Lower Bounds for Pseudo-Deterministic Counting in a Stream

Vladimir Braverman
Rice University, Houston, TX, USA

Robert Krauthgamer
Weizmann Institute of Science, Rehovot, Israel

Aditya Krishnan
Pinecone, San Francisco, CA, USA

Shay Sapir
Weizmann Institute of Science, Rehovot, Israel

Abstract

Many streaming algorithms provide only a high-probability relative approximation. These two relaxations, of allowing approximation and randomization, seem necessary – for many streaming problems, both relaxations must be employed simultaneously, to avoid an exponentially larger (and often trivial) space complexity. A common drawback of these randomized approximate algorithms is that independent executions on the same input have different outputs, that depend on their random coins. *Pseudo-deterministic* algorithms combat this issue, and for every input, they output with high probability the same "canonical" solution.

We consider perhaps the most basic problem in data streams, of counting the number of items in a stream of length at most n. Morris's counter [CACM, 1978] is a randomized approximation algorithm for this problem that uses $O(\log\log n)$ bits of space, for every fixed approximation factor (greater than 1). Goldwasser, Grossman, Mohanty and Woodruff [ITCS 2020] asked whether pseudo-deterministic approximation algorithms can match this space complexity. Our main result answers their question negatively, and shows that such algorithms must use $\Omega(\sqrt{\log n/\log\log n})$ bits of space.

Our approach is based on a problem that we call *Shift Finding*, and may be of independent interest. In this problem, one has query access to a shifted version of a known string $F \in \{0,1\}^{3n}$, which is guaranteed to start with n zeros and end with n ones, and the goal is to find the unknown shift using a small number of queries. We provide for this problem an algorithm that uses $O(\sqrt{n})$ queries. It remains open whether poly$(\log n)$ queries suffice; if true, then our techniques immediately imply a nearly-tight $\Omega(\log n/\log\log n)$ space bound for pseudo-deterministic approximate counting.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Lower bounds and information complexity; Theory of computation → Pseudorandomness and derandomization

Keywords and phrases streaming algorithms, pseudo-deterministic, approximate counting

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.30

Category Track A: Algorithms, Complexity and Games

Related Version *arXiv Version*: `https://arxiv.org/abs/2303.16287`

Funding *Vladimir Braverman*: Work partially supported by ONR Award N00014-18-1-2364 and NSF awards 1652257, 1813487 and 2107239.
Robert Krauthgamer: Work partially supported by ONR Award N00014-18-1-2364, by a Weizmann-UK Making Connections Grant, by a Minerva Foundation grant, and the Weizmann Data Science Research Center.
Aditya Krishnan: Work partially done while the author was at Johns Hopkins University and supported by the MINDS Data Science Fellowship.

50th International Colloquium on Automata, Languages, and Programming (ICALP 2023).
Editors: Kousha Etessami, Uriel Feige, and Gabriele Puppis; Article No. 30; pp. 30:1–30:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Shay Sapir: This research was partially supported by the Israeli Council for Higher Education (CHE) via the Weizmann Data Science Research Center.

1 Introduction

Computing over data streams is a rich algorithmic area that has developed enormously, and actually started with the simple-looking problem of approximate counting [28]. Let us first recall the streaming model: The input is a stream, i.e., a sequence of items, and the goal is to compute a pre-defined function of these items, such as the number of items (or number of the distinct items), while making one sequential pass over the stream (or sometimes a few passes). Many useful functions actually depend on the items as a multiset, i.e., ignoring their order, or even only on their frequencies (like the famous ℓ_p-norm of the frequency vector). Another possible goal is to produce a sample, rather than computing a function, e.g., to produce a uniformly random item.

The primary measure of efficiency for streaming algorithms is their space complexity, and for many problems, researchers have designed space-efficient algorithms, often with space complexity that is even polylogarithmic in the input size. However, this comes at a price – these algorithms are usually randomized (and not deterministic) and/or compute an approximate solution (rather than exact one). In fact, oftentimes both relaxations are needed in order to achieve low space complexity. For example, to count the number of items in a stream of length at most n, there is a randomized approximation algorithm using $O(\log\log n)$ bits of space, but algorithms that are exact or deterministic must use $\Omega(\log n)$ bits [28]. Another example is the ℓ_2-norm of the frequency vector of items from a ground set $[d]$ (or equivalently, of a d-dimensional vector under a sequence of additive updates) – there is a randomized approximation algorithm that uses $O(\log d)$ bits of space, but algorithms that are exact or deterministic must use $\Omega(d)$ bits of space [1].

Gat and Goldwasser [9] initiated the study of *pseudo-deterministic* algorithms, which informally means that when run (again) on the same input, with high probability they produce exactly the same output. This notion combats a potential issue with randomized algorithms, that independent executions on the same input might return different outputs, depending on the algorithm's coin tosses. Many known streaming algorithms suffer from this issue, which is a serious concern for some users and applications. Pseudo-deterministic algorithms were later considered in the streaming model by Goldwasser, Grossman, Mohanty and Woodruff [16], and these are formally defined as follows.

▶ **Definition 1.1.** *A streaming algorithm A is* pseudo-deterministic *(PD) if there is a function $F(\cdot)$ defined on inputs of A (streams), such that for every stream σ,*

$$\Pr[A(\sigma) = F(\sigma)] \geq 9/10,$$

where the probability is over the random choices of the algorithm. We shall refer to F as the canonical function *of algorithm A.*[1]

We focus on *estimation problems*, which ask to approximate a numerical value, and are very popular in the streaming model. For such problems, the notion of PD relaxes the exact setting and the deterministic one, since exact algorithms have one canonical output (the

[1] The canonical function F depends on the order arrival of the stream items. In an alternative definition, the canonical function depends on the items only as a multiset, i.e., ignoring their order in the stream. These two definitions are equivalent in the setting of approximate counting, which is the focus of our work.

Table 1 Known space bounds (in bits) for 2-approximate counting in a stream of length at most n. Folklore bounds are stated without a reference.

Algorithms	Upper bound		Lower bound	
Exact or deterministic	$O(\log n)$		$\Omega(\log n)$	
Randomized and approximate	$O(\operatorname{loglog} n)$	[28]	$\Omega(\operatorname{loglog} n)$	[29]
Pseudo-deterministic	$O(\log n)$		$\Omega(\sqrt{\log n/\operatorname{loglog} n})$	[Thm. 1.2]

exact numerical value), and hence they are PD. Thus the known lower bounds for these settings do not apply for PD algorithms, and a central question, identified in [16], remains open:

Are there efficient PD streaming algorithms for estimation problems?

Currently, no lower bounds are known for natural estimation problems, although for several search problems, like reporting an element from a stream with deletions (equivalently, an index from the support of the frequency vector), it is known that lower bounds for deterministic algorithms extend to PD algorithms [16].

1.1 Main Result: Approximate Counting

Perhaps the most basic problem in the streaming model is to count the number of stream items. Exact counting, i.e., computing the number of items exactly, requires $\Theta(\log n)$ bits of space when the stream has length at most n, even for randomized algorithms with some error probability. Work by Morris [28], later refined in [8, 18, 29], showed that the number of stream items can be $(1+\epsilon)$-approximated with probability 9/10 using $O_\epsilon(\operatorname{loglog} n)$ bits of space, where $\epsilon > 0$ is arbitrary but fixed. Throughout, we refer to multiplicative approximation, and use the notations $O_c(\cdot)$ and $\Omega_c(\cdot)$ to hide factors that are polynomial in c. Morris's algorithm has found many applications, both in theory and in practice [27, 29]. An open question stated explicitly by Goldwasser, Grossman, Mohanty and Woodruff [16] is whether there is a PD algorithm for this problem using $O(\operatorname{loglog} n)$ bits of space. We answer their question negatively, by proving the following lower bound.

▶ **Theorem 1.2** (Main Result). *For every $c, n > 1$, every PD streaming algorithm that c-approximates the number of items in a stream of length at most $(c+1)n$ must use $\Omega_c(\sqrt{\log n/\operatorname{loglog} n})$ bits of space.*

To be more precise, our lower bound is actually $\Omega(\frac{\log n}{\sqrt{\log n \operatorname{loglog}(cn)}+\log c})$, which is still $\Omega(\sqrt{\frac{\log n}{\operatorname{loglog} n}})$ as long as $c < 2^{\sqrt{\log n \operatorname{loglog} n}}$. Previously, there was a large gap for this problem, between $O(\log n)$ bits (by a deterministic algorithm) and $\Omega(\operatorname{loglog} n)$ bits (from the randomized setting) [29]. See Table 1 for a summary of the known bounds.

Our proof analyzes the promise variant of c-approximate counting for streams of length at most $(c+1)n$, which we denote by $\Pi^{AC}_{c,n}$; this variant asks to distinguish whether the number of stream items is $\leq n$ or $> cn$ (see Definition 2.1). A crucial property of PD algorithms is that they have to be PD also for inputs in the range $[n+1, cn]$ (i.e., outside the promise). We rely on this property of PD algorithms to prove the following result, which immediately yields Theorem 1.2 as a corollary.

▶ **Theorem 1.3** (Main Result). *For every $c, n > 1$, every PD streaming algorithm for problem $\Pi^{AC}_{c,n}$ must use $\Omega_c(\sqrt{\log n / \operatorname{loglog} n})$ bits of space.*

Our proof of Theorem 1.3 appears in Section 4. It is based on a problem that we call Shift Finding, which may be of independent interest, as it is very natural and likely to find connections to other problems. In addition, it can potentially lead to a near-tight $\Omega(\log n / \operatorname{loglog} n)$ lower bound for PD streaming, by simply improving our algorithmic result for Shift Finding. A very recent independent work by Grossman, Gupta and Sellke [20] shows a tight $\Omega(\log n)$ bound for $\Pi^{AC}_{c,n}$, using a very different technique, which views the PD streaming algorithm as a Markov chain with a limited number of states.

1.2 Main Technique: The Shift Finding Problem

Our main result relies on *algorithms* for the shift Finding problem $\Pi^{SF}_{c,n}$, which is defined below. Let us first introduce some basic terminology. A function $F : [m] \to \{0, 1\}$ can also be viewed as a string $F \in \{0, 1\}^m$, and vice versa, and we sometimes use these interchangeably. Given $s \in [0, n]$, let the shifted version of this F be the function $F_s : x \mapsto F(s + x)$, with a properly restricted domain, see Section 2.

▶ **Definition 1.4** (Shift Finding). *Let $c, n > 1$. In problem $\Pi^{SF}_{c,n}$, the input is a string $P \in \{0, 1\}^{(c-1)n}$, and one has query access to a string F_{s^*} that is the concatenation of $n - s^*$ zeros, then P, and finally s^* ones, for an unknown $s^* \in [0, n]$. Thus, a query for $x \in [0, cn]$ returns $F_{s^*}(x)$. The goal is to output s^*.*

The measure of complexity of an algorithm for this problem is the number of queries that it makes to F_{s^*}. A randomized algorithm is required to be correct (in its output s^*) with probability 9/10.

This problem may be also of independent interest. In a different variant of shift finding, the input is a random string $c \in \{0, 1\}^n$ and a vector x that is obtained from the string c by a cyclic shift τ and some noise (random bit flips), and the goal is to compute the shift τ with high probability. This problem is related to GPS synchronization, see [23, 2] for more details. There is a sublinear time algorithm for this problem, running in time roughly $O(n^{0.641})$ [2]. One main difference is that in our Definition 1.4, one string is completely known to the algorithm, and the only concern is the number of queries to the second string.

1.2.1 Connection to PD Counting

We show that an algorithm for Shift Finding ($\Pi^{SF}_{c,n}$) implies a space lower bound for PD streaming algorithm for counting ($\Pi^{AC}_{c,n}$).

▶ **Theorem 1.5.** *Let $c, n > 1$, and suppose that the Shift Finding problem $\Pi^{SF}_{c,n}$ admits a randomized algorithm that makes at most $q = q(c, n)$ queries (possibly adaptive). Then, every PD streaming algorithm for the approximate counting problem $\Pi^{AC}_{c,n}$ must use $\Omega(\frac{\log n}{\log q})$ bits of space.*

It immediately follows that if the Shift Finding problem $\Pi^{SF}_{c,n}$ can be solved using polylog(n) queries (for fixed $c > 1$), then PD approximate counting requires $\Omega(\frac{\log n}{\operatorname{loglog} n})$ bits of space. However, our current upper bound for Shift Finding is $q = O(\sqrt{cn})$ queries (Theorem 1.8) and is not strong enough to yield a nontrivial lower bound for PD approximate counting.

Therefore, to prove our main lower bound (Theorem 1.3), we revert to a generalization of Theorem 1.5 where the Shift Finding algorithm is still given an instance of problem $\Pi^{SF}_{c,n}$ (namely, a string F and query access to F_{s^*}), but reports a small set $R \subset [0, n]$ (say of size

$|R| \leq t$) that contains the unknown shift (i.e., $s^* \in R$). This algorithm may be randomized provided that it is PD, and its canonical function maps each instance of problem $\Pi^{SF}_{c,n}$ to a set R of size t that contains s^*.

▶ **Theorem 1.6.** *Let $c, n > 1$, and suppose there is a PD algorithm Q that, given an instance of problem $\Pi^{SF}_{c,n}$, makes at most $q = q(c,n)$ queries (possibly adaptive) to F_{s^*} and its canonical function M maps the input to a set $R \subset [0,n]$ of size $t = t_c(n)$ that contains s^*. Then every PD streaming algorithm for problem $\Pi^{AC}_{c,n}$ must use $\Omega(\frac{\log(n/t)}{\log q})$ bits of space.*

We use Theorem 1.6, (more precisely its proof arguments rather than its statement) to prove our main result (Theorem 1.3), see Section 4. At a high level, the proof of Theorem 1.3 proceeds by splitting into two cases, depending on the canonical function F. Roughly speaking, in one case we show a Shift Finding algorithm that returns a set of size $t = n/2^{\sqrt{\log n}}$ using $q = O(\log n)$ queries by binary search, and in the other case an algorithm to find the shift (i.e., $t = 1$) with probability $9/10$ using $q = 2^{\sqrt{\log n}}$ uniformly random queries.

As a corollary of Theorem 1.5, we get that the *tracking* version of approximate counting must use $\Omega(\log n)$ bits of space, which is tight with a straightforward deterministic counting. Tracking means that the algorithm produces an output after every stream item rather than at the end of the stream, and with probability $9/10$, all the outputs are simultaneously correct (i.e., approximate the number of items seen so far).

▶ **Corollary 1.7** (Tracking). *For every $c, n > 1$, every PD tracking algorithm that c-approximates the number of items in a stream of length $(c+1)n$ must use $\Omega(\log n)$ bits of space.*

In contrast, for standard randomized algorithms, there is a tracking algorithm for $(1+\epsilon)$-approximate counting that uses $O_\epsilon(\text{loglog}\, n)$ bits of space, for any fixed $\epsilon > 0$ [29]. Corollary 1.7 follows by an easy modification of the proof of Theorem 1.5. That proof uses $O(\log q)$ repetitions of a PD streaming algorithm, and then employs a union bound on q input streams, which is not necessary for tracking algorithms and thus the bound follows.

A more direct argument is essentially by equivalence to exact counting. For a stream with $s < n$ items, the state of a PD tracking algorithm with canonical function F can be used to compute s, as follows. Simulate insertion of more items to the stream until the output of the algorithm changes to 1 (which corresponds to the first 1 in F_s), from which s can be computed.

1.2.2 An Algorithm for Shift Finding

Consider a special case of the Shift Finding problem $\Pi^{SF}_{c,n}$, where the input string P is a run of zeros followed by a run of ones (viewed as a function, it is a step function); then the algorithm can perform a binary search using $O(\log(cn))$ queries, and find the unique location where F_{s^*} switches from value 0 to 1, and hence recover s^*. At the other extreme, suppose the input string P is random; then with high probability every set of $O(\log n)$ queries from P (and thus from F_{s^*}) will be answered differently (viewed as a string in $\{0,1\}^{O(\log n)}$). Based on these observations, one may hope that problem $\Pi^{SF}_{c,n}$ admits an algorithm that makes $\text{polylog}(cn)$ queries. We leave this as an open question and prove a weaker bound of $O(\sqrt{cn})$ queries.

▶ **Theorem 1.8** (Shift Finding Algorithm). *There is a deterministic algorithm for problem $\Pi^{SF}_{c,n}$ that makes $O(\sqrt{cn})$ queries.*

A key observation in our result, that may be useful in future work, is that for every shift s^* there is a "short witness" that uses exactly 2 queries. We formalize this as verifying a given guess s for the shift s^*.

▶ **Lemma 1.9** (Short Witness). *There is a deterministic algorithm that, given as input an instance of problem $\Pi^{SF}_{c,n}$ and $s < n$, makes 2 queries to F_{s^*} and returns "yes" if $s = s^*$ and "no" otherwise.*

The proofs of Theorem 1.8 and Lemma 1.9 appear in Section 5. At a high level, the Shift Finding algorithm in Theorem 1.8 queries the set $\{F_{s^*}(0), F_{s^*}(\sqrt{cn}), F_{s^*}(2\sqrt{cn}), ..., F_{s^*}(cn)\}$, and then uses the short witness (Lemma 1.9) to check every feasible $s \in [n]$ (i.e., that agrees with the query answers). Following an observation by Peter Kiss, we are able to improve our Shift Finding algorithm to use only $O((cn)^{1/3} \log n)$ queries; details omitted.

1.3 Related Work

Pseudo-deterministic algorithms

The notion of pseudo-deterministic algorithms was introduced by [9] (they originally called them Bellagio algorithms), followed by a long sequence of works that studied it in different models [13, 19, 14, 30, 24, 15, 5, 31, 11, 21, 12, 16, 26, 6, 17, 10, 7]. In the streaming and sketching models, [16] proved strong lower bounds for finding a non-zero entry in a vector (given in a stream with deletions), and for sketching ℓ_2-norms. Another related setting is that of sublinear time computation. Under certain assumptions, PD algorithms (in the sublinear time region) were shown to admit the following relation with deterministic algorithms – if for a certain problem there is a PD algorithm using q queries, then there is a deterministic algorithm using $O(q^4)$ queries [13]. The techniques of [13] do not seem to extend to streaming algorithms.

Adaptive adversarial streams

In this setting, the stream items are chosen adversarially and depend on past outputs of the streaming algorithm (i.e., the stream is adaptive) [3]. This model is considered to be between PD algorithms and the standard randomized setting, in the sense that for streams of length m, amplifying a PD algorithm to success probability $1 - \frac{1}{10m}$ (by $O(\log m)$ repetitions and taking the median) guarantees (by a union bound) that the algorithm outputs the canonical solution after every stream item with probability $9/10$, thus the adversary acts as an oblivious one (the adversary knows in advance the output of the streaming algorithm, which is the canonical function). For approximate counting, adaptive streams and standard (oblivious) streams are equivalent (since the stream items are identical) and thus admit an algorithm using $O(\text{loglog}\, n)$ bits of space.

There is a vast body of work designing algorithms for adaptive streams, but not much is known in terms of lower bounds. Lower bounds are known for some search problems, like finding a spanning forest in a graph undergoing edge insertions and deletions, but also for graph coloring [4]. Regarding estimation problems, the only lower bound we are aware of is for some artificial problem [25]. Recently, Stoeckl [32] showed a lower bound on streaming algorithms that use a bounded amount of randomness, conditioned on a lower bound for PD algorithms. In the related model of linear sketching, Hardt and Woodruff [22] showed lower bounds on the dimensions of sketching algorithms, which applies to many classical problems, like ℓ_p-norm estimation and heavy hitters.

2 Preliminaries

▶ **Definition 2.1** (Approximate counting). *Let $c, n > 1$. In problem $\Pi^{AC}_{c,n}$, the input is a stream of $l \leq (c+1)n$ identical items. The goal is to output 0 if $l \leq n$ and 1 if $l > cn$ (and otherwise the output can be either 0 or 1).*

Let A be a PD algorithm for problem $\Pi^{AC}_{c,n}$, and let $F : [0,(c+1)n] \to \{0,1\}$ be the canonical function of A. Thus, there is a fixed string $P \in \{0,1\}^{(c-1)n}$ such that

$$F(x) = \begin{cases} 0 & \text{if } x \in [0,n]; \\ 1 & \text{if } x \in [cn+1,(c+1)n]; \\ P(x-n) & \text{otherwise.} \end{cases}$$

For $s^* \in [0,n]$, let $F_{s^*} : [0,(c+1)n - s^*] \to \{0,1\}$ be a shifted version of F, namely the function $F_{s^*} : x \mapsto F(s^* + x)$. We use these notations throughout the paper.

Our proofs are based on a reduction from a simple one-way communication problem, called MESSAGE and denoted Π^{MSG}_{Σ}, where Alice's input x is from an alphabet Σ that is fixed in advance, Bob has no input, and the goal is that Bob outputs x with probability at least $2/3$. It is well known that this problem requires $\Omega(\log|\Sigma|)$ bits of communication, even for randomized protocols using shared randomness. We provide a proof for completeness.

▶ **Lemma 2.2.** *For every alphabet Σ, every one-way communication protocol (even with shared randomness) for problem Π^{MSG}_{Σ} must use $\Omega(\log|\Sigma|)$ bits of communication.*

Proof. Let $\mathcal{A}$ be a protocol for problem Π^{MSG}_{Σ}. For a random string r representing the randomness of $\mathcal{A}$, let $\Sigma_r \subset \Sigma$ be the set of all $s \in \Sigma$ for which Bob correctly recovers s. Let r^* be a string maximizing $|\Sigma_r|$, then by averaging, $|\Sigma_{r^*}| \geq \frac{2}{3}|\Sigma|$. Consider an instance of $\mathcal{A}$ that uses r^* as its random string. Assume by contradiction that the number of communication bits is less than $\log|\Sigma_{r^*}|$, then by the pigeonhole principle there are two distinct inputs $s, s' \in \Sigma_{r^*}$ such that $\mathcal{A}(s)$ and $\mathcal{A}(s')$ result in the same message. Bob then cannot distinguish between (i.e., has the same output distribution for) s and s', a contradiction. Hence, the number of bits of communication is at least $\log|\Sigma_{r^*}| = \Omega(\log|\Sigma|)$. ◀

3 Lower Bounds for PD Approximate Counting via Shift Finding

In this section, we prove Theorem 1.5. The proof involves three problems from different settings: (a) PD approximate counting in the streaming model; (b) Shift Finding in the query-access model; and (c) MESSAGE in one-way communication with shared randomness. The proof essentially shows that if there is an algorithm for Shift Finding that makes only q queries and also a streaming algorithm for PD approximate counting that uses b bits of space, then MESSAGE can be solved using $O(b \log q)$ bits of communication. Combining this bound with the well-known lower bound for MESSAGE in Lemma 2.2 yields a lower bound for b.

A core idea in the proof is that an execution of a PD streaming algorithm A for the approximate counting problem $\Pi^{AC}_{c,n}$ on a stream with s^* insertions, can be used (even without knowing s^*, by making additional insertions and then querying the streaming algorithm A) to provide query access to the shifted function $F_{s^*} : x \mapsto F(s^* + x)$. This query access, along with a query-efficient algorithm for the Shift Finding problem $\Pi^{SF}_{c,n}$, is then used to solve an instance of the MESSAGE problem Π^{MSG}_{Σ}.

In fact, we prove the following theorem, which holds for each string F separately (rather than a bound that depends on the worst-case F), and yields Theorem 1.5 as an immediate corollary.

▶ **Theorem 3.1.** *Let A be a PD streaming algorithm for problem $\Pi^{AC}_{c,n}$, where $c, n > 1$, and let $F : [0, (c+1)n] \to \{0, 1\}$ be the canonical function of A. Suppose that Shift Finding with respect to this specific F (the problem of finding an unknown shift $s^* \in [n]$ with probability at least $9/10$ given query access to F_{s^*}) admits a randomized algorithm that makes at most $q = q(F)$ (possibly adaptive) queries. Then the streaming algorithm A must use $\Omega(\frac{\log n}{\log q})$ bits of space.*

Proof. Define algorithm A' to be an amplification of A to success probability $1 - 1/(10q)$, by running $O(\log q)$ independent repetitions and reporting their majority. Assume there exists an algorithm Q that for every $s^* \in [n]$, makes at most $q = q(F)$ queries to F_{s^*} (possibly adaptive) and outputs s^* with probability at least $9/10$.

Consider an instance of problem Π^{MSG}_{Σ} with alphabet $\Sigma = [0, n]$, and consider the following protocol for it. Alice starts an execution of the streaming algorithm A' using the shared randomness, then takes her input $s^* \in \Sigma$ and makes s^* stream insertions to algorithm A', and finally sends the state (memory contents) of A' to Bob.

Bob continues the execution of the streaming algorithm A' (using the shared randomness), and uses it to provide query access to F_{s^*}, as follows. In order to query F_{s^*} at any index x, Bob makes a fresh copy A_0 of the streaming algorithm A', insert x stream items to algorithm A_0 and then reads its output. With probability at least $1 - 1/(10q)$, the answer that Bob gets is indeed $F_{s^*}(x)$ (because the number of items inserted to this instance of the algorithm is $x + s^*$). Bob uses this query access and his knowledge of F to simulate algorithm Q (with the goal of recovering s^*).

Consider Bob's simulation of algorithm Q. If Q was executed with true query access to F_{s^*}, then it would have had success probability $9/10$, and would have made a sequence of queries X_Q to F_{s^*}. This sequence X_Q depends only on F_{s^*} and the coin tosses of algorithm Q. In particular, revealing X_Q (i.e., conditioned on X_Q) does not affect the coins of the streaming algorithm A', and it still succeeds with probability at least $1 - 1/(10q)$. We can thus apply a union bound to conclude that algorithm A' succeeds on all queries $x \in X_Q$ (i.e., outputs the corresponding $F_{s^*}(x)$) with probability at least $1 - q \cdot \frac{1}{10q} = 9/10$. Hence, when Bob simulates algorithm Q using the streaming algorithm A', with probability $9/10$ (over the coins of A') the execution is identical to running algorithm Q with true access to F_{s^*}, which itself succeeds with probability $9/10$. By a union bound, with probability $8/10$ both algorithm Q and the streaming algorithm A' succeed, in which case Bob recovers s^*, and therefore this communication protocol solves problem Π^{MSG}_{Σ} with alphabet $\Sigma = [0, n]$.

By Lemma 2.2, the message Alice sends must contain $\Omega(\log n)$ bits, and thus the streaming algorithm A' must use $\Omega(\log n)$ bits of space. Recall that algorithm A' consists of $O(\log q)$ copies of the streaming algorithm A and thus algorithm A must use $\Omega(\frac{\log n}{\log q})$ bits of space. ◀

4 Lower Bound for PD Approximate Counting

In this section, we prove Theorem 1.3, i.e., for every $c, n > 1$, we prove that every PD streaming algorithm for the approximate counting problem $\Pi^{AC}_{c,n}$ must use $\Omega_c(\sqrt{\frac{\log n}{\operatorname{loglog} n}})$ bits of space.

Let F be the canonical function of a PD streaming algorithm for problem $\Pi^{AC}_{c,n}$. Our analysis is split into two cases depending on F, which informally correspond to whether a fixed pattern (like "01") appears in the string F at most t times or not. These cases are analyzed using Theorems 1.6 and 3.1. The overall bound will be derived by optimizing the threshold t between the two cases to roughly $t = n/2^{\sqrt{\log n}}$.

4.1 Scenario One

In this scenario, there is a specific pattern in F that appears at most t times, where $t = t_c(n)$ will be set at the end of our proof. We first consider the pattern "01" in F, which corresponds to $x \in [0, (c+1)n-1]$ such that $F(x) = 0$ and $F(x+1) = 1$, and later generalize this pattern to a broader family.

▶ **Lemma 4.1.** *If the pattern "01" appears at most t times in F, then every PD streaming algorithm for problem $\Pi^{AC}_{c,n}$ whose canonical function is F must use $\Omega(\frac{\log(n/t)}{\operatorname{loglog}(cn)})$ bits of space.*

Proof. The proof is by a reduction from problem MESSAGE, similarly to the proof of Theorem 3.1. Perhaps the most delicate part is the definition of an alphabet Σ for the MESSAGE problem Π^{MSG}_{Σ}, and it proceeds as follows.

Given $s \in [n]$, consider the following execution of Binary Search on the function F_s. Initialize $l = 0$ and $r = cn + 1$, and at every iteration query $F_s(\lfloor\frac{l+r}{2}\rfloor)$; if $F_s(\lfloor\frac{l+r}{2}\rfloor) = 0$, then $l \leftarrow \lfloor\frac{l+r}{2}\rfloor$, otherwise $r \leftarrow \lfloor\frac{l+r}{2}\rfloor$. These iterations maintain the invariant that $F_s(l) = 0$ and $F_s(r) = 1$, and after at most $\log(cn)$ iterations arrive at $r = l + 1$ with the pattern "01". Define a mapping $M : [n] \to [cn]$ such that $M(s)$ is the location where the binary search finds a "01" in F_s, i.e., the final index l; thus $F(s + M(s)) = 0$ and $F(s + M(s) + 1) = 1$.

In order to define an alphabet Σ, consider a partitioning of $[n]$ to buckets, defined such that items s, s' are from the same bucket B if and only if they are mapped to the same value $M(s) = M(s')$. For every bucket B and every $s, s' \in B$, we know from above that $F(s' + M(s)) = 0$ and $F(s' + M(s) + 1) = 1$, so there are at most t possibilities for s' (one of which is $s' = s$), and thus the size of the bucket $|B| \leq t$. Define $\Sigma \subset [n]$ by taking one representative from each bucket. Thus, every $s_1 \neq s_2 \in \Sigma$ satisfy $M(s_1) \neq M(s_2)$ and $|\Sigma| \geq n/t$.

Let A be a streaming algorithm whose canonical function is F and let algorithm A' be an amplification of algorithm A that succeeds with probability $1 - 1/(10\log(cn))$ (by making $O(\operatorname{loglog}(cn))$ repetitions and taking the majority). Consider an instance of the MESSAGE problem Π^{MSG}_{Σ}, and proceed similarly to the proof of Theorem 3.1. We provide a self-contained analysis for completeness. Alice and Bob perform the following protocol. Alice starts an execution of algorithm A' using the shared randomness. For input $s^* \in \Sigma$, she inserts s^* stream items to algorithm A' and sends the state (memory contents) of this algorithm A' to Bob. In order to get query access to F_{s^*} at index x, Bob makes a fresh copy A_0 of algorithm A', continues the algorithm's execution (using the shared randomness), inserts x stream items to algorithm A_0 and finally reads its output. Bob uses this query access to simulate the Binary Search algorithm on F_{s^*} (with the goal of recovering $M(s^*)$). He then infers which bucket corresponds to his result, and outputs the representative of that bucket (which is s^* if he recovers $M(s^*)$).

If the Binary Search algorithm were executed with true query access to F_{s^*}, then it would have output $M(s^*)$ and would have made a sequence of queries X_{BS} to F_{s^*}. This sequence depends only on F_{s^*}, and in particular independent of the random coins of algorithm A'. Thus by a union bound, algorithm A' succeeds on all queries $x \in X_{BS}$ (i.e. outputs the corresponding $F_{s^*}(x)$) with probability at least $1 - \log(cn) \cdot 1/(10\log(cn)) = 9/10$. Hence, when Bob simulates the Binary Search algorithm using the streaming algorithm A', then with probability 9/10 the execution is identical to running the Binary Search algorithm with true query access to F_{s^*}. Thus with this probability 9/10, Bob recovers $M(s^*)$, and hence outputs s^*, which concludes the correctness analysis of the communication protocol.

By Lemma 2.2, the message Alice sends must contain $\Omega(\log|\Sigma|) \geq \Omega(\log(n/t))$ bits, and thus algorithm A' must use $\Omega(\log(n/t))$ bits of space. Recall that algorithm A' is made of $O(\text{loglog}(cn))$ copies of algorithm A and thus algorithm A must use $\Omega(\frac{\log(n/t)}{\text{loglog}(cn)})$ bits of space. ◀

▶ Remark 4.2. This proof can be easily generalized to prove Theorem 1.6. The first extension is by replacing the Binary Search algorithm and the corresponding buckets with any deterministic algorithm Q that returns a subset containing s^*. In order to generalize Q to any PD algorithm Y, consider the canonical function of Y instead of the mapping M, and apply the same proof. It holds because the crucial property of the Binary Search algorithm was the existence of the mapping M. Then by an additional union bound, both algorithms Q and A' succeed with probability 8/10 (as in the proof of Theorem 3.1).

We now generalize Lemma 4.1 to a larger family of patterns in F, where each pattern is characterized by a parameter $k \in [n]$, and appears at index $x \in [0,(c+1)n-k]$ such that $F(x) = 0$ and $F(x+k) = 1$. These patterns are allowed to overlap with each other (for different values of k). Denote such a pattern by "$0?^{k-1}1$", where each question mark can represent either 0 or 1, and the number of question marks is $k-1 < n$. A copy of this pattern can be found in $O(\log\frac{n}{k})$ queries to F_{s^*} by a binary search on the grid $(0, k, ..., \lceil\frac{cn}{k}\rceil k)$, since $F_{s^*}(0) = 0$ and $F_{s^*}(\lceil\frac{cn}{k}\rceil k) = 1$. Hence, if there exists k for which this pattern appears at most t times in F, then the communication protocol above can be adjusted to imply that algorithm A must use at least $\Omega(\frac{\log(n/t)}{\text{loglog}(cn/k)}) \geq \Omega(\frac{\log(n/t)}{\text{loglog}(cn)})$ bits of space. The only change in the proof is in the number of queries that Bob makes, which affects the number of repetitions in algorithm A', and thus only affects the loglog term.

▶ **Corollary 4.3.** *If for some $k \leq n$ the pattern "$0?^{k-1}1$" appears at most t times in F, then every PD streaming algorithm for problem $\Pi^{AC}_{c,n}$ whose canonical function is F, must use $\Omega(\frac{\log(n/t)}{\text{loglog}(cn)})$ bits of space.*

4.2 Scenario Two

In this scenario, for every $k \leq n$ the pattern "$0?^{k-1}1$" appears at least t times in F.

▶ **Lemma 4.4.** *If for all $k \in [n]$, the pattern "$0?^{k-1}1$" appear at least t times in F, then every PD streaming algorithm for problem $\Pi^{AC}_{c,n}$ whose canonical function is F, must use $\Omega(\frac{\log n}{\log(cn/t)+\text{loglog}\, n})$ bits of space.*

Proof. In this case, there is an algorithm for the Shift Finding problem $\Pi^{SF}_{c,n}$ using $q = O(\frac{cn\log n}{t})$ queries to F_{s^*}, as follows.

```
1. let $S = [0,n]$
2. repeat the following $\frac{10cn\log n}{t}$ times:
   a. pick $r \in [cn]$ uniformly at random and query $F_{s^*}(r)$
   b. let $S \leftarrow \{s \in S : F(s+r) = F_{s^*}(r)\}$
3. if $|S| = 1$, return $s \in S$; else return FAIL
```

The final set S clearly contains the shift s^*. It remains to show that all $s \neq s^*$ are removed from the set S with high probability.

Fix $s \in [n], s \neq s^*$. There are t values for $r \in [cn]$ for which $F(s^* + r) \neq F(s + r)$, as follows. Assume without loss of generality that $s^* < s$ and denote $k = s - s^* \in [n]$. Let l be a location that corresponds to the pattern "$0?^{k-1}1$" in F, i.e. $F(l) = 0$ and $F(l + k) = 1$. If $l \in [s^* + 1, s^* + cn]$, then there is $r \in [cn]$ such that $s^* + r = l$, for which $F(s^* + r) = 0 \neq F(l + k) = F(s + r)$. There are at least t locations for this pattern (i.e. possible values for l), thus it remains to show that indeed $l \in [s^* + 1, s^* + cn]$. It must be that $l + k > n$ since $F(x) = 0$ for all $x \leq n$, and similarly $l \leq cn$ since $F(x) = 1$ for all $x > cn$. Hence $l \in [n - k + 1, cn] \subset [s^* + 1, s^* + cn]$, and thus there are t values for $r \in [cn]$ for which $F(s^* + r) \neq F(s + r)$ (each value for r corresponds to a possible value for l).

Thus, in each repetition, s is removed from the set S with probability at least $\frac{t}{cn}$. The probability s is not removed after $\frac{10cn \log n}{t}$ repetitions is $(1 - \frac{t}{cn})^{(10cn \log n)/t} < \frac{1}{n^2}$. By a union bound, all $s \neq s^*$ are removed with probability $1 - \frac{1}{n}$, which concludes the correctness analysis of the algorithm for problem $\Pi^{SF}_{c,n}$.

By Theorem 3.1, every PD streaming algorithm for the approximate counting problem $\Pi^{AC}_{c,n}$ with a canonical function F must use $\Omega(\frac{\log n}{\log((cn \log n)/t)})$ bits of space. ◀

4.3 Concluding the Proof of Theorem 1.3

Concluding the two scenarios, set $t = n/2^{\sqrt{\log n \cdot \log\log(cn)}}$ and get by Corollary 4.3 and Lemma 4.4 that every PD streaming algorithm for the approximate counting problem $\Pi^{AC}_{c,n}$ must use

$$\Omega(\min\{\frac{\log(n/t)}{\operatorname{loglog}(cn)}, \frac{\log n}{\log((cn/t)\log n)}\}) = \Omega(\frac{\log n}{\sqrt{\log n \operatorname{loglog}(cn)}+\log c})$$

bits of space, which boils down to $\Omega(\sqrt{\frac{\log n}{\operatorname{loglog} n}})$ for $c < 2^{\sqrt{\log n \operatorname{loglog} n}}$.

5 Shift Finding Algorithm

One can hope to prove tighter lower bounds for PD streaming algorithms for the approximate counting problem $\Pi^{AC}_{c,n}$, and a possible approach is by solving the Shift Finding problem $\Pi^{SF}_{c,n}$ using $\operatorname{polylog} n$ queries. Recall that in problem $\Pi^{SF}_{c,n}$, the input is a string $P \in \{0,1\}^{(c-1)n}$, which can be represented by a string F which is a concatenation of n zeros, P and then n ones; and query access to a shifted version of F with shift s^*, denoted F_{s^*}. As stated in Theorem 1.8, we show a deterministic algorithm for problem $\Pi^{SF}_{c,n}$ using $O(\sqrt{cn})$ queries (Algorithm 1), and we leave open the question whether it is the right bound. The proof relies on an efficient verification algorithm that for input s, uses 2 queries and returns "yes" if and only if $s = s^*$, as stated in Lemma 1.9 and described next.

Proof of Lemma 1.9. Denote by $l \in [n + 1, cn + 1]$ the smallest number such that $F(l) = 1$, and by $r \in [n, cn]$ the largest number such that $F(r) = 0$. For input $s \in [0, n]$, the verification algorithm returns "no" if $F_{s^*}(l - s) = 0$ or $F_{s^*}(r - s) = 1$, and otherwise returns "yes".

If $s = s^*$, then $F_{s^*}(x - s) = F(x)$ and the verification algorithm outputs "yes". If $s > s^*$, then $s^* - s + l < l$ and thus $F_{s^*}(l - s) = F(s^* - s + l) = 0$ and the verification algorithm outputs "no". Similarly, if $s < s^*$ then $F_{s^*}(r - s) = 1$ and the verification algorithm outputs "no". ◀

▶ Remark 5.1. There is a randomized algorithm for problem $\Pi^{SF}_{c,n}$ using $\tilde{O}_c(\sqrt{n})$ queries that is similar to the proof of Theorem 1.3 in Section 4. It proceeds by considering those two scenarios. In scenario one, instead of constructing the set Σ, query witnesses for all the t

possible shifts using $2t$ queries and hence recover the unknown shift s^*. In scenario two, the proof of Theorem 1.3 shows how to find the unknown shift s^* in $O(\frac{cn}{t}\log n)$ queries with high probability. Hence, by setting $t = \sqrt{cn \log n}$, this algorithm finds the unknown shift in $O(\max\{t + \log(cn), \frac{cn}{t}\log n\}) \leq O(\sqrt{cn \log n})$ queries with high probability.

Next is a slight improvement, a deterministic algorithm in $O(\sqrt{cn})$ queries, proving Theorem 1.8.

Algorithm 1 Deterministic Shift Finding in $O(\sqrt{cn})$ queries.

```
Input: $n, c, F$ and query access to $F_{s^*}$
Output: $s^*$
$Q \leftarrow (F_{s^*}(0), F_{s^*}(\sqrt{cn}), F_{s^*}(2\sqrt{cn}), ..., F_{s^*}(cn))$
let $S \leftarrow \{s \in [0,n] : \forall i \in [0,\sqrt{cn}], F_s(i\sqrt{cn}) = Q(i)\}$ ▷ i.e. the set of all shifts that could have produced $Q$
for $s \in S$ do
    check the witness of $s$
    if $s = s^*$ then return $s$
```

▶ **Lemma 5.2.** *The set S in Algorithm 1 is of size $O(\sqrt{cn})$.*

Proof. Assume by contradiction that $|S| \geq \sqrt{cn} + 1$. Hence by the pigeonhole principle, there exists $s_1 < s_2 \in S$ such that $s_1 = s_2 \mod \sqrt{cn}$. Hence for all $i \in [0, \sqrt{cn} - \frac{s_2 - s_1}{\sqrt{cn}}]$,

$$Q(i) = F_{s_2}(i\sqrt{cn}) = F_{s_1}(s_2 - s_1 + i\sqrt{cn}) = Q(\tfrac{s_2 - s_1}{\sqrt{cn}} + i),$$

where the first and last transitions hold since $s_1, s_2 \in S$ and $\frac{s_2 - s_1}{\sqrt{cn}}$ is an integer number, and the second transition is by definition. Thus Q has a period of length $\frac{s_2 - s_1}{\sqrt{cn}} \leq \lfloor \frac{s_2}{\sqrt{cn}} \rfloor$. However, for $i \in [\sqrt{cn} - \lfloor \frac{s_2}{\sqrt{cn}} \rfloor + 1, \sqrt{cn}]$ the values that Q get are $Q(i) = F_{s_2}(i\sqrt{cn}) = 1$ since $s_2 + i\sqrt{cn} \geq cn$; thus all entries in Q are equal 1, which contradicts the fact that $Q(0) = 0$, and thus completes the proof. ◀

Algorithm 1 returns the shift s^* since $s^* \in S$ and by the correctness of the verifier in Lemma 1.9. The number of queries Algorithm 1 makes is $O(|S| + |Q|) = O(\sqrt{cn})$, which proves Theorem 1.8.

References

1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996. doi:10.1145/237814.237823.

2 Alexandr Andoni, Piotr Indyk, Dina Katabi, and Haitham Hassanieh. Shift finding in sub-linear time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 457–465, 2013. doi:10.1137/1.9781611973105.33.

3 Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. *J. ACM*, 69(2):17:1–17:33, 2022. doi:10.1145/3498334.

4 Amit Chakrabarti, Prantar Ghosh, and Manuel Stoeckl. Adversarially robust coloring for graph streams. In *13th Innovations in Theoretical Computer Science Conference, ITCS*, volume 215 of *LIPIcs*, pages 37:1–37:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.37.

5 Peter Dixon, A. Pavan, and N. V. Vinodchandran. On pseudodeterministic approximation algorithms. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 117 of *LIPIcs*, pages 61:1–61:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.61.

6 Peter Dixon, A. Pavan, and N. V. Vinodchandran. Complete problems for multi-pseudodeterministic computations. In *12th Innovations in Theoretical Computer Science Conference, ITCS*, volume 185 of *LIPIcs*, pages 66:1–66:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ITCS.2021.66.

7 Peter Dixon, A. Pavan, Jason Vander Woude, and N. V. Vinodchandran. Pseudodeterminism: promises and lowerbounds. In *STOC '22: 54th Annual ACM Symposium on Theory of Computing*, pages 1552–1565, 2022. doi:10.1145/3519935.3520043.

8 Philippe Flajolet. Approximate counting: A detailed analysis. *BIT*, 25(1):113–134, 1985. doi:10.1007/BF01934993.

9 Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electron. Colloquium Comput. Complex.*, TR11-136, 2011. URL: https://eccc.weizmann.ac.il/report/2011/136, arXiv:TR11-136.

10 Sumanta Ghosh and Rohit Gurjar. Matroid intersection: A pseudo-deterministic parallel reduction from search to weighted-decision. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 207 of *LIPIcs*, pages 41:1–41:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.APPROX/RANDOM.2021.41.

11 Michel X. Goemans, Shafi Goldwasser, and Dhiraj Holden. Doubly-efficient pseudo-deterministic proofs. *CoRR*, abs/1910.00994, 2019. arXiv:1910.00994.

12 Oded Goldreich. Multi-pseudodeterministic algorithms. *Electron. Colloquium Comput. Complex.*, TR19-012, 2019. URL: https://eccc.weizmann.ac.il/report/2019/012, arXiv:TR19-012.

13 Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *Innovations in Theoretical Computer Science, ITCS*, pages 127–138. ACM, 2013. doi:10.1145/2422436.2422453.

14 Shafi Goldwasser and Ofer Grossman. Perfect bipartite matching in pseudo-deterministic RNC. *Electron. Colloquium Comput. Complex.*, TR15-208, 2015. URL: https://eccc.weizmann.ac.il/report/2015/208, arXiv:TR15-208.

15 Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-deterministic proofs. In *9th Innovations in Theoretical Computer Science Conference, ITCS*, volume 94 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.17.

16 Shafi Goldwasser, Ofer Grossman, Sidhanth Mohanty, and David P. Woodruff. Pseudo-deterministic streaming. In *11th Innovations in Theoretical Computer Science Conference, ITCS*, volume 151 of *LIPIcs*, pages 79:1–79:25, 2020. doi:10.4230/LIPIcs.ITCS.2020.79.

17 Shafi Goldwasser, Russell Impagliazzo, Toniann Pitassi, and Rahul Santhanam. On the pseudo-deterministic query complexity of NP search problems. In *36th Computational Complexity Conference, CCC*, volume 200 of *LIPIcs*, pages 36:1–36:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CCC.2021.36.

18 André Gronemeier and Martin Sauerhoff. Applying approximate counting for computing the frequency moments of long data streams. *Theory Comput. Syst.*, 44(3):332–348, 2009. doi:10.1007/s00224-007-9048-z.

19 Ofer Grossman. Finding primitive roots pseudo-deterministically. *Electron. Colloquium Comput. Complex.*, TR15-207, 2015. URL: https://eccc.weizmann.ac.il/report/2015/207, arXiv:TR15-207.

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

21 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, SODA*, pages 606–620. SIAM, 2019. doi:10.1137/1.9781611975482.38.

22 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, 2013. doi:10.1145/2488608.2488624.

23 Haitham Hassanieh, Fadel Adib, Dina Katabi, and Piotr Indyk. Faster GPS via the sparse fourier transform. In *The 18th Annual International Conference on Mobile Computing and Networking, Mobicom*, pages 353–364. ACM, 2012. doi:10.1145/2348543.2348587.

24 Dhiraj Holden. A note on unconditional subexponential-time pseudo-deterministic algorithms for BPP search problems. *CoRR*, abs/1707.05808, 2017. arXiv:1707.05808.

25 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*, volume 12827 of *Lecture Notes in Computer Science*, pages 94–121. Springer, 2021. doi:10.1007/978-3-030-84252-9_4.

26 Zhenjian Lu, Igor Carboni Oliveira, and Rahul Santhanam. Pseudodeterministic algorithms and the structure of probabilistic time. In *STOC '21: 53rd Annual ACM Symposium on Theory of Computing*, pages 303–316, 2021. doi:10.1145/3406325.3451085.

27 Jérémie O. Lumbroso. How Flajolet processed streams with coin flips. *CoRR*, abs/1805.00612, 2018. arXiv:1805.00612.

28 Robert Morris. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842, 1978. doi:10.1145/359619.359627.

29 Jelani Nelson and Huacheng Yu. Optimal bounds for approximate counting. In *Proceedings of the 41st ACM Symposium on Principles of Database Systems*, PODS, pages 119–127, 2022. doi:10.1145/3517804.3526225.

30 Igor Carboni Oliveira and Rahul Santhanam. Pseudodeterministic constructions in subexponential time. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing, STOC*, pages 665–677, 2017. doi:10.1145/3055399.3055500.

31 Igor Carboni Oliveira and Rahul Santhanam. Pseudo-derandomizing learning and approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, volume 116 of *LIPIcs*, pages 55:1–55:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.APPROX-RANDOM.2018.55.

32 Manuel Stoeckl. Streaming algorithms for the missing item finding problem. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 793–818, 2023. doi:10.1137/1.9781611977554.ch32.