Pseudo-Deterministic Streaming

Shafi Goldwasser
Simons Institute for the Theory of Computing, Berkeley, CA, USA
MIT, Cambridge, MA, USA
http://people.csail.mit.edu/shafi/
shafi.goldwasser@gmail.com

Ofer Grossman
MIT, Cambridge, MA, USA
ofer.grossman@gmail.com

Sidhanth Mohanty
University of California Berkeley, CA, USA
http://sidhanthm.com/
sidhanthm@cs.berkeley.edu

David P. Woodruff
Carnegie-Mellon University, Pittsburgh, PA, USA
http://www.cs.cmu.edu/~dwoodruf/
dwoodruf@cs.cmu.edu

Abstract

A pseudo-deterministic algorithm is a (randomized) algorithm which, when run multiple times on the same input, with high probability outputs the same result on all executions. Classic streaming algorithms, such as those for finding heavy hitters, approximate counting, ℓ_2 approximation, finding a nonzero entry in a vector (for turnstile algorithms) are not pseudo-deterministic. For example, in the instance of finding a nonzero entry in a vector, for any known low-space algorithm A, there exists a stream x so that running A twice on x (using different randomness) would with high probability result in two different entries as the output.

In this work, we study whether it is inherent that these algorithms output different values on different executions. That is, we ask whether these problems have low-memory pseudo-deterministic algorithms. For instance, we show that there is no low-memory pseudo-deterministic algorithm for finding a nonzero entry in a vector (given in a turnstile fashion), and also that there is no low-dimensional pseudo-deterministic sketching algorithm for ℓ_2 norm estimation. We also exhibit problems which do have low memory pseudo-deterministic algorithms but no low memory deterministic algorithm, such as outputting a nonzero row of a matrix, or outputting a basis for the row-span of a matrix.

We also investigate multi-pseudo-deterministic algorithms: algorithms which with high probability output one of a few options. We show the first lower bounds for such algorithms. This implies that there are streaming problems such that every low space algorithm for the problem must have inputs where there are many valid outputs, all with a significant probability of being outputted.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases streaming, pseudo-deterministic

Digital Object Identifier 10.4230/LIPIcs.ITCS.2020.79

Funding *Shafi Goldwasser*: Supported by NSF CNS-1413920, DARPA/NJIT 491512803, Sloan Foundation 996698, and MIT/IBM W1771646. This work was done at the Simons Institute for the Theory of Computing.
Ofer Grossman: Supported by the Fannie and John Hertz Foundation fellowship, an NSF GRFP award, NSF CNS-1413920, DARPA/NJIT 491512803, Sloan Foundation 996698, and MIT/IBM W1771646. This work was done in part at the Simons Institute for the Theory of Computing.

11th Innovations in Theoretical Computer Science Conference (ITCS 2020).
Editor: Thomas Vidick; Article No. 79; pp. 79:1–79:25

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Sidhanth Mohanty: Supported by NSF grant CCF-1718695. This work was done in part at the Simons Institute for the Theory of Computing.
David P. Woodruff: Supported by the National Science Foundation under Grant No. CCF-1815840. This work was done in part at the Simons Institute for the Theory of Computing.

1 Introduction

Consider some classic streaming problems: heavy hitters, approximate counting, ℓ_p approximation, finding a nonzero entry in a vector (for turnstile algorithms), counting the number of distinct elements in a stream. These problems were shown to have low-space randomized algorithms in [4, 27, 7, 2, 20, 26], respectively. All of these algorithms exhibit the property that when running the algorithm multiple times on the same stream, different outputs may result on the different executions.

For the sake of concreteness, let's consider the problem of ℓ_2 approximation: given a stream of poly(n) updates to a vector (the vector begins as the zero vector, and updates are of the form "increase the i^{th} entry by 1" or "decrease the j^{th} entry by 1"), output an approximation of the ℓ_2 norm of the vector. There exists a celebrated randomized algorithm for this problem [2]. This algorithm has the curious property that running the same algorithm multiple times on the same stream may result in different approximations. That is, if Alice runs the algorithm on the same stream as Bob (but using different randomness), Alice may get some approximation of the ℓ_2 norm (such as 27839.8), and Bob (running the same algorithm, but with your own randomness) may get a different approximation (such as 27840.2). The randomized algorithm has the guarantee that both of the approximations will be close to the true value. However, interestingly, Alice and Bob end up with slightly different approximations. Is this behavior inherent? That is, could there exist an algorithm which, while being randomized, for all streams with high probability both Alice and Bob will end up with the *same* approximation for the ℓ_2 norm?

Such an algorithm, which when run on the same stream multiple times outputs the same output with high probability is called *pseudo-deterministic*. The main question we tackle in this paper is:

What streaming problems have low-memory pseudo-deterministic algorithms?

1.1 Our Contributions

This paper is the first to investigate pseudo-determinism in the context of streaming algorithms. We show certain problems have pseudo-deterministic algorithms substantially faster than the optimal deterministic algorithm, while other problems do not.

1.1.1 Lower Bounds

1.1.1.1 Find-Support-Elem

We show pseudo-deterministic lower bounds for finding a nonzero entry in a vector in the turnstile model. Specifically, consider the problem Find-Support-Elem of finding a nonzero entry in a vector for a turnstile algorithm (the input is a stream of updates of the form "increase entry i by 1" or "decrease entry j by 1", and we wish to find a nonzero entry in the final vector). We show this problem does not have a low-memory pseudo-deterministic algorithm:

▶ **Theorem 1.** *There is no pseudo-deterministic algorithm for* Find-Support-Elem *which uses* $\tilde{o}(n)$ *memory.*

This is in contrast with the work of [26], which shows a randomized algorithm for the problem using polylogarithmic space.

Theorem 1 can be viewed as showing that any low-memory algorithm A for Find-Support-Elem must have an input x where the output $A(x)$ (viewed as a random variable depending on the randomness used by A) must have at least a little bit of entropy. The algorithms we know for Find-Support-Elem have a very high amount of entropy in their outputs (the standard algorithms, for an input which is the all 1s vector, will find a uniformly random entry). Is this inherent, or can the entropy of the output be reduced? We show that this is inherent: for every low memory algorithm there is an input x such that $A(x)$ has high entropy.

▶ **Theorem 2.** *Every randomized algorithm for* Find-Support-Elem *using* $o(s)$ *space must have an input* x *such that* $A(x)$ *has entropy at least* $\log\left(\frac{n}{s\log n}\right)$.

So, in particular, an algorithm using $n^{1-\varepsilon}$ space must have outputs with entropy $\Omega(\log n)$, which is maximal up to constant factors.

We also show analogous lower bounds for the problem Find-Duplicatein which the input is a stream of $3n/2$ integers between 1 and n, and the goal is to output a number k which appears at least twice in the stream:

▶ **Theorem 3.** *Every randomized algorithm for* Find-Duplicate *using* $o(s)$ *space must have an input* x *such that* $A(x)$ *has entropy at least* $\log\left(\frac{n}{s\log n}\right)$.

1.1.1.2 Techniques

To prove a pseudo-deterministic lower bound for Find-Support-Elem, the idea is to show that if a pseudo-deterministic algorithm existed for Find-Support-Elem, then there would also exist a pseudo-deterministic one-way communication protocol for the problem One-Way-Find-Duplicate, where Alice has a subset of $[n]$ of size $3n/4$, and so does Bob, and they wish to find an element which they share.

To prove a lower bound on the one-way communication problem One-Way-Find-Duplicate, we show that if such a pseudo-deterministic protocol existed, then Bob can use Alice's message to recover many $(n/10)$ elements of her input (which contains much more information than one short message). The idea is that using Alice's message, Bob can find an element they have in common. Then, he can remove the element he found that they have in common from his input, and repeat to find another element they have in common (using the original message Alice sent, so Alice does not have to send another message). After repeating $n/10$ times, he will have found many elements which Alice has.

It may not be immediately obvious where pseudo-determinism is being used in this proof. The idea is that because the algorithm is pseudo-deterministic, the element which Bob finds as the intersection with high probability does not depend on the randomness used by Alice. That is, let $b_1, b_2, \ldots$ be the sequence of elements which Bob finds. Because the algorithm is pseudo-deterministic, there exists a specific sequence $b_1, b_2, \ldots$ such that with high probability this will be the sequence of elements which Bob finds. Notice that a randomized (but not pseudo-deterministic) algorithm for One-Way-Find-Duplicatewould result in different sequences on different executions.

When the sequence $b_1, b_2, \ldots$ is determined in advance, we can use a union bound and argue that with high probability, one of Alice's messages will likely work on all of Bob's inputs. If $b_1, b_2, \ldots$ is not determined in advance, then it's not possible to use a union bound.

Proving a lower bound on the entropy of the output of an algorithm for Find-Support-Elem uses a similar idea, but is more technically involved. It is harder to ensure that Bob's later inputs will be able to succeed with Alice's original message. The idea, at a very high level, is to have Alice send many messages (but not too many), so that Bob's new inputs will not strongly depend on any part of Alice's randomness, and also to have Alice send additional messages to keep Bob from going down a path where Alice's messages will no longer work.

This lower bound technique may seem similar to the way one would show a deterministic lower bound. It's worth noting that for certain problems, deterministic lower bounds do not generalize to pseudo-deterministic lower bounds; see our results on pseudo-deterministic upper bounds for some examples and intuition for why certain problems remain hard in the pseudo-deterministic setting while others do not.

1.1.1.3 Sketching lower bounds for pseudo-deterministic ℓ_2 norm estimation

The known randomized algorithms (such as [2]) for approximating the ℓ_2 norm of a vector x in a stream rely on *sketching*, i.e., storing $\boldsymbol{S}x$ where $\boldsymbol{S}$ is a $d \times n$ random matrix where $d \ll n$ and outputting the ℓ_2 norm of $\boldsymbol{S}x$. More generally, an abstraction of this framework is the setting where one has a distribution over matrices $\mathcal{D}$ and a function f. One then stores a *sketch* of the input vector $\boldsymbol{S}x$ where $\boldsymbol{S} \sim \mathcal{D}$ and outputs $f(\boldsymbol{S}x)$. By far, most streaming algorithms fall into this framework and in fact some recent work [24, 1] proves under some caveats and assumptions that low-space turnstile streaming algorithms imply algorithms based on low-dimensional sketches. Since sketching-based streaming algorithms are provably optimal in many settings, it motivates studying whether there are low-dimensional sketches of x from which the ℓ_2 norm can be estimated pseudo-deterministically.

We prove a lower bound on the dimension of sketches from which the ℓ_2 norm can be estimated pseudo-deterministically:

▶ **Theorem 4.** *Suppose $\mathcal{D}$ is a distribution over $d \times n$ matrices and f is a function from $\mathbb{R}^d$ to $\mathbb{R}$ such that for all $x \in \mathbb{R}^n$, when $\boldsymbol{S} \sim \mathcal{D}$:*

- *$f(\boldsymbol{S}x)$ approximates the ℓ_2 norm of x to within a constant factor with high probability,*
- *$f(\boldsymbol{S}x)$ takes a unique value with high probability.*

Then d must be $\Omega(n)$.

As an extension, we also show that

▶ **Theorem 5.** *For every constant $\varepsilon, \delta > 0$, every randomized sketching algorithm A for ℓ_2 norm estimation using a $O(n^{1-\delta})$-dimensional sketch, there is a vector x such that the output entropy of $A(x)$ is at least $1 - \varepsilon$. Furthermore, there is a randomized algorithm using a $O(\operatorname{poly}\log n)$-dimensional sketch with output entropy at most $1 + \varepsilon$ on all input vectors.*

1.1.1.4 Techniques

The first insight in our lower bound is that if there is a pseudo-deterministic streaming algorithm A for ℓ_2 norm estimation in k space, then that means there is a fixed function g such that $g(x)$ approximates $\|x\|_2$ and A is a randomized algorithm to compute $g(x)$ with high probability. The next step uses a result in the work of [17] to illustrate a (randomized) sequence of vectors $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}$ only depending on g such that any linear sketching-based

algorithm that uses sublinear dimensional sketches outputs an incorrect approximation to the ℓ_2 norm of some vector in that sequence with constant probability, thereby implying a dimension lower bound.

1.1.2 Upper Bounds

On the one hand, all the problems considered so far were such that

1. There were "low-space" randomized algorithms.
2. The pseudo-deterministic and deterministic space complexity were "high" and equal up to logarithmic factors.

This raises the question if there are natural problems where pseudo-deterministic algorithms outperform deterministic algorithms (by more than logarithmic factors). We answer this question in the affirmative.

We illustrate several natural problems where the pseudo-deterministic space complexity is strictly smaller than the deterministic space complexity.

The first problem is that of finding a nonzero row in a matrix given as input in a turnstile stream. Our result for this problem has the bonus of giving a natural problem where the pseudo-deterministic streaming space complexity is strictly sandwiched between the deterministic and randomized streaming space complexity.

In the problem Find-Nonzero-Row, the input is an $n \times d$ matrix A streamed in the turnstile model, and the goal is to output an i such that the i^{th} row of the matrix A is nonzero.

▶ **Theorem 6.** *The randomized space complexity for* Find-Nonzero-Row *is* $\widetilde{\Theta}(1)$*, the pseudo-deterministic space complexity for* Find-Nonzero-Row *is* $\widetilde{\Theta}(n)$*, and the deterministic space complexity for* Find-Nonzero-Row *is* $\widetilde{\Theta}(nd)$*.*

The idea behind the proof of Theorem 6 is to sample a random vector x, and then deterministically find a nonzero entry of Ax. With high probability, if a row of A is nonzero, then the corresponding entry of Ax will be nonzero as well.

1.1.2.1 Discussion

Roughly speaking, in this problem there is a certain structure that allows us to use randomness to "hash" pieces of the input together, and then apply a deterministic algorithm on the hashed pieces. The other upper bounds we show for pseudo-deterministic algorithms also have a structure which allows us to hash, and then use a deterministic algorithm. It is interesting to ask if there are natural problems which have faster pseudo-deterministic algorithms than the best deterministic algorithms, but for which the pseudo-deterministic algorithms follow a different structure.

The next problems we show upper bounds for are estimating frequencies in a length-m stream of elements from a large universe $[n]$ up to error εm, and that of estimating the inner product of two vectors x and y in an insertion-only stream of length-m up to error $\varepsilon \cdot \|x\|_1 \cdot \|y\|_1$. We show a separation between the deterministic and (weak) pseudo-deterministic space complexity in the regime where $m \ll n$.

▶ **Theorem 7.** *There is a pseudo-deterministic algorithm for point query estimation and inner product estimation that uses* $O\left(\frac{\log m}{\varepsilon} + \log n\right)$ *bits of space. On the other hand, any deterministic algorithm needs* $\Omega\left(\frac{\log n}{\varepsilon}\right)$ *bits of space.*

1.2 Related work

Pseudo-deterministic algorithms were introduced by Gat and Goldwasser [9]. Such algorithms have since been studied in the context of standard (sequential algorithms) [15, 30], average case algorithms [18], parallel algorithms [12], decision tree algorithms [11, 10], interactive proofs [13], learning algorithms [31], approximation algorithms [31, 6], and low space algorithms [16]. In this work, we initiate the study of pseudo-determinism in the context of streaming algorithms (and in the context of one-way communication complexity).

The problem of finding duplicates in a stream of integers between 1 and n was first considered by [14], where an $O(\log^3 n)$ bits of space algorithm is given, later improved by [22] to $O(\log^2 n)$ bits. We show that in contrast to these low space randomized algorithms, a pseudo-deterministic algorithm needs significantly more space in the regime where the length of the stream is, say, $3n/2$. [23] shows optimal lower bounds for randomized algorithms solving the problem.

The method of ℓ_p-sampling to sample an index of a turnstile vector with probability proportional to its ℓ_p mass, whose study was initiated in [26], is one way of outputting an element from the support of a turnstile stream. A line of work [8, 26, 22, 3], ultimately leading to an optimal algorithm in [21] and tight lower bounds in [23], characterizes the space complexity of randomized algorithms to output an element from the support of a turnstile vector as $\Theta(\operatorname{poly}\log n)$, in contrast with the space lower bounds we show for algorithms constrained to a low entropy output.

1.3 Open Problems

Morris Counters

In [27], Morris showed that one can approximate (up to a multiplicative error) the number of elements in a stream with up to n elements using $O(\log\log n)$ bits of space. Does there exist an $O(\log\log n)$ bits of space pseudo-deterministic algorithm for the problem?

ℓ_2-norm estimation

In this work, we show that there are no low-dimensional pseudo-deterministic sketching algorithms for estimating the ℓ_2-norm of a vector. However, we do not show a turnstile streaming lower bound for pseudo-deterministic algorithms, which motivates the following question. Does there exist a $O(\operatorname{poly}\log n)$ space pseudo-deterministic algorithm for ℓ_2-norm estimation?

Multi-pass streaming lower bounds

All the streaming lower bounds we prove are in the single pass model, i.e., where the algorithm receives the stream exactly once. How do these lower bounds extend to the multi-pass model, where the algorithm receives the stream multiple times? All of the pseudo-deterministic streaming lower bounds in this paper do not even extend to 2-pass streaming algorithms.

1.4 Table of complexities

In the below table, we outline the known space complexity of various problems considered in our work.

Table 1 Table of space complexities.

Problem	Randomized	Deterministic	Pseudo-deterministic
Morris Counters	$\Theta(\log\log n)$	$\Theta(\log n)$	$O(\log n)$, $\Omega(\log\log n)$
Find-Duplicate	$\Theta(\log n)$	$\Theta(n)$	$\widetilde{\Theta}(n)$
ℓ_2-approximation (streaming)	$\Theta(\log n)$	$\widetilde{\Theta}(n)$	$\widetilde{\Theta}(n)$
ℓ_2-approximation (sketching)			$\widetilde{O}(n)$, $\widetilde{\Omega}(\log n)$
Find-Nonzero-Row	$\widetilde{\Theta}(1)$	$\widetilde{\Theta}(nd)$	$\widetilde{\Theta}(n)$

2 Preliminaries

A randomized algorithm is called *pseudo-deterministic* if for every valid input x, when running the algorithm twice on x, the same output is obtained with probability at least $2/3$. Equivalently (up to amplification of error probabilities), one can think of an algorithm as pseudo-deterministic if for every input x, there is a unique value $f(x)$ such that with probability at least $2/3$ the algorithm outputs $f(x)$ on input x

▶ **Definition 8** (Pseudo-deterministic). *A (randomized) algorithm A is called pseudo-deterministic if for all valid inputs x, the algorithm A satisfies*

$$\Pr_{r_1,r_2}[A(x,r_1) = A(x,r_2)] \geq 2/3.$$

An extension of pseudo-determinism is that of k-entropy randomized algorithms [16]. Such algorithms have the guarantee that for every input x, the distribution $A(x,r)$ (over a random choice of randomness r) has low entropy, in particular bounded by k.

Another extension of pseudo-determinism is that of m-pseudo-deterministic algorithms, from [10]. Intuitively speaking, any algorithm is k-pseudo-deterministic if for every valid input, with high probability the algorithm outputs one of k options (so, a 1-pseudo-deterministic algorithm is the same as a standard pseudo-deterministic algorithm, since it outputs the one unique option with high probability):

▶ **Definition 9** (k-pseudo-deterministic). *We say that an algorithm A is k-pseudo-deterministic if for all valid inputs x, there is a set $S(x)$ of size at most k, such that $\Pr_r[A(x,r) \in S(x)] \geq \frac{k+1}{k+2}$*

For the purposes of this work, we define a simple notion that we call a *k-concentrated algorithm.*

▶ **Definition 10.** *We say that an algorithm A is k-concentrated if for all valid inputs x, there is some output $F(x)$ such that $\Pr_r[A(x,r) = F(x)] \geq \frac{1}{k}$.*

The reason for making this definition is that any $\log k$-entropy randomized algorithm, and any $(k+2)$-pseudo-deterministic algorithm is k-concentrated. Thus, showing an impossibility result for k-concentrated algorithms also shows an impossibility result for $\log k$-entropy and $(k+2)$-pseudo-deterministic algorithms. Indeed, in this work, we use space lower bounds against k-concentrated algorithms to simultaneously conclude space lower bounds against low entropy and multi-pseudo-deterministic algorithms.

▶ **Definition 11.** *A turnstile streaming algorithm is one where there is a vector v, and the input is a stream of updates of the form "increase the i^{th} coordinate of v by r" or "decrease the i^{th} coordinate of v by r'". The goal is to compute something about the final vector, after all of the updates.*

We use a pseudorandom generator for space-bounded computation due to Nisan [29], which we recap below.

▶ **Theorem 12.** *There is a function* $G : \{0,1\}^{s \log r} \to \{0,1\}^r$ *such that*

1. *Any bit of* $G(x)$ *for any input* x *can be computed in* $O(s \log r)$ *space.*
2. *For all functions* f *from* $\{0,1\}^r$ *to some set* A *such that* f *is computable by a finite state machine on* 2^s *states, the total variation distance between the random variables* $f(\boldsymbol{x})$ *and* $f(G(\boldsymbol{y}))$ *where* $\boldsymbol{x}$ *is uniformly drawn from* $\{0,1\}^r$ *and* $\boldsymbol{y}$ *is uniformly drawn from* $\{0,1\}^{s \log r}$ *is at most* 2^{-s}.

3 Find-Duplicate: Pseudo-deterministic lower bounds

Consider the following problem: the input is a stream of $3n/2$ integers between 1 and n. The goal is to output a number k which appears at least twice in the stream. Call this problem Find-Duplicate. Recall that this problem has been considered in the past literature, specifically in [14, 22, 23], where upper and lower bounds for randomized algorithms have been shown.

Indeed, we know the following is true from [14, 22].

▶ **Theorem 13.** Find-Duplicate *has an algorithm which uses* $O(\text{poly} \log n)$ *memory and succeeds with all but probability* $\frac{1}{\text{poly}(n)}$.

We formally define a *pseudo-deterministic streaming algorithm* and show a pseudo-deterministic lower bound for Find-Duplicate to contrast with the randomized algorithm from Theorem 13.

▶ **Definition 14** (Pseudo-deterministic Streaming Algorithm)**.** *A pseudo-deterministic streaming algorithm is a (randomized) streaming algorithm* A *such that for all valid input streams* $s = \langle x_1, \ldots, x_m \rangle$*, the algorithm* A *satisfies* $\Pr_{r_1,r_2}[(A(x,r_1) = A(x,r_2)] \geq 2/3$.

One can also think of a pseudo-deterministic streaming algorithm as an algorithm A such that for every valid input stream s, there exists some valid output $f(s)$ such that the algorithm A outputs $f(s)$ with probability at least $2/3$ (one would have to amplify the success probability using repetition to see that this alternate notion is the same as the definition above).

▶ **Definition 15** (Find-Duplicate)**.** *Define* Find-Duplicate *to be the streaming problem where the input is a stream of length* $3n/2$ *consisting of up to* n*, and the output must be an integer which has occured at least twice in the string.*

▶ **Theorem 16.** Find-Duplicate *has no pseudo-deterministic algorithm with memory* $o(n)$.

Proof Overview

In order to prove Theorem 16, we introduce two communication complexity problems – One-Way-Find-Duplicate and One-Way-Partial-Recovery:

In the One-Way-Find-Duplicate problem, Alice has a list of $3n/4$ integers between 1 and n, and so does Bob. Alice sends a message to Bob, after which Bob must output an integer which is in both Alice's and Bob's list. Formally:

▶ **Definition 17** (One-Way-Find-Duplicate)**.** *Define* One-Way-Find-Duplicate *to be the one-way communication complexity problem where Alice has input* $S_A \subseteq [n]$ *and Bob has input* $S_B \subseteq [n]$*, where* $|S_A|, |S_B| \geq 3n/4$*. The goal is for Bob to output an element in* $S_A \cap S_B$.

The idea is that one can reduce One-Way-Find-Duplicate to Find-Duplicate. So, our new goal will be to show that One-Way-Find-Duplicate requires high communication. To do so, we will show that it is possible to reduce a different problem, denoted One-Way-Partial-Recovery(defined below), to One-Way-Find-Duplicate. Informally, in the One-Way-Partial-Recovery problem, Alice has a list of $3n/4$ integers between 1 and n. Bob does not have an input. Alice sends a message to Bob, after which Bob must output $n/10$ distinct elements which are all in Alice's list. Formally:

▶ **Definition 18** (One-Way-Partial-Recovery)**.** *Define* One-Way-Partial-Recovery *to be the one-way communication complexity problem where Alice has input $S_A \subseteq [n]$ and Bob has no input. The goal is for Bob to output a set S satisfying $S \subseteq S_A$ and $|S| \geq n/10$.*

We will show in Claim 19 that a low memory pseudo-deterministic algorithm for Find-Duplicate implies a low-communication pseudo-deterministic algorithm for One-Way-Find-Duplicate, and in Claim 20 that a low-communication pseudo-deterministic algorithm for One-Way-Find-Duplicate implies a low communication algorithm for One-Way-Partial-Recovery. Finally, in Claim 21, we show that One-Way-Partial-Recovery cannot be solved with low communication. Combining the claims yields Theorem 16.

Proof of Theorem 16.

▷ Claim 19. A pseudo-deterministic algorithm for Find-Duplicate with space S and success probability p implies a pseudo-deterministic communication protocol for One-Way-Find-Duplicate with communication S and success probability at least p.

Proof. To prove the above claim, we construct a protocol for One-Way-Find-Duplicate from a streaming algorithm for Find-Duplicate. Given an instance of One-Way-Find-Duplicate, Alice can stream her input set of integers in increasing order, and simulate the streaming algorithm for Find-Duplicate. Then, she sends the current state of the algorithm (which is at most S bits) to Bob, who continues the execution of the streaming algorithm. At the end, the streaming algorithm outputs a repetition with probability p, which means the element showed up in both Alice and Bob's lists. Note that for a given input to Alice and Bob, Bob outputs a unique element with high probability because the streaming algorithm is pseudo-deterministic. ◁

▷ Claim 20. A pseudo-deterministic one-way communication protocol for One-Way-Find-Duplicate with S communication and failure probability $O\left(\frac{1}{n^2}\right)$ implies a pseudo-deterministic communication protocol for One-Way-Partial-Recovery with S communication and $O\left(\frac{1}{n}\right)$ failure probability.

Proof. We will show how to use a protocol for One-Way-Find-Duplicate to solve the instance of One-Way-Partial-Recovery.

Suppose we have an instance of One-Way-Partial-Recovery. Alice sends the same message to Bob as if the input was an instance of One-Way-Find-Duplicate, which is valid since in both of these problems, Alice's input is a list of length $3n/4$ of integers between 1 and n.

Now, Bob's goal is to use the message sent by Alice to recover $n/10$ elements of Alice. Let X be the (initially empty) set of elements of Alice's input that Bob knows and let B be a set of $3n/4$ elements in $\{1, \ldots, n\}$ disjoint from X, where we initially set B to $\{1, 2, \ldots, n\}$. While the size of X is less than $n/10$, Bob simulates the protocol of One-Way-Find-Duplicate with Alice's message and input B. This will result in Bob finding a single element x in Alice's input that is (i) in B, and (ii) not in X. Bob adds x to X, and deletes x from B. Once the size of X is $n/10$, Bob outputs X.

If Alice has the set A as her input, define $f_A(B)$ to be the output which the pseudo-deterministic algorithm for One-Way-Find-Duplicate outputs with high probability when Alice's input is A and Bob's input is B. Now, set $B_0 = \{1, 2, \ldots, n\}$, and $B_i = B_{i-1} \setminus \{f_A(B)\}$. Note that these B_i (for $i = 0$ through $n/10$) are the sets which, assuming the pseudo-deterministic algorithm never errs during the reduction (where we say the algorithm errs if it does not output the unique element which is guaranteed to be output with high probability), Bob will use as his inputs for the simulated executions of One-Way-Find-Duplicate. The pseudo-deterministic algorithm does not err on any of the B_i except with probability at most $1/n$, by a union bound. If Bob succeeds on all of the B_i, that means that the sequence of inputs which will be his inputs for the simulated executions of One-Way-Find-Duplicate are indeed $B_0, B_1, \ldots, B_{n/10}$. So, since we have shown with high probability the algorithms succeeds on all of the B_i, and therefore with high probability the B_i are also Bob's inputs for the simulated executions of One-Way-Find-Duplicate, we see that with high probability Bob will succeed on all of the $n/10$ inputs he tries to simulate executions of One-Way-Find-Duplicate with.

Note that we used the union bound over all the B_i for $i = 1$ through $n/10$. All of these B_i are a function of A. In particular, notice that by definition, the B_i do not depend on the randomness chosen by Alice. ◁

▷ Claim 21. Every pseudo-deterministic One-Way-Partial-Recovery protocol which succeeds with probability at least $\frac{2}{3}$ requires $\Omega(n)$ bits of communication.

Proof. We prove this lower bound by showing that a protocol for One-Way-Partial-Recovery can be used to obtain a protocol with exactly the same communication for the problem where Alice is given a string x in $\{0,1\}^{Cn}$ as input, she sends a message to Bob, and Bob must exactly recover x from Alice's message with probability at least $2/3$. This problem has a lower bound of $\Omega(n)$ bits of communication.

Suppose there exists a pseudo-deterministic algorithm for One-Way-Partial-Recovery. Given such a pseudo-deterministic protocol that succeeds with probability at least $2/3$, there is a function F such that $F(S)$ (a set with $n/10$ elements) is Bob's output after the protocol with probability at least $2/3$ when Alice is given S as input.

We will construct sets $S_1, \ldots, S_t$ to be subsets of $[n]$ of size $3n/4$ such that for any $i \neq j$, $F(S_i)$ is not a subset of S_j. To do so, we use the probabilistic method: set $S_1, \ldots, S_t$ be random subsets of $[n]$ of size $3n/4$. The probability that $F(S_i)$ is contained S_j for fixed $i \neq j$ is at most $\left(\frac{3}{4}\right)^{n/10}$. Thus, by a union bound, the probability that for any $i \neq j$, $F(S_i)$ is contained S_j is at most $t^2 \left(\frac{3}{4}\right)^{n/10}$, a quantity which is strictly less than 1 when t is $\left(\frac{4}{3}\right)^{n/100}$, so $S_1, \ldots, S_t$ satisfying the desired guarantee exist.

Alice and Bob can (ahead of time) agree on an encoding of $\lfloor \log t \rfloor$-bit strings that is an injective function G from $\{0,1\}^{\lfloor \log t \rfloor}$ to $\{S_1, \ldots, S_t\}$. Now, if Alice is given a $\lfloor \log t \rfloor$-bit string x as input, she can send a message to Bob according to the pseudo-deterministic protocol for One-Way-Partial-Recovery by treating her input as $G(x)$. Bob then recovers $F(G(x))$ with probability at least $2/3$, and can use it to recover $G(x)$ since there is unique S_i in which $F(G(x))$ is contained. Since G is injective, Bob can also recover x with probability $2/3$.

This reduction establishes a lower bound of $\Omega(\lfloor \log t \rfloor)$ on the pseudo-deterministic communication complexity of One-Way-Partial-Recovery, which is an $\Omega(n)$ lower bound. ◁

Combining Claim 19, Claim 20 and Claim 21 completes the proof of Theorem 16. ◀

It is worth noting that the problem has pseudo-deterministic algorithms with sublinear space if one allows multiple passes through the input. Informally, a p-pass streaming algorithm is a streaming algorithm which, instead of seeing the stream only once, gets to see the stream p times.

▷ Claim 22. There is a p-pass deterministic streaming algorithm that uses $\widetilde{O}(n^{1/p})$ memory for the Find-Duplicate problem.

Proof. At the start of t-th pass, the algorithm maintains a candidate interval I of width $n^{1-(t-1)/p}$ from which it seeks to find a repeated element. At the very beginning, this candidate interval is $[1, n]$. In the t-th pass, first partition the interval into $n^{1/p}$ equal sized intervals $I'_1, \ldots, I'_{n^{1/p}}$, each of whose width (the width of an interval $[a, b]$ is $b - a$) is $n^{1-t/p}$ and count the number of elements of the stream that lie in each such subinterval – this count must exceed the width of at least one subinterval I'_t. Update I to I'_t and proceed to the next pass. After p passes, this interval will contain at most 1 integer. ◁

4 Entropy Lower Bound for Find-Duplicate

▶ **Theorem 23.** *Every zero-error randomized algorithm for* Find-Duplicate *that is* $\frac{n}{s}$*-concentrated must use* $\Omega\left(\frac{s}{\log n}\right)$ *space.*

By *zero error*, we mean that the algorithm never outputs a number k which is not repeated. With probability one it either outputs a valid output, or $\perp$.

Proof. We use a reduction similar to that of the pseudo-deterministic case (cf. Proof of Claim 20). Using the exact same reduction from the proof of Claim 19, we get that a $\frac{n}{s}$-concentrated streaming algorithm for Find-Duplicate using T space must give us a $\frac{n}{s}$-concentrated protocol for One-Way-Find-Duplicate with communication complexity T. If we can give a way to convert such a protocol for One-Way-Find-Duplicate into an $O\left(\frac{Tn \log n}{s}\right)$-communication protocol for One-Way-Partial-Recovery, the desired lower bound on T follows from the lower bound on communication complexity of One-Way-Partial-Recovery from Claim 21. We will now show how to make such a conversion by describing a protocol for One-Way-Partial-Recovery.

Alice sends Bob $\Theta(n \log n/s)$ messages according to the protocol for One-Way-Find-Duplicate (that is, she simulates the protocol for One-Way-Find-Duplicate a total of $\Theta(n \log n/s)$ times). Bob's goal is to use these $\Theta(n \log n/s)$ messages to recover at least $n/10$ input elements of Alice. Towards this goal, he maintains a set of elements recovered so far, X (initially empty), and a family of "active sets" $\mathcal{B}$ (initially containing the set $\{1, 2, \ldots, n\}$). While the size of X is smaller than $n/10$, Bob simulates the remainder of the One-Way-Find-Duplicate protocol on every possible pair (B, M) where B is a set in $\mathcal{B}$ and M is one of the messages of Alice. For each such pair (B, M) where the protocol is successful in finding a duplicate element x, Bob adds x to X, removes B from $\mathcal{B}$ and adds $B \setminus \{x\}$ to $\mathcal{B}$.

We now wish to prove that this protocol indeed lets Bob recover $n/10$ elements of Alice. Suppose Alice has input A. For each set S, define $f_A(S)$ be an element of $A \cap S$ that has probability at least s/n of being outputted by Bob on input S at the end of a One-Way-Find-Duplicate protocol. Let $S_0 := \{1, 2, \ldots, n\}$ and $S_i := S_{i-1} \setminus \{f_A(S_i)\}$ be defined for $0 \leq i \leq n/10$. Note that S_i are predetermined: it is a function of Alice's input (and, in particular, not a function of the randomness she uses when choosing her messages). For a fixed i, the probability of failure to recover $f_A(S_i)$ from any of Alice's messages is at most

$1/n^2$. A failure to fill in X with $n/10$ elements implies that for some i, Bob failed to recover $f_A(S_i)$ from all of Alice's messages. The probability that such a failure happens for a specific i is at most $(1-s/n)^{\Theta(n\log n/s)}$. By setting the constant in the Θ to be large enough, we can have this be at most $\frac{1}{n^2}$, and so by a union bound the probability that there is an i such that $f_A(S_i)$ is not recovered by Bob is at most $1/n$.

Thus, we obtain a protocol for One-Way-Partial-Recovery with communication complexity $O(Tn\log n/s)$, and so $T \le s/\log n$, completing the proof. ◀

We obtain the following as immediate corollaries:

▶ **Corollary 24.** *Any zero-error* $\log\left(\frac{n}{s}\right)$*-entropy randomized algorithm for* Find-Duplicate *must use* $\Omega\left(\frac{s}{\log n}\right)$ *space.*

▶ **Corollary 25.** *Any zero-error* $O\left(\frac{n}{s}\right)$*-pseudo-deterministic algorithm for* Find-Duplicate *must use* $\Omega\left(\frac{s}{\log n}\right)$ *space.*

Below we show that the above lower bound is tight up to log factors.

▶ **Theorem 26.** *For all* s*, there exists a zero-error randomized algorithm for* Find-Duplicate *using* $\widetilde{O}(s)$ *space (where* $\widetilde{O}$ *hides factors polylogarithmic in* n*) that is* $O\left(\frac{n}{s}\right)$*-concentrated.*

Proof. Define the following algorithm A for Find-Duplicate: pick a random number i in $[3n/2]$, then remember the i^{th} element a of the stream, and see if a appears again later in the stream. If it does, return x. Otherwise return $\perp$.

The $O\left(\frac{n}{s}\right)$-concentrated algorithm algorithm is as follows: Run $s\log n$ copies of Algorithm A independently (in parallel), and then output the minimum of the outputs.

We are left to show that this algorithm is indeed $O\left(\frac{n}{s}\right)$-concentrated.

Define f to be a function where $f(i)$ is the total number of times which i shows up in the stream, and define $g(i) = \max((f(i)-1), 0)$. Note that then, the probability that i is outputted by algorithm A is $g(i)/(3n/2)$, since i will be outputted if A chooses to remember one of the first $i-1$.

Consider the smallest a such that $\sum_{i=1}^{a} g(i) \ge n/(2s)$. We will show that the probability that the output is less than a with high probability. It will follow that the algorithm is s-concentrated, since of the $a-1$ smallest elements, at most $\sum_{i=1}^{a-1} g(i)$ outputs are possible (since if $g(i)=0$, then i is not a possible output). So, we will see that with high probability, one of at most $\sum_{i=1}^{a-1} g(i) + 1 \le n/(2s)$ outputs (namely, the valid outputs less than or equal to a) will be outputted with high probability. And hence, at least one of them will be outputted with probability at least $\frac{s}{n}$.

The probability that the output is at most a in a single run of algorithm A is $\frac{3n}{2}\sum_{i=1}^{a} g(i)$ $\ge 3/(4s)$. So, the probability that in $s\log n$ runs of algorithm, in at least one of them an element which is at most a is outputted is $1-(1-\frac{3}{4s})^{s\log n}$, which is polynomially small in n. Hence, with high probability an element which is at most a (and there are $n/(2s)$ valid outputs less than a) will be outputted. ◀

4.1 Getting Rid of the Zero Error Requirement

A downside of Theorem 23 is that it shows a lower bound only for zero-error algorithms. In this section, we strengthen the theorem by getting rid of that requirement:

▶ **Theorem 27.** *Every randomized algorithm for* Find-Duplicate *that is* $\frac{n}{s}$*-concentrated and errs with probability at most* $\frac{1}{n^2}$ *must use* $\tilde{\Omega}\left(s^{1-\epsilon}\right)$ *space (for all* $\epsilon > 0$*).*

Proof overview

We begin by outlining why the approach of Theorem 23 does not work without the zero-error requirement. Recall that the idea in the proof was to have Alice send many messages (for One-Way-Find-Duplicate) to Bob, and Bob simulates the One-Way-Find-Duplicate algorithm (using simulated inputs he creates for himself) using these messages to find elements in Alice's input.

The problem is that the elements we end up removing from Bob's simulated input[1] depend on Alice's messages, and therefore we can't use a union bound to bound the probability that the protocol failed for a certain simulated input. So, we want the elements we remove from Bob's fake input not to depend on the inputs Alice sent. One idea to achieve this is to have Alice send a bunch of messages (for finding a shared element), and then Bob will remove the element that gets output the largest number of times (by simulating the protocol with each of the many messages Alice sent). The issue with this is that if the two most common outputs have very similar probability, the outputted element depends not only on Alice's input, but also on the randomness she uses when choosing what messages to send to Bob. This makes it again not possible to use a union bound.

There are two new ideas to fix this issue. The first is to use the following "Threshold" technique: Bob will pick a random "threshold" T between $ks/(2n)$ and $ks/(4n)$ (where we wish to show a lower bound on n/s-concentrated algorithms, and k is the total number of messages Alice sends to Bob). He simulates the algorithm for One-Way-Find-Duplicate with all k messages Alice sent him, and gets a list L of k outputs. Then, he will consider the "real" output to be the lexicographically first output $y \in L$ where there are more than T copies of y in the list L (note that since the algorithm is n/s-concentrated, its very unlikely for no such element to exist).

Now, it follows that with high probability, the shared element does not really depend on the messages. This is because with all but probability approximately $1/\sqrt{ks/n}$, the threshold is far (more than $\sqrt{ks/n}\log^2 ks/n$ away) from the the frequency of every element in L. We note that we pick $\sqrt{ks/n}\log^2 ks/n$ since from noise we would expect to have the frequencies of elements in L change by up to $\sqrt{ks/n}\log^2 ks/n$, depending on the randomness of A. We want the threshold to be further than that from the expected frequencies, so that with high probability there will be no element which sometimes has frequency more than T and sometimes has frequency less than T, depending on Alice's messages (recall that the goal is to make the outputs depend as little as possible on Alice's messages, but to only depend on shared randomness and on Alice's input).

This is still not enough for us: we still cannot use a union bound, as $1/\sqrt{ks/n}$ fraction of the time Bob's output will depends on Alice's message (and not just her input). The next idea resolves this. What Alice will do is send $n/\sqrt{ks/n}$ additional pieces of information: telling Bob where the chosen thresholds are bad, and what threshold to use instead. We assume that we have shared randomness so Alice knows all of the thresholds that will be chosen by Bob (note heavy-recovery is hard, even in the presence of shared randomness, so the lower bound is sufficient with shared randomness). Now, Alice can tell for which executions there the threshold chosen will be too close to the likelihood of an element. So, Alice will send approximately $n/\sqrt{ks/n}$ additional pieces of information: telling Bob where the chosen thresholds are bad, and what threshold to use instead. By doing so, Alice has guaranteed that a path independent of her messages will be taken.

[1] recall that Bob simulates an input to the One-Way-Find-Duplicate problem, and then he repeatedly finds elements he shares with Alice, removes them from the "fake" input, and reconstructs a large fraction Alice's inputs

To recap, idea 1 is to use the threshold technique so that with probability $1 - 1/\sqrt{ks/n}$ what Bob does doesn't depend on Alice's messages (only on her input). Idea 2 is to have Alice tell Bob where these $1/\sqrt{ks/n}$ bad situations are, and how to fix them.

The total amount of information Alice sends (ignoring logs) is $\tilde{\Theta}(kb + n/\sqrt{ks/n})$, (where b is the message size we are assuming exists for pseudo-deterministically finding a shared element, and k is the number of messages Alice sends). The factor $n/\sqrt{ks/n}$ follows since $1/\sqrt{ks/n}$ of the times, short messages will be sent to Bob due to a different threshold. A threshold requires $\log n$ bits to describe, which can be dropped since we are ignoring log factors. Setting $n/s \ll k \ll n/b$, we conclude that Alice sends a total of $\tilde{o}(n)$ bits. This establishes a contradiction, since we need $\tilde{\Theta}(n)$ bits to solve One-Way-Partial-Recovery. So, whenever $s = \tilde{\omega}(b)$, we can pick a k such that we get a contradiction.

Proof. Below we write the full reduction written out as an algorithm for One-Way-Find-Duplicate.

- Alice Creates $k = n/\sqrt{sb}$ messages for One-Way-Find-Duplicate, and sends them to Bob (Call these messages of type A).
- Additionally, Alice looks at the thresholds in the shared randomness. every time there is a threshold that is close (within $\sqrt{ks/n}\log^2 ks/n$) of the expected number of times a certain y will be outputted on the corresponding input (that is, for each fake input Bob will try, Alice checks if the probability of outputting some y is close to T – to be precise, say she checks if its probability of being outputted, assuming a randomly chosen message by Alice, is close to T), she sends a message to Bob informing him about the bad threshold, and suggests a good threshold to be used instead (call these messages of type B). Notice that these messages do not depend on the messages of type A that Alice sends, and that each such message is of size $O(\log n)$.
- Bob sets B to be the simulated input $\{1, ..., n\}$
- Bob uses each of the messages of type A that Alice sent, along with B, to construct a list of outputs.
- Bob looks at the shared randomness to find a threshold T (if Alice has informed him it is a bad threshold, use the threshold Alice suggests instead), and consider the lexicographically minimal output y that is contained in the multiset more than T times.
- Bob removes y from the fake input and repeat the last three steps of the algorithm (this time using a new threshold).

▷ Claim 28. The above protocol solves One-Way-Partial-Recoverywith high probability using $\tilde{o}(n)$ bits.

Proof. First we show that the total number of bits communicated is $\tilde{o}(n)$. Notice that the total number of messages of type A that are sent is $n/\sqrt{sb}$. We assume that each of these is of size at most b, giving us a total of $n\sqrt{b}/\sqrt{s}$ bits sent in messages of type A. Under the assumption that $b = \tilde{o}(n)$, we see that this is $\tilde{o}(n)$ total bits for messages of type A.

We now count the total number of bits communicated in messages of type B. Each message of type B is of size $O(\log n)$ (it is describing a single element, and a number corresponding to which execution the message is relevant for, each requiring $O(\log n)$ bits). So, we wish to show that with high probability the total number of messages of type B is $\tilde{o}(n)$. The total number of messages of type B that will be sent is $O(\frac{n}{\sqrt{ks/n}})$, since for every input, the probability that the randomly chosen threshold (which is sampled using public randomness) is more than $\sqrt{ks/n}\log^2 ks/n$ away from the frequency of every output is $O(\frac{1}{\sqrt{ks/n}})$. Note that $\frac{n}{\sqrt{ks/n}} = \tilde{o}(n)$ since $ks = n\sqrt{\frac{s}{b}}$, and we assume $b = \tilde{o}(s)$.

We are now left to show the protocol correctly solves One-Way-Partial-Recovery with high probability. We will first show that, after fixing Alice's input and the public randomness, with high probability there will be a single sequence of inputs that Bob will try that will occur with high probability (that is, there is a sequence of y's that Bob goes through with high probability). To do this, consider a certain input that Bob tries. We will bound the probability that there are two values y and y' such that both y and y' have probability at least $\frac{1}{n}$ of being outputted. Suppose there exists two such y and y' that means that at least one of them (say y, without loss of generality) has to be the output of more than T of the k executions with probability more than $\frac{1}{n}$, but less than $\frac{n-1}{n}$. Additionally, we know that the expected number of times that y will be outputted of the k times is more than $\sqrt{ks/n}\log^2 ks/n$ away from T (otherwise Alice will pick a different value of T such that this will be true, and send that value to Bob in a message of type B). However, the probability of being more than $\sqrt{ks/n}\log^2 ks/n = \Theta((\frac{s}{b})^{1/4}\log^2 s/b) = \Theta(n^{\varepsilon/4}\log^2 n)$ away from the expectation, by a Chernoff bound, is (asymptotically) less than $\frac{1}{n}$.

Notice also, that by the assumption that the algorithm in n/s-concentrated, there will always be an output $y_{\max}$ which is expected to appear at least $\frac{s}{n}$ of the time. Also, since the threshold T is at most $ks/2n$, the probability that $y_{\max}$ appeared fewer than T times is exponentially low in $ks/n = \sqrt{s/b} = \tilde{\Theta}(n^{\epsilon/2})$, and so with high probability there will always exist a y which was outputted on more than T of the executions, so in the second to last step, the multiset will always have an element that appears at least T_1 times.

Hence, by a union bound over all inputs that Bob tries, with high probability there will be a single sequence of inputs which Bob goes through (which depends only only on the public thresholds and Alice's input).

We will show that each y generated by Bob is an element in Alice's input with high probability. Notice that the y that Bob picks has appeared more than T times out of k, where T is at least $ks/(4n)$. If y is not a valid output then its probability of being outputted is $\frac{1}{n^2}$. The probability it is outputted at least once is at most $\frac{k}{n^2} \leq \frac{1}{n}$. Taking a union bound over the inputs that Bob tries (of which there are $n/10$), we get that the probability that there is an invalid y at any point is at most $1/10$. So, with probability $9/10$, no invalid y is ever outputted. ◁

◀

5 Entropy lower bounds for finding a support element

Consider the turnstile model of streaming, where a vector $z \in \mathbb{R}^n$ starts out as 0 and receives updates of the form "increment z_i by 1" or "decrement z_i by 1", and the goal of outputting a nonzero coordinate of z. This is a well studied problem and a common randomized algorithm to solve this problem in a small amount of space is known as ℓ_0 sampling [8]. ℓ_0 sampling uses polylogarithmic space and outputs a uniformly random coordinate from the support of z. A natural question one could ask is whether the output of any low space randomized algorithm is necessarily close to uniform, i.e., has high entropy. We answer this affirmatively and show a nearly tight tradeoff between the space needed to solve this problem and the entropy of the output of a randomized algorithm under the assumption that the algorithm is not allowed to output anything outside the support[2]

[2] We note that using similar ideas to those in Subsection 4.1, the zero error requirement could be removed. We omit this adaptation since it is very similar to that of Subsection 4.1.

▶ **Theorem 29.** *Every zero-error randomized algorithm for* Find-Support-Elem *that is* $\frac{n}{s}$*-concentrated must use* $\Omega\left(\frac{s}{\log n}\right)$ *space.*

We only provide a sketch of the proof and omit details since they are nearly identical to the proof of Theorem 23.

Proof Sketch. Let $\mathcal{A}$ be such an algorithm that uses T space. Just like the proof of Theorem 23, the way we show this lower bound is by illustrating that $\mathcal{A}$ can be used to obtain an $O\left(\frac{Tn\log n}{s}\right)$-communication protocol for One-Way-Partial-Recovery, which combined with Claim 21 yields the desired result.

For every element a in Alice's input set A, she streams "increment z_a by 1" and runs $\Theta\left(\frac{n}{s}\log n\right)$ independent copies of $\mathcal{A}$ on the input. She then sends the states of each these independent runs of $\mathcal{A}$ to Bob, which is at most $\frac{Tn\log n}{s}$ bits, to Bob. Bob maintains a set of states $\mathcal{M}$, initially filled with all of Alice's messages. While he has not yet recovered $n/10$ elements, Bob picks a message $M \in \mathcal{M}$ and recovers x in A using algorithm $\mathcal{A}$. And for each $M \in \mathcal{M}$, Bob resumes $\mathcal{A}$ on state M and streams "decrement z_x by 1" and adds the new state to $\mathcal{M}$, and deletes M from $\mathcal{M}$.

The proof of correctness for why Bob indeed eventually recovers $n/10$ elements of A is identical to that in the proof of Theorem 23, thus giving a protocol for One-Way-Partial-Recovery and proving the statement. ◀

We can immediately conclude the following.

▶ **Corollary 30.** *Any zero-error* $\log\left(\frac{n}{s}\right)$*-entropy randomized algorithm for* Find-Support-Elem *must use* $\Omega\left(\frac{s}{\log n}\right)$ *space.*

▶ **Corollary 31.** *Any zero-error* $O\left(\frac{n}{s}\right)$*-pseudo-deterministic algorithm for* Find-Support-Elem *must use* $\Omega\left(\frac{s}{\log n}\right)$ *space.*

This lower bound is also tight up to polylogarithmic factors due to an algorithm nearly identical to the one from Theorem 26. In particular, we have:

▶ **Theorem 32.** *For all* s*, there exists a zero-error randomized algorithm for* Find-Duplicate *using* $O(s)$ *space that is* $O\left(\frac{n}{s}\right)$*-concentrated.*

6 Space complexity of pseudo-deterministic ℓ_2-norm estimation

In this section, we once again consider the pseudo-deterministic complexity of ℓ_2 norm estimation in the *sketching model*. The algorithmic question here is to design a distribution $\mathcal{D}$ over $s \times n$ matrices along with a function $f : \mathbb{R}^s \to \mathbb{R}$ so that for any $x \in \mathbb{R}^n$:

$$\Pr_{\boldsymbol{S}\sim\mathcal{D}}[f(\boldsymbol{S}x) \notin \left[\frac{1}{\alpha}\|x\|_2, \alpha\|x\|_2\right] \leq \frac{1}{\text{poly}(n)}.$$

Further, we want $f(\boldsymbol{S}x)$ to be a pseudo-deterministic function; i.e., we want $f(\boldsymbol{S}x)$ to be a unique number with high probability.

▶ **Theorem 33.** *The pseudo-deterministic sketching complexity of* ℓ_2 *norm estimation is* $\Omega(n)$*.*

The following query problem is key to our lower bound.

▶ **Definition 34** (ℓ_2 adaptive attack)**.** *Let $\alpha > 0$ be some constant. Let S be an $s \times n$ matrix with real-valued entries and $f : \mathbb{R}^s \to \mathbb{R}$ be some function. Now, consider the query model where an algorithm is allowed to specify a vector $x \in \mathbb{R}^n$ as a query and is given $f(Sx)$ as a response. The goal of the algorithm is to output y such that*

$$f(Sy) \notin \left[\frac{1}{\alpha}\|y\|_2, \alpha\|y\|_2\right]$$

*in as few queries as possible. We call this algorithmic problem the ℓ_2-*adaptive attack problem.

We use a theorem on adaptive attacks on ℓ_2 sketches proved in [17].

▶ **Theorem 35.** *There is a $\mathrm{poly}(n)$-query protocol to solve the ℓ_2 adaptive attack problem with probability at least $9/10$, i.e., the problem in Definition 34 when $s = o(n)$.*

Proof of Theorem 33. Suppose $\mathcal{D}$ is a distribution over $s \times n$ sketching matrices and f is a function mapping $\mathbb{R}^s$ to $\mathbb{R}$ with the property that the pair $(\mathcal{D}, f)$ gives a pseudo-deterministic sketching algorithm for ℓ_2 norm estimation. Henceforth, we use $\boldsymbol{S}$ to denote a random matrix sampled from $\mathcal{D}$. Then there is a function $g : \mathbb{R}^n \to \mathbb{R}$ such that:

1. g is an α-approximation of the ℓ_2 norm.
2. On every input x, $f(\boldsymbol{S}x) = g(x)$ with probability at least $1 - \frac{1}{n^c}$ for some constant c.

We will show that s must be $\Omega(n)$ by deducing a contradiction when $k = o(n)$. Let r be a parameter to be chosen later. Let $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(r)}$ be the (random) sequence of vectors in $\mathbb{R}^n$ obtained by the adaptive query protocol from Theorem 35 based on responses $g(\boldsymbol{x}^{(0)}), \ldots, g(\boldsymbol{x}^{(r)})$ where $r = \mathrm{poly}(n)$, and let $\boldsymbol{y}$ be the (random) output of the protocol. Note that the guarantee that $r = \mathrm{poly}(n)$ hinges on assuming $s = o(n)$. From the guarantees of Theorem 35, for any fixed matrix B and function h such that $h(B\boldsymbol{x}^{(i)}) = g(\boldsymbol{x}^{(i)})$ for all i, it is true with probability at least $9/10$ that $h(B\boldsymbol{y}) \neq g(\boldsymbol{y})$. On the other hand, for any sequence of $r + 2$ fixed vectors $v_0, \ldots, v_{r+1}$, $f(\boldsymbol{S}v_i) = g(v_i)$ for all i with probability at least $1 - \frac{1}{\mathrm{poly}(n)}$. Call the event $\{f(\boldsymbol{S}\boldsymbol{x}^{(0)}) = g(\boldsymbol{x}^{(0)}), \ldots, f(\boldsymbol{S}\boldsymbol{x}^{(r)}) = g(\boldsymbol{x}^{(r)}), f(\boldsymbol{S}\boldsymbol{y}) = g(\boldsymbol{y})\}$ as $\mathcal{E}$. Let p_S be the probability density function of $\boldsymbol{S}$ and let p_T be the probability density function of $(\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(r)}, \boldsymbol{y})$. This results in the following two estimates of $\Pr[\mathcal{E}]$.

On the one hand,

$$\begin{aligned}\Pr[\mathcal{E}] &= \int_{\boldsymbol{S}} \Pr[\mathcal{E}|\boldsymbol{S}] p_S(\boldsymbol{S}) \\ &\leq \int_{\boldsymbol{S}} \frac{1}{10} p_S(\boldsymbol{S}) \\ &= \frac{1}{10},\end{aligned}$$

and on the other hand,

$$\begin{aligned}\Pr[\mathcal{E}] &= \int_{\boldsymbol{x}^{(0)},\ldots,\boldsymbol{x}^{(r)},\boldsymbol{y}} \Pr[\mathcal{E}|\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(r)}, \boldsymbol{y}] p_T(\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(r)}, \boldsymbol{y}) \\ &\geq \int_{\boldsymbol{x}^{(0)},\ldots,\boldsymbol{x}^{(r)},\boldsymbol{y}} \left(1 - \frac{1}{\mathrm{poly}(n)}\right) p_T(\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(r)}, \boldsymbol{y}) \\ &= 1 - \frac{1}{\mathrm{poly}(n)}.\end{aligned}$$

The contradiction arises since $\Pr[\mathcal{E}]$ cannot simultaneously be at least $1 - \frac{1}{\mathrm{poly}(n)}$ and at most $\frac{1}{10}$, and hence s cannot be $o(n)$. ◀

▶ **Corollary 36.** *For any constant $\delta > 0$, any $(2-\delta)$-concentrated sketching algorithm that where the sketching matrix is $s \times n$ can be turned into a pseudo-deterministic one by running $\log n$ independent copies of the sketch and outputting the majority answer. Thus, as an upshot of Theorem 33 we obtain a lower bound of $\Omega\left(\frac{n}{\log n}\right)$ on $(2-\delta)$-concentrated algorithms for pseudo-deterministic ℓ_2-norm estimation in the sketching model.*

In contrast to Corollary 36 which says that $(2-\delta)$-concentrated algorithms for ℓ_2 estimation in the sketching model need near linear dimension, we show that there is an $O(\text{poly}\log n)$-dimension $(2+\delta)$-concentrated sketching algorithm to solve the problem, thus exhibiting a "phase transition".

▶ **Theorem 37.** *There is a distribution $\mathcal{D}$ over $s \times n$ matrices and a function $f : \mathbb{R}^s \to \mathbb{R}$ when $s = O(\text{poly}\log n)$ For every constant $\delta > 0$, there is an $O(\text{poly}(\log n, \log m))$-space $(2+\delta)$-concentrated sketching algorithm for ℓ_2-norm estimation.*

Proof. Let the true ℓ_2 norm of the input vector be r. Run the classic sketching algorithm of [2] for randomized ℓ_2 norm estimation with error $\min\{1/2^{20}, \varepsilon^4\}$ and failure probability $\frac{1}{\text{poly}(n)}$ where $(1+\varepsilon)$ is the desired approximation ratio. This uses a sketch of dimension $O(\text{poly}\log n)$. Now, we describe the function f we use. Take the output of the sketching algorithm of [2] and return the number obtained by zeroing out all its bits beyond the first $\max\{2\log\left(\frac{1}{\varepsilon}\right), 5\}$ significant bits.[3] First, the outputted number is a $(1+\varepsilon)$ approximation. Further, for each input, the output is one of two candidates with probability $1 - \frac{1}{\text{poly}(n)} > 1-\delta$ for every constant δ. This is because [2] produces a $(1+\varepsilon^4)$-approximation to r, and there are only two candidates for the $2\log\left(\frac{1}{\varepsilon}\right)$ most significant bits of any real number that lies in an interval $[(1-\varepsilon^4)r, (1+\varepsilon^4)r]$. ◀

7 Pseudo-deterministic Upper Bounds

7.1 Finding a nonzero row

Given updates to an $n \times d$ matrix A (where we assume $d \leq n$) that is initially 0 in a turnstile stream such that all entries of A are always in range $[-n^3, n^3]$, the problem Find-Nonzero-Row is to either output an index i such that the ith row of A is nonzero, or output `none` if A is the zero matrix.

▶ **Theorem 38.** *The randomized space complexity for* Find-Nonzero-Row *is $\widetilde{\Theta}(1)$, the pseudo-deterministic space complexity for* Find-Nonzero-Row *is $\widetilde{\Theta}(n)$, and the deterministic space complexity for* Find-Nonzero-Row *is $\widetilde{\Theta}(nd)$.*

Proof. We first will show a randomized $\widetilde{\Theta}(1)$ space algorithm for the problem, then we will show pseudo-deterministic upper and lower bounds, and then show the deterministic lower bound.

Randomized algorithm for Find-Nonzero-Row. A randomized algorithm for this problem is given below. Note that the version of the algorithm as stated below does not have the desirable $\widetilde{O}(1)$ space guarantee, but we will show how to use a pseudorandom generator of Nisan [29] to convert the below algorithm to one that uses low space.

[3] The parameters $1/2^{20}, 5$ and ε^4 are chosen purely for *safety* reasons.

1. Sample a random d-dimensional vector $\boldsymbol{x}$ where each entry is an independently drawn integer in $[-n^3, n^3]$ and store it.
2. Simulate a turnstile stream which maintains $A\boldsymbol{x}$. In particular, consider the n-dimensional vector y, which is initially 0, and for each update to A of the form "add Δ to A_{ij}", add $\Delta \boldsymbol{x}_j$ to y_i. We run an ℓ_0-sampling algorithm [8] on this simulated stream updating y, and return the output of the ℓ_0-sampler, which is close in total variation distance to a uniformly random element in the support of y.

In the above algorithm, step 1 is not low-space as stated. Before we give a way to perform step 1 in $\widetilde{O}(1)$ space, we prove the correctness of the above algorithm. Suppose A_i is a nonzero row of A, then let j be an index where A_i is nonzero. Suppose all coordinates of $\boldsymbol{x}$ except for the j-th coordinate have been sampled, there is at most one value C for $\boldsymbol{x}_j$ for which $\langle A_i, \boldsymbol{x}\rangle$ is 0, and there is at most a $1/n^3$ probability that $\boldsymbol{x}_j$ equals C, which means if i is a nonzero row, then $(A\boldsymbol{x})_i$ is nonzero except with probability at most $1/n^3$. In fact, by taking a union bound over all nonzero rows we can conclude that the set of nonzero rows and the set of nonzero indices of $A\boldsymbol{x}$ are exactly the same, except with probability bounded by $1/n^2$.

Now we turn our attention to implementing step 1 in low space. Towards doing so we use Nisan's pseudorandom generator for space bounded computation in a very similar manner to [19].

Instead of sampling $3d \log n + 1$ bits to store $\boldsymbol{x}$, we sample and store a uniformly random seed $\boldsymbol{w}$ of length $O(\text{poly}\log(n, d))$ and add $\Delta G(\boldsymbol{w})_j$ to y_i when an update "add Δ to A_{ij}" is received, where G is the function from Theorem 12 that maps the random seed to a sequence $3d \log n + 1$ bits. To prove the algorithm is still correct if we use the pseudorandom vector $G(\boldsymbol{w})$ instead of the uniformly random vector $\boldsymbol{x}$, we must show that when A_i is nonzero, then $\langle A_i, G(\boldsymbol{w})\rangle$ is nonzero with probability at least $1 - O(1/n^3)$. Towards this, for a fixed d-dimensional vector q, consider the following finite state machine. The states are labeled by pairs (i, a) where i is in $\{0, 1, \ldots, d\}$ and a is in $[-n^6 d, n^6 d]$. The FSM takes a d-dimensional vector r as input, starts at state $(0, 0)$, and transitions from state (i, a) to $(i+1, a+q_{i+1} \cdot r_{i+1})$ until it reaches a state (d, ℓ). The FSM then outputs ℓ. This establishes that for a fixed q, the function $f(x) \coloneqq \langle q, x\rangle$ is computable by an FSM on $\text{poly}(d, n)$ states, and hence from Theorem 12, $f(\boldsymbol{x})$ and $f(G(\boldsymbol{w}))$ are $1/dn^6$ close in total variation distance, which means when A_i is nonzero, then $\langle A_i, G(\boldsymbol{w})\rangle$ is nonzero except with probability bounded by $O(1/n^3)$.

A pseudo-deterministic algorithm and lower bound for Find-Nonzero-Row. The pseudo-deterministic algorithm is very similar to the randomized algorithm from the previous section.

1. Sample a random d-dimensional vector $\boldsymbol{x}$ where each entry is an independently drawn integer in $[-n^3, n^3]$. Store $\boldsymbol{x}$ and maintain $A\boldsymbol{x}$.
2. Output the smallest index i such that $(A\boldsymbol{x})_i$ is nonzero.

Storing $\boldsymbol{x}$ takes $O(d \log n)$ space, and maintaining $A\boldsymbol{x}$ takes $O(n \log n)$ space. Recall from the discussion surrounding the randomized algorithm that the set of nonzero indices of $A\boldsymbol{x}$ and the set of nonzero rows were equal with probability $1 - 1/n^2$, which establishes correctness of the above pseudo-deterministic algorithm. The space complexity is thus $O((d + n) \log n)$, which is equal to $O(n \log n)$ from the assumption that $d \leq n$.

A pseudo-deterministic lower bound of $\widetilde{\Omega}(n)$ follows immediately from Corollary 31 since Find-Nonzero-Row specialized to the $d = 1$ case is the same as Find-Support-Elem.

Lower Bound for deterministic algorithms. An $\Omega(nd \log n)$ bit space lower bound for deterministic algorithms follows from a reduction to the communication complexity problem of EQUALITY. Alice and Bob are each given $nd \log n$ bit strings as input, which they interpret as $n \times d$ matrices, A and B respectively, where each entry is a chunk of length $\log n$. Suppose a deterministic algorithm $\mathcal{A}$ takes S bits of space to solve this problem. We will show that this can be converted to a S-bit communication protocol to solve EQUALITY. Alice runs $\mathcal{A}$ on a turnstile stream updating matrix X initialized at 0 by adding A_{ij} to X_{ij} for all (i, j) in $[n] \times [d]$. Alice then sends the S bits corresponding to the state of the algorithm to Bob and he continues running $\mathcal{A}$ on the updates "add $-B_{ij}$ to X_{ij}". $\mathcal{A}$ outputs `none` if and only if $A = B$ and thus Bob outputs the answer to EQUALITY depending on the output of $\mathcal{A}$. Due to a communication complexity lower bound of $\Omega(nd \log n)$ on EQUALITY, S must be $\Omega(nd \log n)$. ◀

7.2 Point Query Estimation and Inner Product Estimation

In this section, we give pseudo-deterministic algorithms that beat the deterministic lower bounds for two closely related streaming problems – point query estimation and inner product estimation.

Point Query Estimation

Given a parameter ε and a stream of m elements where each element comes from a universe $[n]$, followed by a query $i \in [n]$, output f_i' such that $|f_i - f_i'| \leq \varepsilon m$ where f_i is the frequency of element i in the stream.

Inner Product Estimation

Given a parameter ε and a stream of m updates to (initially 0-valued) n-dimensional vectors x and y in an insertion-only stream[4], output estimate e satisfying $|e - \langle x, y \rangle| < \varepsilon \cdot \|x\|_1 \cdot \|y\|_1$. In the above problems, we will be interested in the regime where $m \ll n$.

Our main result regarding a pseudo-deterministic algorithm for point query estimation is:

▶ **Theorem 39.** *There is an $O\left(\frac{\log m}{\varepsilon} + \log n\right)$-space pseudo-deterministic algorithm $\mathcal{A}$ for point query estimation with the following precise guarantees. For every sequence $s_1, \ldots, s_m$ in $[n]^m$, there is a sequence $f_1', \ldots, f_n'$ such that*

1. *For all i, $|f_i' - f_i| \leq \varepsilon m$ where f_i is the frequency of i in the stream.*
2. *Except with probability $1/m$, for all $i \in [n]$ $\mathcal{A}$ outputs f_i' on query i.*

We remark that the deterministic complexity of the problem is $\Omega(\frac{\log n}{\varepsilon}$ (see Theorem 44). Towards establishing Theorem 39, we recall two facts.

▶ **Theorem 40** (Misra–Gries algorithm [25])**.** *Given a parameter ε and a length-m stream of elements in $\{1, \ldots, d\}$, there is a deterministic $O\left(\frac{\log d + \log m}{\varepsilon}\right)$-space algorithm that given any query $s \in [d]$, outputs f_s' such that $|f_s' - f_s| \leq \varepsilon m$ where f_s is the number of occurrences of s in the stream. An additional guarantee that the algorithm satisfies is the following, which we call* permutation invariance*. Consider the stream*

$$s_1, s_2, \ldots, s_m$$

[4] A stream where only increments by positive numbers are promised.

and for any permutation $\pi : [d] \to [d]$, consider the stream

$$\pi(s_1), \pi(s_2), \ldots, \pi(s_m).$$

When the algorithm is given the first stream as input, let f'_s denote its output on query s, and when the algorithm is given the second stream as input, let $g'_{\pi(s)}$ denote its output on query $\pi(s)$. The algorithm has the guarantee that $f'_s = g'_{\pi(s)}$.

▶ **Theorem 41** (Pairwise independent hashing, [32, Corollary 3.34]). *Assume $d \ll n$. There is a pairwise independent hash function $h : [n] \to [d]$, which can be sampled using $O(\log n)$ random bits and also can be stored in $O(\log n)$ bits.*

Proof of Theorem 39. The algorithm is as follows.

- Sample a random pairwise independent hash function $h : [n] \to [m^3]$, which can be sampled and stored in $O(\log n)$ bits.
- Run the Misra–Gries algorithm with the following simulated stream as input: for each s streamed as input, stream $h(s)$ to the simulation.
- Given any query s, perform query $h(s)$ to the Misra–Gries algorithm running on the simulated stream, and return its output.

Let S be the collection of elements of $[n]$ that occur in the input stream $s_1, \ldots, s_m$. Assuming h maps S into $[m^3]$ without any collisions[5], it follows from the permutation invariance property of the Misra–Gries algorithm from Theorem 40 the output of the above algorithm on any query q is equal to $F(s_1, \ldots, s_m, q)$ for a *fixed* function F. Thus if we show that h indeed maps S into $[m^3]$ injectively pseudo-determinism of the given algorithm would follow.

Given $i, j \in S$, due to pairwise independence of h, the probability that $h(i) = h(j)$ is equal to $1/m^3$. A union bound over all pairs of elements in S tells us that h is collision-free except with probability at most $1/m$, which implies that the above algorithm is indeed pseudo-deterministic. ◀

▶ **Theorem 42.** *There is a (weakly) pseudo-deterministic algorithm for inner product estimation that uses $O\left(\frac{\log m}{\varepsilon} + \log n\right)$ space.*

The algorithm for inner product estimation is based on point query estimation, and towards stating the algorithm we first state a known result that helps relate the two problems.

▶ **Lemma 43** (Easily extracted from the proof of [28, Theorem 1]). *Let x, y, x', y' be vectors such that $\|x - x'\|_\infty \le \varepsilon \|x\|_1$ and $\|y - y'\|_\infty \le \varepsilon \|y\|_1$. Now, let x'' (and respectively y'') denote x' with everything except the maximum $1/\varepsilon$ entries zeroed out. Then the following holds:*

$$|\langle x'', y'' \rangle - \langle x, y \rangle| \le \varepsilon \cdot \|x\|_1 \cdot \|y\|_1.$$

Proof of Theorem 42. Given a stream of updates to x and y, run two instances of the point query estimation algorithm from Theorem 39 – one for updates to x and one for updates to y. There are x' and y' that only depend on the stream such that

$$\|x - x'\|_\infty \le \varepsilon \cdot \|x\|_1 \quad \text{and} \quad \|y - y'\|_\infty \le \varepsilon \cdot \|y\|_1$$

and except with probability $O(1/m)$ both point query algorithms respond to any query i with x'_i (and y'_i respectively). Maintaining these two instances takes $O\left(\frac{\log m}{\varepsilon} + \log n\right)$ space.

[5] I.e. the restriction of h to domain S is an injective function.

Next, enumerate over elements of $[n]$ and for each $i \in [n]$ query both instances with i, and store the running max-$1/\varepsilon$ answers to queries to each instance along with the hashed identities of the indices of entries that are part of the running max. Storing the running max takes $O\left(\frac{\log m}{\varepsilon}\right)$ space, and storing a counter to enumerate over $[n]$ takes $\log n$ space. Thus, at the end of this routine, except with probability $O(1/m)$ our two lists are equal to $(x'_{i_1}, h(i_1)), \ldots, (x'_{i_{1/\varepsilon}}, h(i_{1/\varepsilon}))$ and $(y'_{j_1}, h(j_1)), \ldots, (y'_{j_{1/\varepsilon}}, h(j_{1/\varepsilon}))$ respectively where $x'_{i_1}, \ldots, x_{i_{1/\varepsilon}}$ are the max-$1/\varepsilon$ entries of x' and $y'_{j_1}, \ldots, y_{j_{1/\varepsilon}}$ are the max-$1/\varepsilon$ entries of y'.

Finally, if there is t, u such that $h(i_t) = h(i_u)$ or $h(j_t) = h(j_u)$, return "fail"; otherwise output

$$\sum_{\ell \in \{h(i_t)\}_{t=1,\ldots,1/\varepsilon} \cap \{h(j_t)\}_{t=1,\ldots,1/\varepsilon}} x'_\ell y'_\ell.$$

With probability at least $1 - 2/m$, the above quantity is equal to $\langle x'', y'' \rangle$ from Lemma 43, which lets us conclude via Lemma 43 that the output is within $\varepsilon \cdot \|x\|_1 \cdot \|y\|_1$ of the true inner product. ◀

Finally, we remark that the following lower bounds can be proved for deterministic algorithms.

▶ **Theorem 44.** *Any deterministic algorithm for point query estimation and inner product estimation needs* $\Omega\left(\frac{\log n}{\varepsilon}\right)$ *space.*

Proof. We prove a lower bound for point query estimation via a reduction from EQUALITYin communication complexity. Alice encodes a $\log \binom{n}{1/(3\varepsilon)}$ bit string as a subset S of $[n]$ of size $1/(3\varepsilon)$ and runs the point query streaming algorithm on the input where she streams each element of this subset $3\varepsilon m$ times. She then sends the state of the algorithm to Bob, who can query every index in the universe and learn S (the element corresponding to the query is in S if and only if the response to the query is at least $2\varepsilon \cdot m$), decode S back to a $\log \binom{n}{1/(3\varepsilon)}$ and check if it is equal to his own input. The space lower bound from the theorem statement then follows since $\log \binom{n}{1/(3\varepsilon)} = \Omega\left(\frac{\log n}{\varepsilon}\right)$.

A space lower bound for inner product estimation follows from the lower bound for point query estimation since the latter is a special case of the former when x is the vector of frequencies and y is a standard unit vector e_i corresponding to query i. ◀

7.3 Retrieving a Basis of a Row-space

We now work in a "mixed" model, where an input $n \times d$ matrix A of rank-$\leq k$ is given to us via a sequence of updates in a turnstile stream, and each entry at all times in the stream can be represented by an $O(\log n)$-bit word. During this phase, there is an upper bound T on the number of bits of space an algorithm is allowed to use. In the "second phase", we are allowed to perform arbitrary computation and the goal is to output a basis for the row-span of A

We show a lower bound on T of $\widetilde{\Omega}(nd)$ for deterministic algorithms, and a pseudo-deterministic algorithm that uses $\widetilde{O}(\text{poly}(k) \cdot d)$ space in the streaming phase.

▶ **Theorem 45.** *Any deterministic streaming algorithm for* RECOVERBASIS *needs* $\widetilde{\Omega}(nd)$ *space.*

Proof. Suppose the matrix A is 0, then the algorithm would have to output the empty set. A T space streaming algorithm for this problem could be used to solve the communication complexity problem of equality EQUALITY using T bits of communication. In particular,

Alice and Bob could encode their respective inputs x and y as matrices M_x and M_y. Alice can then run the T-space algorithm on adding M_x in a turnstile stream, and send Bob the state of the algorithm. Bob can then resume running the algorithm from Alice's state on updates that subtract M_y. If Bob outputs the empty set, then $x = y$ and Bob outputs "yes". Otherwise, Bob outputs "no". ◀

While the deterministic complexity is $\widetilde{\Omega}(nd)$, there is a pseudo-deterministic streaming algorithm which uses only $\widetilde{O}(\text{poly}(k) + k \cdot d)$ in its streaming phase:

▶ **Theorem 46.** *There is a pseudo-deterministic algorithm for* RecoverBasis *that uses* $\widetilde{O}(\text{poly}(k) + k \cdot d)$ *space in its streaming phase, where the* $\widetilde{O}(\cdot)$ *hides factors of* $\text{poly}\log n$.

Towards giving a pseudo-deterministic algorithm, we first state a result about pseudorandom matrices that is a special case of [5, Lemma 3.4].

▶ **Theorem 47.** *There is a distribution* $\mathcal{D}$ *over* $m \times n$ *matrices where* $m = O(k \log n)$ *with* ± 1 *entries such that for any* $n \times m$ *matrix* U *with orthonormal columns and* $\boldsymbol{S} \sim \mathcal{D}$*, the following holds with probability* $1 - 1/\text{poly}(n)$*:*

$$\|U^T \boldsymbol{S}\boldsymbol{S}^T U - I\|_2 \leq 1/2.$$

Further, the rows of $\boldsymbol{S}$ *are independent and each row can be generated by a* $(k + \log n)$*-wise independent hash family.*

▶ **Theorem 48** (t-wise independent hash families [32, Corollary 3.34])**.** *There is a* t*-wise independent hash family* $\mathcal{H}$ *of functions from* $[n] \to \{\pm 1\}$ *such that sampling a uniformly random* h *from* $\mathcal{H}$ *can be done using a* $\text{poly}(\log n, t)$*-length random seed, and* $h(x)$ *for any* $x \in [n]$ *can be computed in* $\text{poly}(\log n, t)$ *time and space from the random seed used to sample it.*

As a consequence we have:

▶ **Corollary 49.** *Let* A *be a* $n \times d$ *matrix of rank* k *and let* $\mathcal{D}$ *be the distribution over* $O(k \log n) \times n$ *matrices from the statement of Theorem 47. Then, for* $\boldsymbol{S} \sim \mathcal{D}$*,* $\boldsymbol{S}A$ *has rank* k *with probability* $1 - 1/\text{poly}(n)$*.*

Proof. We start by writing A in its singular value decomposition $U\Sigma V^T$. Since A has rank k, U is a $n \times k$ matrix with orthonormal columns and ΣV^T surjectively maps $\mathbb{R}^d$ to $\mathbb{R}^k$. From Theorem 47, $\boldsymbol{S}A$ is also full rank, which means the collection of vectors

$$\{\boldsymbol{S}Ax : x \in \mathbb{R}^d\} = \{\boldsymbol{S}U\Sigma V^T x : x \in \mathbb{R}^d\} = \{\boldsymbol{S}Ux : x \in \mathbb{R}^k\}$$

is a k-dimensional space, and hence $\boldsymbol{S}A$ has rank k. ◀

Proof of Theorem 46. Begin by sampling $\boldsymbol{S} \sim \mathcal{D}$ via a seed $\boldsymbol{s}$ of length $O(\text{poly}(k) \cdot \text{poly}\log(n))$ from which entries of $\boldsymbol{S}$ can be efficiently computed where $\mathcal{D}$ is the distribution over matrices given by Corollary 49, and maintain the sketch $\boldsymbol{S}A$ in the stream.

The row-span of $\boldsymbol{S}A$ is exactly the same as that of A assuming the two matrices have equal rank, which happens with probability $1 - 1/\text{poly}(n)$.

$\boldsymbol{S}A$ is an $O(k) \times d$ matrix and each entry is a signed combination of at most n entries of A and hence there is a bit complexity bound of $\widetilde{O}(kd)$ on the space used to store $\boldsymbol{S}A$.

In the second phase (i.e., after the stream is over) of the algorithm, we first find an orthonormal basis Q for the row-span of $\boldsymbol{S}A$ and compute $\widetilde{\Pi}_A = QQ^T$. And finally, use a deterministic algorithm to compute the singular value decomposition $\widetilde{U}\widetilde{\Sigma}\widetilde{V}^T$ of $\widetilde{\Pi}_A$ and output the rows of $\widetilde{V}^T$.

The row-span of $\boldsymbol{S}A$ and A are equal except with probability $1/\text{poly}(n)$; assuming this happens, $\widetilde{\Pi}_A$ is exactly equal to Π_A, the unique projection matrix onto the row-span of A. Write Π_A in its singular value decomposition $U\Sigma V^T$. If $\widetilde{\Pi}_A = \Pi_A$, $\widetilde{V}^T$ is exactly equal to V^T. Since V^T is given by a deterministic function of A, and the output of the algorithm $\widetilde{V}$ is equal to V^T with high probability, our algorithm is pseudo-deterministic. ◀

References

1 Yuqing Ai, Wei Hu, Yi Li, and David P. Woodruff. New Characterizations in Turnstile Streams with Applications. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 20:1–20:22, 2016. doi:10.4230/LIPIcs.CCC.2016.20.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

3 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Streaming algorithms from precision sampling. *arXiv preprint*, 2010. arXiv:1011.1263.

4 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

5 Kenneth L Clarkson and David P Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 205–214. ACM, 2009.

6 Peter Dixon, A Pavan, and NV Vinodchandran. On Pseudodeterministic Approximation Algorithms. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

7 Philippe Flajolet. Approximate counting: a detailed analysis. *BIT Numerical Mathematics*, 25(1):113–134, 1985.

8 Gereon Frahling, Piotr Indyk, and Christian Sohler. Sampling in dynamic data streams and applications. *International Journal of Computational Geometry & Applications*, 18(01n02):3–28, 2008.

9 Eran Gat and Shafi Goldwasser. Probabilistic Search Algorithms with Unique Answers and Their Cryptographic Applications. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, page 136, 2011.

10 Oded Goldreich. Multi-pseudodeterministic algorithms. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2019.

11 Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 127–138. ACM, 2013.

12 Shafi Goldwasser and Ofer Grossman. Perfect Bipartite Matching in Pseudo-Deterministic RNC. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 22, page 208, 2015.

13 Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-deterministic Proofs. *arXiv preprint*, 2017. arXiv:1706.04641.

14 Parikshit Gopalan and Jaikumar Radhakrishnan. Finding duplicates in a data stream. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 402–411. Society for Industrial and Applied Mathematics, 2009.

15 Ofer Grossman. Finding Primitive Roots Pseudo-Deterministically. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 22, page 207, 2015.

16 Ofer Grossman and Yang P Liu. Reproducibility and Pseudo-Determinism in Log-Space. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 606–620. SIAM, 2019.

17 Moritz Hardt and David P Woodruff. How robust are linear sketches to adaptive inputs? In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 121–130. ACM, 2013.

18 Dhiraj Holden. A Note on Unconditional Subexponential-time Pseudo-deterministic Algorithms for BPP Search Problems. *arXiv preprint*, 2017. `arXiv:1707.05808`.

19 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

20 Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 202–208. ACM, 2005.

21 Rajesh Jayaram and David P Woodruff. Perfect lp sampling in a data stream. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 544–555. IEEE, 2018.

22 Hossein Jowhari, Mert Sağlam, and Gábor Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 49–58. ACM, 2011.

23 Michael Kapralov, Jelani Nelson, Jakub Pachocki, Zhengyu Wang, David P Woodruff, and Mobin Yahyazadeh. Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 475–486. Ieee, 2017.

24 Yi Li, Huy L Nguyen, and David P Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 174–183. ACM, 2014.

25 Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

26 Morteza Monemizadeh and David P Woodruff. 1-pass relative-error L p-sampling with applications. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1143–1160. Society for Industrial and Applied Mathematics, 2010.

27 Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

28 Jelani Nelson, Huy L Nguyen, and David P Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. *Linear Algebra and its Applications*, 441:152–167, 2014.

29 Noam Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

30 Igor C Oliveira and Rahul Santhanam. Pseudodeterministic Constructions in Subexponential Time. *arXiv preprint*, 2016. `arXiv:1612.01817`.

31 Igor C Oliveira and Rahul Santhanam. Pseudo-Derandomizing Learning and Approximation. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 116. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

32 Salil P Vadhan et al. Pseudorandomness. *Foundations and Trends® in Theoretical Computer Science*, 7(1–3):1–336, 2012.