for  $h$  in  $1, 2, 3, 4$  do
15:     ▷ gather statistics and find common elements among the sets  $T_y$ 
16:     for each  $j \in S$  do
17:        $f_j^{(h)} \leftarrow |\{y \in \text{SEQS}(Q_{h-1}, t_k) : j \in T_y\}|$  ▷ count frequencies
18:      $\theta \leftarrow C_{k,h} w_{k-1} / (16|S|)$  ▷ set random threshold
19:      $P_h \leftarrow \{j \in S : f_j^{(h)} \geq \theta \binom{|Q_{h-1}|}{t_k}\}$  ▷ identify "sufficiently common" elements
20:      $Q_h \leftarrow Q_{h-1} \cup P_h$ 
21:     if  $|Q_h| \geq w_k$  then
22:       return the  $w_k$  smallest elements in  $Q_h$ 
23:   return arbitrary subset of  $S$  of size  $w_k$  (failure)

```

Central lemma. For a series of error probabilities with $1 > \varepsilon_d \gg \varepsilon_{d-1} \dots \gg \varepsilon_1 \approx 1/n^{\Omega(d)}$, we prove, by induction, the following lemma. It asserts that the set of common outputs is likely the same for the canonical function Π as it is for a random draw $A \sim \mathcal{B}$. It also asserts two other key properties of FCO. The lemma can be thought of as a “proof of correctness” of FCO.

► **Lemma 11.** *Let $k \in [d]$ and $x \in [n]^{t_d + \dots + t_{k+1}}$. Then FCO satisfies the following properties.*

1. $\Pr_{A \sim \mathcal{B}, C \sim [1,2]^{d \times N}}[\text{FCO}(A, C, x, k) = \text{FCO}(\Pi, C, x, k)] \geq 1 - \varepsilon_k$.
2. For all $C \in [1, 2]^d$, the set $\text{FCO}(\Pi, C, x, k)$ is disjoint from x and a subset of S .
3. For all $A: [n]^\ell \rightarrow [n]$ and $C \in [1, 2]^d$, $\text{FCO}(A, C, x, k)$ outputs a set of size w_k .

Its proof is split over a number of helper lemmas:

► **Lemma 12.** *Lemma 11 holds for $k = 1$.*

► **Lemma 13.** *Let $x \in [n]^{t_d + \dots + t_{k+1}}$. When computing $\text{FCO}(\Pi, C, x, k)$, in the h th loop iteration, if $|Q_{h-1}| < w_k$, then $|P_h \setminus Q_{h-1}| \geq \lceil \frac{1}{4} w_{k-1} \rceil$. Consequently, the procedure will return using Line 22, not Line 23.*

► **Lemma 14.** *For $k > 1$, $x \in [n]^{t_d + \dots + t_{k+1}}$, $\text{FCO}(\Pi, C, x, k)$ is disjoint from x and a subset of S ; and for all B, C, x, k , $\text{FCO}(B, C, x, k)$ outputs a set of size w_k .*

► **Lemma 15.** *For $k > 1$, and all $x \in [n]^{t_d + \dots + t_{k+1}}$,*

$$\Pr_{A \sim \mathcal{B}, C}[\text{FCO}(A, C, x, k) \neq \text{FCO}(\Pi, C, x, k)] \leq \varepsilon_k.$$

Using the central lemma. A consequence of Lemma 11 is that $\text{FCO}(\Pi, C, d)$ will output a set of size w_d ; this gives a lower bound on n . Solving for a lower bound on z gives:

► **Theorem 16.** *Pseudo-deterministic δ -error random oracle algorithms for $\text{MIF}(n, \ell)$ require*

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

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

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

► **Remark.** For $\delta \leq 2^{-\ell}$, Theorem 16 reproduces the deterministic algorithm space lower bound for $\text{MIF}(n, \ell)$ from [24] within a constant factor.

References

- 1 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. doi:10.1145/3517804.3526228.
- 2 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. doi:10.1145/3584372.3588681.
- 3 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. doi:10.1145/168588.168596.

- 4 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. doi:10.1137/1.9781611977066.15.
- 5 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proc. 39th ACM Symposium on Principles of Database Systems*, pages 63–80, 2020. doi:10.1145/3375395.3387658.
- 6 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. doi:10.1145/3375395.3387643.
- 7 Vladimir Braverman, Robert Krauthgamer, Aditya Krishnan, and Shay Sapir. Lower bounds for pseudo-deterministic counting in a stream. *arXiv preprint arXiv:2303.16287*, 2023. doi:10.48550/arXiv.2303.16287.
- 8 Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Adversarially robust coloring for graph streams. In *Proc. 13th Conference on Innovations in Theoretical Computer Science*, pages 37:1–37:23, 2022. doi:10.4230/LIPIcs.ITCS.2022.37.
- 9 Amit Chakrabarti and Manuel Stoeckl. Finding missing items requires strong forms of randomness. *arXiv preprint*, 2024. arXiv:2310.03634.
- 10 Uriel Feige. A randomized strategy in the mirror game. *arXiv preprint*, 2019. doi:10.48550/arXiv.1901.07809.
- 11 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. doi:10.4230/LIPIcs.ITCS.2019.36.
- 12 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. doi:10.4230/LIPIcs.ITCS.2020.79.
- 13 Ofer Grossman, Meghal Gupta, and Mark Sellke. Tight space lower bound for pseudo-deterministic approximate counting. *arXiv preprint*, 2023. doi:10.48550/arXiv.2304.01438.
- 14 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. doi:10.1145/2488608.2488624.
- 15 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. URL: <https://proceedings.neurips.cc/paper/2020/hash/0172d289da48c48de8c5ebf3de9f7ee1-Abstract.html>.
- 16 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006. doi:10.1145/1147954.1147955.
- 17 Rajesh Jayaram and David P Woodruff. Towards optimal moment estimation in streaming and distributed models. *ACM Trans. Alg.*, 19(3):1–35, 2023. doi:10.1145/3596494.
- 18 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. doi:10.1007/978-3-030-84252-9_4.
- 19 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. doi:10.4230/LIPIcs.FUN.2022.20.
- 20 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. doi:10.4230/LIPIcs.ITCS.2022.107.
- 21 Ilan Newman. Private vs. common random bits in communication complexity. *Inform. Process. Lett.*, 39(2):67–71, 1991. doi:10.1016/0020-0190(91)90157-D.

28:20 Finding Missing Items Requires Strong Forms of Randomness

- 22 Noam Nisan. Pseudorandom generators for space-bounded computation. In *Proc. 22nd Annual ACM Symposium on the Theory of Computing*, pages 204–212, 1990. doi:10.1145/100216.100242.
- 23 Noam Nisan. On read once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993. doi:10.1016/0304-3975(93)90258-U.
- 24 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](https://arxiv.org/abs/2211.05170v1). doi:10.1137/1.9781611977554.ch32.
- 25 Manuel Stoeckl. *On adaptivity and randomness for streaming algorithms*. PhD thesis, Dartmouth College, 2024. URL: <https://digitalcommons.dartmouth.edu/dissertations/229/>.
- 26 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. doi:10.1109/FOCS52979.2021.00116.