2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)

The Coin Problem with Applications to Data Streams

Mark Braverman
Computer Science Department
Princeton University
Princeton, USA
Email: mbraverm@gmail.com

Sumegha Garg
Computer Science Department
Harvard University
Cambridge, USA
Email: sumegha.garg@gmail.com

David P. Woodruff
Computer Science Department
CMU
Pittsburgh, USA
Email: dwoodruf@andrew.cmu.edu

***Abstract*—Consider the problem of computing the majority of a stream of n i.i.d. uniformly random bits. This problem, known as the *coin problem*, is central to a number of counting problems in different data stream models. We show that any streaming algorithm for solving this problem with large constant advantage must use $\Omega(\log n)$ bits of space. We extend our lower bound to proving tight lower bounds for solving multiple, randomly interleaved copies of the coin problem, as well as for solving the OR of multiple copies of a variant of the coin problem. Our proofs involve new measures of information complexity that are well-suited for data streams.**

We use these lower bounds to obtain a number of new results for data streams. In each case there is an underlying d-dimensional vector x with additive updates to its coordinates given in a stream of length m. The input streams arising from our coin lower bound have nice distributional properties, and consequently for many problems for which we only had lower bounds in general turnstile streams, we now obtain the same lower bounds in more natural models, such as the bounded deletion model, in which $\|x\|_2$ never drops by a constant fraction of what it was earlier, or in the random order model, in which the updates are ordered randomly. In particular, in the bounded deletion model, we obtain nearly tight lower bounds for approximating $\|x\|_\infty$ up to additive error $\frac{1}{\sqrt{k}}\|x\|_2$, approximating $\|x\|_2$ up to a multiplicative $(1+\epsilon)$ factor (resolving a question of Jayaram and Woodruff in PODS 2018), and solving the Point Query and ℓ_2-Heavy Hitters Problems. In the random order model, we also obtain new lower bounds for the Point Query and ℓ_2-Heavy Hitters Problems. We also give new algorithms complementing our lower bounds and illustrating the tightness of the models we consider, including an algorithm for approximating $\|x\|_\infty$ up to additive error $\frac{1}{\sqrt{k}}\|x\|_2$ in turnstile streams (resolving a question of Cormode in a 2006 IITK Workshop), and an algorithm for finding ℓ_2-heavy hitters in randomly ordered insertion streams (which for random order streams, resolves a question of Nelson in a 2018 Warwick Workshop).

***Keywords*-space lower bounds; majority; streaming algorithms; coin problem; information cost; heavy hitters; bounded deletion model**

Research supported in part by the NSF Alan T. Waterman Award, Grant No. 1933331, a Packard Fellowship in Science and Engineering, and the Simons Collaboration on Algorithms and Geometry.

This work was completed when Sumegha was at Princeton.

Research supported in part by the National Science Foundation under Grant No. CCF-1815840.

I. Introduction

The data stream model is an important model for processing sequences of data that are too large to be stored in memory. Common examples of data streams include search logs, scientific data, sensor networks, and network traffic. In many of these applications, an algorithm is only allowed one pass over the stream and must store a short summary of what it has seen in order to answer a pre-defined query. We refer the reader to several surveys on data streams [1], [2], [3].

The arguably most fundamental problem in data streams is maintaining a counter C. Formally, suppose we are given a sequence of at most n updates of the form $C \leftarrow C+1$ in an arbitrary order and we are allowed one pass over this sequence. While one can compute the exact count C using $\log_2 n$ bits of space, one can use significantly less space to compute a constant factor multiplicative approximation with constant probability; indeed, this is possible using only $O(\log\log n)$ bits of space using a probabilistic counter [4]. However, suppose now we see a stream of both positive and negative updates of the form $C \leftarrow C+1$ or $C \leftarrow C-1$. In this case $\Omega(\log n)$ bits of space are needed even to compute a constant factor approximation with constant probability [5].

But perhaps surprisingly, suppose now we see a uniformly random sequence of n i.i.d. updates of the form $C \leftarrow C+1$ or $C \leftarrow C-1$, meaning each update is independently $C \leftarrow C+1$ with probability $1/2$ and $C \leftarrow C-1$ with probability $1/2$. The following question is open:

How much space is needed to compute a constant factor approximation to C w.h.p. for a uniformly random sequence of n i.i.d. updates of the form $C \leftarrow C+1$ or $C \leftarrow C-1$ (each with probability $1/2$)?

This problem is also known as the *coin problem* and has been studied in the context of branching program and circuit complexity [6], [7], [8], [9], though such results only give an $\Omega(\log\log n)$ space lower bound for our problem. This is far from the best known upper bound, which is $O(\log n)$ bits.

Despite the large body of work on data streams, and the simplicity of this question, why is it still open? To understand this, we look at common techniques for proving lower bounds in data streams. Arguably the most common is communication complexity, and in particular, 2-player communication complexity. We (typically) give half of the stream to Alice, and half of the stream to Bob, and Alice runs the streaming algorithm on her input. She then transmits the state of her algorithm to Bob, who continues the execution of the streaming algorithm on his input. If the output of the streaming algorithm can be used to solve the communication game, then the amount of memory of the streaming algorithm must be at least the 1-way communication of the game. Unfortunately, this approach fails for the above problem. Indeed, with large probability the count on Alice's stream will be $C\sqrt{n}$ in absolute value, for a constant $C > 0$, and it suffices for Alice to just send $O(1)$ bits to round the value of C to a smaller constant. Then Bob adds this to the count of his stream, which is also random, and obtains a constant factor approximation with large constant probability. The issue is the random order of the stream makes this problem too easy.

When 2-player communication complexity fails, one often resorts to multiplayer communication complexity to prove lower bounds. We again give each player a portion of the stream and perform the reduction above, passing the state of the streaming algorithm from one player to the next. However, when lower bounding the communication for the purposes of streaming, existing techniques have only looked at the blackboard or coordinator models of communication. In the blackboard model, the message sent from one player to the next is seen by all future players. For the question above, suppose we arbitrarily partition the stream into pieces and give a piece to each player. In this case, it is known that if each player randomly rounds their own input count so that the expectation is their actual count, then one can obtain a constant factor approximation to the overall count from these rounded counts with large probability [10]. This $O(\log \log n)$ bit upper bound is a simultaneous protocol (since each player's message does not depend on prior messages) and thus also holds in the so-called coordinator model. In fact, an $o(\log n)$ bit upper bound holds whenever the diameter of the underlying communication topology is sub-polynomial; see [10] for further discussion.

This helps explain why the above question is still open. Existing techniques simply do not capture the intuition for the problem above that each player needs to forward many bits of information about the inputs of previous players; in both communication models described above, the players only need to concern themselves with their own input, and let the blackboard or coordinator figure out the rest. While there are some topology-dependent communication lower bounds [11], [12], the problems studied in those works are not useful here and logarithmic factors for those problems are not accounted for, but here this is the main goal.

A. *Our Results*

We resolve the complexity of the coin problem. Namely, we show the following:

Theorem 1. *(Informal Coin Problem Lower Bound) Let $X_1, \ldots, X_n$ be a stream of uniform i.i.d. $\{0, 1\}$ bits. Let A be any 1-pass streaming algorithm which reads $X_1, \ldots, X_n$ in order and outputs the majority bit with probability at least 0.999. Then A uses $\Omega(\log n)$ bits of memory.*

Here, the probability is over the input and the randomness used. Our proof of Theorem 1 uses techniques from information complexity, elaborated upon below. We note that the right definition of the information cost of a streaming algorithm is crucial for proving Theorem 1. We give two different definitions: (1) $\sum_{i=1}^{n} I(M_i; X_{\leq i})$ and (2) $\sum_{i=1}^{n} \sum_{j \leq i} I(M_i; X_j | M_{j-1})$, where M_i is the memory state of the algorithm after processing the i-th bit X_i. Here $I(A; B)$ denotes the mutual information between random variables A and B. We show that, for any deterministic algorithm computing the majority with 0.999 probability, both of these quantities are $\Omega(n \log n)$. On the other hand, if the streaming algorithm is allowed to use private randomness (the streaming algorithm uses fresh randomness at every time step), then there exists a majority-computing algorithm such that the information cost as defined in (1) is $O(n)$, but for (2) it is still $\Omega(n \log n)$ for any streaming algorithm computing majority that uses private randomness. Nevertheless, since we want a memory lower bound for the streaming algorithm for the uniform distribution on $X_1, \ldots, X_n$, we can always assume w.l.o.g. that the algorithm is deterministic by first fixing its private randomness. For deterministic algorithms with memory r, (1) evaluates to at most $n \cdot r$, thus implying a memory lower bound of $\Omega(\log n)$ bits. We also prove that for any streaming algorithm (possibly using private randomness) with memory r, (2) evaluates to at most $n \cdot r$, thus also implying a memory lower bound of $\Omega(\log n)$ bits. We then have a separation between information and space complexity for streaming algorithms (using private randomness) with respect to (1).

We next consider the problem of solving multiple copies of the coin problem simultaneously. Through a non-standard use of the data processing inequality and fundamental properties of mutual information, we are able to prove the following direct sum like theorem.

Theorem 2. *(Informal Direct Sum Theorem) Suppose we are given a sequence of kn i.i.d. stream updates, the j-th of which has the form $Y_j = (X_j, s_j)$, where s_j is chosen uniformly in $\{1, 2, \ldots, k\}$ and X_j is chosen uniformly in $\{0, 1\}$. We interpret this as k independent instances of the coin problem, where the ℓ-th instance of the coin problem consists of the sequence of bits $X_{j_1}, \ldots, X_{j_r}$, where $s_{j_t} = \ell$ for each $t = 1, \ldots, r$. Suppose there is a streaming algorithm which, given an $\ell \in [k]$ at the end of the stream, outputs the majority* *bit of the ℓ-th instance of the coin problem with probability at least $1 - \frac{1}{4000}$. Then the algorithm uses $\Omega(k \log n)$ bits of memory.*

Here, the probability is over the input stream, private randomness used, and ℓ, which is chosen uniformly at random from $[k]$. We refer to the problem in Theorem 2 as the k-COINS PROBLEM for a random order.

We also prove a variant of Theorem 2 in which we see exactly one update to each of the k instances, followed by exactly one update to each of the k instances again, and so on, repeating for n steps. That is, $s_j = (j-1) \mod k + 1$ for all $j \in [nk]$. We call this the SIMULTANEOUS k-COINS PROBLEM.

Next, using the notion of information cost for streaming algorithms, we show the hardness of calculating sums in the coin problem even up to large gaps. Consider the following problem. For a sequence of ± 1 integers, let S denote their sum. We say that the instance is a 0-instance if $|S| \le 4\sqrt{n\alpha}$ and a 1-instance if $|S| \ge 4\sqrt{n\beta}$, which we refer to as the GapCoin(α, β) problem. We show the following:

Theorem 3. *Let M be a streaming algorithm (that might use private randomness) which outputs, with probability at least $2/3$, the answer 0 on every input with $|\sum_i x_i| \le 4\sqrt{n\alpha}$ and 1 on every input with $|\sum_i x_i| > 4\sqrt{n\beta}$. Then (for $\alpha \ge 1$ and $c_2 \le \beta \le \mathrm{poly}(\log n)$, where $c_2 > 0$ is a large enough constant)*

$$\sum_{i=1}^{n}\sum_{j=1}^{i} I(M_i; X_j | M_{j-1}) > \Omega(n \log n / (\beta \log^2 \beta)^{11}),$$

where X is drawn from U_n.

As hinted before, we observe that

$$\sum_{i=1}^{n}\sum_{j=1}^{i} I(M_i; X_j | M_{j-1}) \le \sum_{i=1}^{n} |M_i|.$$

Here, $|M_i|$ represents the size of the memory state M_i. Refer to Section II for more on the notation. This proves that solving the GapCoin(α, β) problem requires $\Omega\left(\frac{\log n}{(\beta \log^2 \beta)^{11}}\right)$ memory. We then look at the problem of solving the OR of k copies of the GapCoin(α, β) problem for $\log k \le \alpha \le \beta \le \log^2 k$. We refer to this as the k-OR PROBLEM. To elaborate, given k sequences of n $\{-1, +1\}$ bits $x_1^j, x_2^j, \ldots, x_n^j$, $j \in \{1, 2, \ldots, k\}$, output 0 if $\forall j,\ |\sum_i x_i^j| \le 4\sqrt{n \log k}$ and 1 if $\exists j \in [k],\ |\sum_i x_i^j| \ge 4\sqrt{n}\log k$. Using Theorem 3 and a direct sum on the information cost, under the uniform distribution over all the k instances, we show that solving the OR of multiple copies requires $\Omega\left(\frac{k \log n}{\mathrm{polylog}(k)}\right)$ bits of memory.

Theorem 4. *Let M be a streaming algorithm which outputs 0 with probability at least $3/4$ on every 0 input of the k-OR problem, and 1 with probability at least $3/4$ on every 1 input. Then there exists a constant $c > 0$[1] (for $c_3 < k < n$ where $c_3 > 0$ is a large enough constant) such that,*

$$\sum_{i=1}^{n}\sum_{j=1}^{i} (I(M_i; Y_j | M_{j-1})) > \Omega(k \cdot n \log n / (\log k \log\log k)^c).$$

Here, $\forall i \in [n], Y_i = (X_i^1, \ldots, X_i^k)$ is i.i.d. uniformly from $\{-1, +1\}^k$.

If the ratio β/α is sufficiently close to 1, we show how to remove the polylog(k) factor, obtaining an optimal $\Omega(k \log n)$ bit lower bound for this problem, which is useful for our applications.

We next give a number of new lower bounds and upper bounds in the data stream model.

B. Data Stream Applications

Our lower bound for the coin problem gives new lower bounds for a number of problems in different data stream models. We now introduce these models and problems.

The most general data stream model is the *turnstile* model, for which there is an underlying d-dimensional vector x, initialized to 0^d, which undergoes a long sequence of additive updates of the form $x_{i_j} \leftarrow x_{i_j} + \Delta_j$, where $i_j \in \{1, \ldots, d\}$, $\Delta_j \in \{-1, 1\}$ and (i_j, Δ_j) is the j-th stream update.

A less general, though often more realistic model, is the *bounded deletions model* for which one still has that the updates Δ_j can be positive or negative, but one is promised that the norm $\|x\|_2 = (\sum_{i=1}^d x_i^2)^{1/2}$ never drops by more than an α-fraction of what it was at any earlier point in the stream, for a parameter α. We will focus on constant α in this work.

We let x_j denote the number of occurrences, or *frequency* of item j. While it is impossible to store accurate approximations to all frequencies with a small amount of memory, there are many useful summaries that suffice for applications.

1) "Lifting" Lower Bounds to the Bounded Deletions Model: There are a large number of lower bounds known in the turnstile model but they involve very sudden drops to the norm of the underlying vector and consequently do not hold in the bounded deletions model. Using our lower bounds for the coin problem, for a number of applications we are able to obtain improved lower bounds in the bounded deletions model, matching those that previously were only known to hold in the turnstile model.

Estimating the maximum frequency.: This is denoted by $\|x\|_\infty = \max_{i \in \{1,\ldots,d\}} |x_i|$.

By a reduction from the so-called INDEX problem in 2-player communication complexity, this problem requires $\Omega(d)$ bits of memory to approximate up to a multiplicative factor of 2, even in random-order streams [13]. A common

[1] $c = 22$.

goal is to instead output an approximation to $\|x\|_\infty$ with additive error $\frac{1}{\sqrt{k}}\|x\|_2$ [14], [15], [16].

One can prove an $\Omega(k \log m)$ lower bound for turnstile streams using the standard AUGMENTED-INDEXING communication problem, see [5] for similar reductions. However, this lower bound inherently requires the norm of the underlying vector x to grow to $\mathrm{poly}(m)$ and then drop to 2^i for a random $i \in \{1, 2, \ldots, \log m\}$. These $\log m$ scales are precisely the source of the $\log m$ factor in the lower bound, while the k factor comes from having to solve a problem requiring k bits of information to solve at each scale. In the bounded deletions model one cannot reduce the norm of x by more than a constant factor and thus, only an $\Omega(k)$ lower bound is known and standard[2].

We resolve this question in the bounded deletions model, up to $\mathrm{poly}(\log k)$ factors. Our lower bound applies even to outputting an estimate Z with both additive and multiplicative error:

$$\|x\|_\infty - \frac{\|x\|_2}{\sqrt{k}} \le Z \le \gamma\|x\|_\infty + \frac{\|x\|_2}{\sqrt{k}} \qquad (1)$$

Theorem 5. *($\|x\|_\infty$-Approximation) Any streaming algorithm achieving (1) in the bounded deletions model with probability at least $2/3$, over its private randomness, requires $\Omega(k \log m)$ bits of memory if $\gamma = 1$, and $\Omega\left(\frac{k \log m}{\mathit{polylog}(k)}\right)$ bits of space for any $1 < \gamma \le \mathit{polylog}(k)$.*

Estimating the variance.: Another problem is to estimate the variance of the frequencies, or equivalently $\|x\|_2$. Here the goal is to output an estimate within $(1 \pm \epsilon)\|x\|_2$. It is known [5] that this requires $\Omega(\epsilon^{-2} \log m)$ bits of memory in the turnstile model. The proof again requires the underlying vector x to grow to $\mathrm{poly}(m)$ and then drop to 2^i for a random $i \in \{1, 2, \ldots, \log m\}$. These $\log m$ scales are the source of the $\log m$ factor in the lower bound, while the ϵ^{-2} factor comes from a single-scale lower bound.

In [17] it was asked if one could obtain a better upper bound for this problem in the bounded deletions model. The only known lower bound is $\Omega((\log(1/\alpha))/\epsilon^2)$ [17], given the promise that at each time in the stream, $\|x\|_2$ is never an α-fraction below of what it was at an earlier time in the stream.

We show the input stream we generate for the variance estimation problem, using the SIMULTANEOUS k-COINS PROBLEM with $k = \Theta(\epsilon^{-2})$, satisfies the bounded deletions property (with constant α) and we resolve the question in [17] in the negative.

[2]One can prove this via a reduction from the INDEX problem. In this problem, Alice has a vector $a \in \{0, 1\}^{k/9}$, and Bob has an index $j \in \{1, 2, \ldots, k/9\}$. In the stream, Alice presents the updates $x_i \leftarrow x_i + 1$ for each i for which $a_i = 1$. She sends the state of the streaming algorithm to Bob, who inserts $x_j \leftarrow x_j + 1$. If $a_j = 1$, then $\|x\|_\infty = 2$, otherwise $\|x\|_\infty = 1$. Note that $\frac{1}{\sqrt{k}}\|x\|_2 < \frac{1}{2}$, so from the approximate output Bob can deduce a_j and thus the space complexity of the streaming algorithm, which is Alice's message, must be at least $\Omega(k)$, the one-way communication complexity of the INDEX problem.

Theorem 6. *(Euclidean Norm Estimation) Any streaming algorithm in the bounded deletions model (with the α above being an absolute small enough constant) which outputs a number within $(1 \pm \epsilon)\|x\|_2$, with probability at least $2/3$ over its private randomness, requires $\Omega((\log m)/\epsilon^2)$ bits of memory.*

For this theorem, and throughout, we assume $\epsilon^{-2} \le \min\{m, d\}^{0.9}$, which is the most common setting.

The Point Query Problem.: Another related problem is the ℓ_2-Point Query Problem, in which one is given a single index j at the end of the stream and one would like to estimate x_j up to additive error $\epsilon\|x\|_2$ with constant probability over the private randomness, see, e.g., [18], [19].

Again this can be shown to require $\Omega(\epsilon^{-2} \log m)$ bits of memory in the turnstile model, using standard techniques as in [5]. However, again the $\log m$ factor occurs in the lower bound because of the need to have $\log m$ geometric scales and drastically shrink the norm of x at the end of the stream. This is also optimal given an $O(\epsilon^{-2} \log m + \log d)$ bit upper bound using the COUNTSKETCH data structure [20] (here we only need a constant number of rows in the data structure of [20], since we only need to be correct on a fixed index j. We can also first hash $\{1, 2, \ldots, d\}$ to a universe of size m^2 using a pairwise independent hash function specified with $O(\log(dm))$ bits.).

No lower bound better than $\Omega(\epsilon^{-2} + \log d)$ was known in the bounded deletions model. We are able to show an $\Omega(\epsilon^{-2} \log m)$ bit lower bound in the bounded deletions model.

Theorem 7. *(Point Query Problem) Any streaming algorithm which, in the bounded deletions model, solves the ℓ_2-Point Query Problem, requires $\Omega(\epsilon^{-2} \log m)$ bits of memory.*

The ℓ_2-Heavy Hitters Problem.: Let $F_2 = \|x\|_2^2 = \sum_{i=1}^d x_i^2$ be the second moment of the data stream. We consider the ℓ_2-heavy hitters problem.

Definition 1. *In the ℓ_2-Heavy Hitters Problem with parameter ϵ, one should output a set S which (1) contains all indices $i \in [d]$ for which $x_i^2 \ge \epsilon^2 \cdot F_2$, and (2) does not contain any $i \in [d]$ for which $x_i^2 \le \frac{\epsilon^2}{2} \cdot F_2$. Further, for all $i \in S$, one should output an estimate $\hat{x}_i$ with $|\hat{x_i} - x_i| \le \epsilon\|x\|_2$.*

In the bounded deletions model, it was observed in [17] that an $O(\epsilon^{-2}(\log(1/\epsilon))\log(dm))$ bits of space algorithm [21], [22] applies (for any constant α in the definition of the bounded deletions model). There is a trivial $\Omega(\epsilon^{-2} \log d)$ lower bound just to write down the identities of the potentially ϵ^{-2} many ℓ_2-heavy hitters. However, in the bounded deletions model, it was unknown if there was also an $\Omega(\epsilon^{-2} \log m)$ lower bound, which may be much larger than $\Omega(\epsilon^{-2} \log d)$ if $d \ll m$.

We show an $\Omega(\epsilon^{-2} \log m)$ lower bound in the bounded deletions model, which together with the trivial $\Omega(\epsilon^{-2} \log d)$

lower bound, implies the algorithms of [21], [22] are optimal up to a $\log(1/\epsilon)$ factor.

Theorem 8. *(ℓ_2-Heavy Hitters Problem) Any streaming algorithm which, in the bounded deletions model, solves the ℓ_2-Heavy Hitters Problem with sufficiently large constant probability, over the private randomness, requires $\Omega(\epsilon^{-2}\log m)$ bits of memory.*

2) Random Order Streaming Lower Bounds: Another well-studied model is the random order model. Here, as in the turnstile model, we allow both positive and negative updates (see Section I-B3 below for further discussion on the necessity of this for the following problems), though the order of the stream is not allowed to be worst case, but rather the stream updates arrive in a uniformly random order. Even in this model we are able to prove strong lower bounds for both the Point Query Problem and ℓ_2-Heavy Hitters Problem:

Theorem 9. *(Point Query Problem) Any streaming algorithm which, in the random order model, solves the ℓ_2-Point Query Problem with sufficiently large constant probability, over the random order and private randomness, requires $\Omega(\epsilon^{-2}\log m)$ bits of memory.*

Theorem 10. *(ℓ_2-Heavy Hitters Problem) Any streaming algorithm which, in the random order model, solves the ℓ_2-Heavy Hitters Problem with sufficiently large constant probability, over the random order and private randomness, requires $\Omega(\epsilon^{-2}\log m)$ bits of memory.*

3) On the Tightness of the Models in Our Lower Bounds: New Upper Bounds: For the problems above, we consider if stronger lower bounds are possible in different models.

In the more general turnstile model, in the 2006 IITK Workshop on Data Streams, in Open Problem 3, Cormode [3] asks whether it is possible to estimate $\|x\|_\infty$ up to additive error $\frac{1}{\sqrt{k}}\|x\|_2$ using fewer than $O(k(\log d)\log m)$ bits of memory, which was the previous upper bound.

Given that we have shown a lower bound of $\Omega(k\log m)$ for this problem in the bounded deletions model, it is natural to ask whether our lower bound can be improved to $\Omega(k(\log d)\log m)$ for turnstile streaming algorithms. We show this is impossible, by giving an algorithm for solving this problem and using $O(k(\log k+\log\log m)\log(dm))$ bits of memory, which works in the turnstile model and thus also the bounded deletions model, showing our lower bound is tight up to a $\log k+\log\log m$ factor.

Theorem 11. *(ℓ_∞-Estimtion) There is a turnstile streaming algorithm which approximates $\|x\|_\infty$ up to additive error $\frac{1}{\sqrt{k}}\|x\|_2$ with probability at least $2/3$ (over the private randomness) and using $O(k(\log(dm))(\log k+\log\log m))$ bits of memory.*

Another natural question that Theorem 9 and Theorem 10 raise is whether our lower bounds hold even in the more restrictive insertion-only model. Recall that stream updates have the form $x_{i_j}\leftarrow x_{i_j}+\Delta_j$ in general, and the insertion-only streaming model requires $\Delta_j=1$ for all j.

We show this is impossible in the insertion-only model, at least if the stream updates are randomly ordered. Recalling that our lower bounds Theorem 9 and Theorem 10 hold in the random order model, this shows that our lower bounds in Theorem 9 and Theorem 10 require deletions to be allowed as well.

To show this, we give a new algorithm which solves both the ℓ_2-Heavy Hitters problem and the ℓ_2-Point Query Problem in the insertion-only model, for randomly ordered streams.

Theorem 12. *(ℓ_2-Heavy Hitters and ℓ_2-Point Query Problems) There is a streaming algorithm in the insertion-only model, with randomly ordered updates, which, with probability at least $2/3$ (over the private randomness and the random order), solves the ℓ_2-Heavy Hitters problem and ℓ_2-Point Query Problem. For the ℓ_2-Heavy Hitters problem, the memory is $O(\epsilon^{-2}(\log d+\log^2(1/\epsilon)+\log^2\log m)+\log m)$ bits. For the ℓ_2-Point Query Problem, the memory is $O(\epsilon^{-2}(\log^2(1/\epsilon)+\log^2\log m)+\log m+(\log(1/\epsilon)\log d))$ bits.*

For the ℓ_2-Heavy Hitters problem, our estimates $\hat{x_i}$ for those i in our output set S satisfy the stronger guarantee that $\hat{x_i}=(1\pm\epsilon)x_i$ (see the full version of the paper[4] for details).

Notice that our algorithm for ℓ_2-heavy hitters bypasses the $\Omega(\epsilon^{-2}\log m)$ lower bound for bounded deletions for $d\ll m$. Theorem 10 also gives a separation between the turnstile model, with randomly ordered updates, and insertion-only models with randomly ordered updates. The above theorem is stated with error probability $1/3$ for simplicity, though can be generalized to any constant failure probability.

Also, note that the previous best algorithm for heavy hitters in insertion streams, even if one assumes a random order, was $O(\epsilon^{-2}(\log 1/\epsilon)\log(dm))$ bits of memory. In addition to a worse memory bound, the previous algorithm was not able to obtain the stronger guarantee that $\hat{x_i}=(1\pm\epsilon)x_i$ for all i in the output set.

In the Warwick Workshop on Data Summarization, Jelani Nelson explicitly poses the question for insertion-only streams[5], if one can achieve $O(\epsilon^{-2}\log(dm))$ bits of memory in insertion-only streams, namely, if the $O(\log 1/\epsilon)$ factor can be removed from the upper bound. Our Theorem 12

[3]https://www.semanticscholar.org/paper/OPEN-PROBLEMS-IN-DATA-STREAMS-AND-RELATED-TOPICS-ON-Agarwal-Baswana/5394ab5bf4b66bfb52f111525d6141a3226ba883.

[4]https://eccc.weizmann.ac.il/report/2020/139/.

[5]See Slide 86 here: https://warwick.ac.uk/fac/sci/dcs/research/focs/conf2017/abstracts/jn-slides.pdf

shows that this is indeed possible in insertion-only streams, at least when the stream updates are randomly ordered. Indeed, under the standard setting of parameters when $\log d = \Omega(\log^2(1/\epsilon) + \log^2 \log m)$, our algorithm gives $O(\epsilon^{-2} \log d + \log m)$ bits of memory, significantly improving the earlier $O(\epsilon^{-2}(\log(1/\epsilon)\log(dm)))$ bit algorithm.

C. Our Techniques

We start by describing the techniques involved in proving our main lower bound for the coin problem. For detailed proofs, see the full version (https://eccc.weizmann.ac.il/report/2020/139/).

1) Technical Overview of our Lower Bound for the Coin Problem: The starting point for our lower bounds is Theorem 1, which states that a streaming algorithm that computes the majority of n bits with a constant probability (≥ 0.999) requires $\Omega(\log n)$ bits of memory. Note that simple counting gives a streaming algorithm that uses exactly $\log n$ bits of memory.

The intuition for the lower bound is that a successful streaming algorithm will need to "remember" $\Omega(1)$ bits of memory about each "scale" of its past bits. At step i the memory state M_i will remember $\Omega(1)$ bits about each of the blocks $\{X_i\}$, $\{X_{i-2..i-1}\}$, $\{X_{i-6..i-3}\}$, $\{X_{i-14..i-7}\}$, etc. Here the notation $X_{a..b}$ is used to represent the input bits $X_a, X_{a+1}, \ldots, X_b$. There are two main challenges in realizing this approach: (1) coming up with the correct information-theoretic formalism (the definition of "remember"), which is sufficiently strong for memory lower bounds and downstream applications; (2) proving the actual lower bound on the information remembered.

Sidestepping the first challenge for a moment, let us sketch the proof of the lower bound on the information remembered. Let us focus at scale 2^j. Let $t = n/2^j$, and let $S_1, \ldots, S_t$ be the sums of each of the t blocks of X_i's (so that $S_m = \sum_{\ell=(m-1)\cdot 2^j+1}^{m\cdot 2^j} X_\ell$). $M_{m\cdot 2^j}$ represents the memory state of the the streaming algorithm after the m-th block.

If $M_{m\cdot 2^j}$ contains $\delta \ll 1$ information about S_m (conditioned on $M_{(m-1)\cdot 2^j}$), then conditioned on $M_{m\cdot 2^j}$ the sum S_m has almost full variance (i.e., $\mathrm{Var}(S_m | M_{m\cdot 2^j}, M_{(m-1)\cdot 2^j}) > 2^j \cdot (1-\varepsilon)$). Conditioned on the memory states $M_{m\cdot 2^j}$, $m \in [t]$, the sums S_m become independent. Therefore, if $M_{m\cdot 2^j}$ contains $\delta \ll 1$ information about S_m (conditioned on $M_{(m-1)\cdot 2^j}$) for the vast majority of m's, then conditioned on the $M_{m\cdot 2^j}$s, the variance of the sum $\sum S_m = \sum X_i$ remains close to the full variance n, which means that the algorithm gains no advantage in estimating the majority of the X_i's.

One complication of the above outline is that it shows that M_i must have $\Omega(1)$ information on average about the block $X_{i-2^j+1..i}$. However, this range actually contains j different scales. For the argument to work we need M_i to contain $\Omega(1)$ information on average about $X_{i-2^j+1..i-\alpha\cdot 2^j}$ for some constant $\alpha > 0$. We prove this — essentially by showing that M_i cannot have $\Omega(1)$ information about the sum of 2^j bits (low variance in the sum) if it only has information about the last $\alpha \cdot 2^j$ of them. Putting those together, we get that M_i has $\Omega(1)$ bits of information about $\Omega(\log n)$ disjoint blocks of X's (the different scales).

It is important to choose the correct conditioning, while defining the "information" M_i knows about $X_{\leq i}$. Consider the following simple example: the streaming algorithm M samples $n/10^4$ $\{\pm 1\}$ bits and starts the counter with their sum, that is, $M_1 \sim \mathrm{Bin}(n/10^4, 1/2)$ (this is the binomial distribution over $n/10^4$ uniformly random ± 1 bits). Then M keeps adding the input stream to the counter, that is $M_i = M_{i-1} + X_i$. M outputs $\mathrm{sign}(M_n)$. It is not hard to see that this algorithm computes the majority of the X_i's correctly with a high constant probability. However, M_i contains almost no information about X_i (only $O(1/n)$). In fact, the information between M_i and $X_{\leq i}$ ($I(M_i; X_{\leq i})$) is $O(1)$. And hence, $\sum_i I(M_i; X_{\leq i})$ is $O(n)$. However, we show that for deterministic algorithms, even this information quantity works to show an $\Omega(\log n)$ lower bound.

For randomized algorithms that use fresh randomness at every time step, to bypass the above example, when measuring the amount of information M_i has about X_i, we should condition on M_{i-1} — this captures the fact that when creating M_i out of M_{i-1} we must store a bit capturing the value of X_i. More generally, when measuring the mutual information between M_i and the block $X_{i-2^j+1..i-\alpha\cdot 2^j}$, we condition on M_{i-2^j}. It appears that the most natural way to formalize this conditioning is by defining the total information cost of a streaming algorithm as

$$IC(M, U_n) := \sum_{i=1}^{n} \sum_{j=1}^{i} I(M_i; X_j | M_{j-1}). \tag{2}$$

Here, U_n represents the uniform distribution over $\{+1, -1\}^n$ and X is drawn from U_n. It will be interesting to see whether this quantity finds other streaming applications. For computing the majority, the discussion above implies that, on average

$$I(M_i; X_j | M_{j-1}) = \Omega\left(\frac{1}{i-j+1}\right).$$

Coming back to the first challenge, the quantity $IC(M, U_n)$ from (2) is bounded from above by $\sum_{i=1}^{n} H(M_i)$, and can be used in direct sum contexts (with private randomness).

One of the key insights of our overall strategy is using variance as a convenient measure of uncertainty of the sum of bits, and relating it to how much information is revealed about the input. We prove the following, and Theorem 1 follows as a corollary.

Theorem 13 (Variance-Information Tradeoff)**.** *For any $\varepsilon > c' n^{-\frac{1}{20}}$, there exists $\delta \geq c''' \epsilon^5$ (for a small enough constant*

$c''' > 0$ *and a large enough constant* $c' > 0$*) such that if*

$$\sum_{i=1}^{n}\sum_{j=1}^{i} I(M_i; X_j | M_{j-1}) \le \delta n \log n,$$

then

$$\mathbb{E}_{m_n \sim M_n} \operatorname{Var}\left[\sum_{i=1}^{n} X_i \;\middle|\; M_n = m_n\right] \ge (1/4 - \varepsilon) \cdot n.$$

2) A Direct Sum Theorem and the OR of Many Copies: In this section, we give a brief overview of our lower bounds for two generalizations of the coin problem. First, we consider *a direct sum theorem* for solving the coin problem for k copies. Before we delve into the lower bound for the problem defined in Theorem 2, let us consider the following problem P_1: M sees k instances of the coin problem $(Y^1, Y^2, \ldots, Y^k)$ in a stream as follows: at the i-th step, M sees the i-th input bit for each instance, that is, $Y_i^j, \forall j \in [k]$. At the end, M is given a random index $\ell \in [k]$ and is asked to output the majority of the ℓ-th instance. A slight variant of the Problem P_1 turns out to be a very important special case for our applications, and we refer to this as the SIMULTANEOUS k-COINS PROBLEM.

The notion of information cost as defined in (2) suffices to prove an $\Omega(n \log n)$ information lower bound for streaming algorithms that use private randomness. Therefore, a direct sum theorem can be readily used to prove that any streaming algorithm M that solves P_1 with probability at least 0.9999 requires $\Omega(k \cdot n \log n)$ information cost. A direct sum theorem just uses the fact that

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{l=1}^{k} I(M_i; Y_j^l | M_{j-1} Y_j^{<l}) \ge \sum_{l=1}^{k}\left(\sum_{i=1}^{n}\sum_{j=1}^{i} I(M_i; Y_j^l | M_{j-1})\right) \quad (3)$$

(because Y_j^l is independent of $Y_j^{<l}$) given M_{j-1}). Therefore, for at least half fraction of ls,

$$\sum_{i=1}^{n}\sum_{j=1}^{i} I(M_i; Y_j^l | M_{j-1}) \le \frac{2}{k}\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{l=1}^{k} I(M_i; Y_j^l | M_{j-1} Y_j^{<l}). \quad (4)$$

And for at least $9/10$ fraction of ls, the success probability is at least 0.999. Let l' be an l in the intersection, Using M, one can obtain a streaming algorithm for solving a single instance of the coin problem, which uses private randomness to generate the other instances (the fact that $\forall j, Y_1^j, Y_2^j, \ldots Y_n^j$ are independently drawn is crucial for this step), has information cost equivalent to $\sum_{i=1}^{n}\sum_{j=1}^{i} I(M_i; Y_j^l | M_{j-1})$ and success probability of at least 0.999.

We stress that the direct sum theorem here is subtle. For example, we cannot give the streaming algorithm all updates to instances ℓ' for $\ell' > \ell$ at the end of the stream, and still prove a direct sum theorem, which a priori one might suspect since it is reminiscent of techniques in 2-player communication complexity used to prove lower bounds for the AUGMENTED-INDEXING problem [5]. Indeed, one can show that if $k = \log n$ and one is given all updates to all coins $\ell' > \ell$ at the end of the stream, then $O(\log n)$ bits suffice to solve this problem, rather than the $\Omega(\log^2 n)$ bits one would expect from a direct sum of $\log n$ copies. Indeed, we leave it to the reader to check that the algorithm which replaces 0 updates to the j-th coin by -10^{4j} and 1 updates to the j-th coin by 10^{4j} in the stream, and maintains a single counter of the sum of all $O(\log n)$ scaled updates across all k coins, and then subtracts all (scaled) updates to all coins $\ell' > \ell$ at the end of the stream, can be used to determine the bias of the ℓ-th coin with only $O(\log n)$ bits of memory.

Things get trickier for the problem defined in Theorem 2 (P_2). In P_2, M is not given all the k instances in parallel, but randomly interleaved in a stream of kn independent updates, of the form $Y_j = (X_j, S_j)$, where S_j is chosen uniformly in $\{1, 2, \ldots, k\}$ and X_j is chosen uniformly in $\{0, 1\}$. Proving a lower bound for this version of the k-Coins Problem is crucial for the streaming applications downstream.

The proof of Theorem 2 is more challenging than the direct sum theorem mentioned above. One of the difficulties faced is that the number of updates in P_2 is kn instead of n, and hence to prove an $\Omega(k \log n)$ memory lower bound, we need an $\Omega(k^2 \cdot n \log n)$ information lower bound on M. To prove the theorem, we first divide the information cost of the streaming algorithm into information cost for the individual instances as follows.

$$\sum_{i=1}^{nk}\sum_{j=1}^{i} I(M_i; X_j | M_{j-1}) = \sum_{s \in [k]}\sum_{i=1}^{nk}\sum_{j \in [i] \text{ and } S_j = s} I(M_i; X_j | M_{j-1})$$

Then we show, through a non-standard use of the data processing inequality, that a streaming algorithm that solves the k-COINS PROBLEM even when S_js are fixed as $S_j = (j-1) \mod k + 1$, requires $\Omega(k \cdot n \log n)$ information cost each for a large fraction of individual instances, and a direct sum can be applied henceforth. In fact, we can show the above information cost lower bound for a typical sequence $\{S_j\}_{j \in [nk]}$ when the S_j are *i.i.d.* uniform in $[k]$, hence, proving a $\Omega(k \log n)$ memory lower bound and Theorem 2.

We next consider the second generalization, that is, a memory lower bound for solving the OR of k coin problems, that is, output 1 if the majority bit of any of the k instances is 1 and 0 otherwise. In communication complexity, the most famous information lower bound for a communication problem that solves the OR of n instances, is that for two-party set-disjointness. [23] gave a simple lower bound of $\Omega(n)$ on the communication required to solve set-disjointness, using a direct sum theorem. There are three steps to using a direct sum theorem to prove such a lower bound: (1) define the information cost for a communication protocol that decomposes into a sum of information costs on single instances; (2) show that an algorithm for solving the OR can be used for solving a single instance; and (3) show that the information cost for solving a single instance is large. Set-disjointness is equivalent to solving the OR of n AND instances. The authors of [23] complete these three steps by lower bounding the information cost of any communication protocol that solves a single instance (AND of two bits), on an input distribution which is supported on $\{00, 01, 10\}$. However, our techniques can only be used to lower bound (2) when the input $X = (X_1, X_2, \ldots, X_n)$ to the coin problem is drawn from a product distribution as M is only allowed to use private randomness.

We sidestep the above issue by changing the underlying coin problem to the following (P_3). Consider the following problem: given a sequence of n $\{-1, +1\}$ bits X_1, X_2,..., X_n, output 0 if $|\sum_i X_i| \leq 4\sqrt{n\alpha}$ and 1 if $|\sum_i X_i| \geq 4\sqrt{n\beta}$. We refer to this as the GapCoin(α, β) problem. We will always set $\alpha = \log k$ and prove our bounds for $4\alpha \leq \beta = O(\log^2 k)$. To describe our results, we focus on the case $\alpha = \log k$ and $\beta = 4\alpha$; the general proof is similar. For a sequence of n ± 1 integers, let S denote their sum. We say that the instance is a 0-instance if $|S| \leq 4\sqrt{n \log k}$ and a 1-instance if $|S| \geq 8\sqrt{n \log k}$. The crucial fact is that: when S is the sum of n i.i.d. uniform ± 1 integers, then with high probability, $|S| \leq 4\sqrt{n \log k}$. Thus, the probability of a 0 instance is high under the uniform distribution on $\{-1, +1\}^n$. Hence, an algorithm A for solving the OR of k P_3 problems, can be used to develop an algorithm A' for a single instance of P_3, where A' privately generates the other instances, to be given to A, one step at a time. Also, whenever A outputs a 0, the answer to P_3 is definitely 0 and whenever A outputs 1, with high probability, the other instances all evaluate to 0, and hence, the answer to P_3 is also 1. This completes Step (2). Step (1) is easy to show and follows similarly to the discussion given for our direct sum theorem for P_1. Step (3) is what requires significant technical work. We prove that any streaming algorithm that solves P_3 requires $\Omega\left(\frac{\log n}{(\log k \log\log k)^{22}}\right)$ bits of memory. To give a glimpse of our proof, we partition our stream into $t = (\log k \log\log k)^2$ blocks and use a corollary of Theorem 1 on n/t sized blocks (the following theorem), which might be of independent interest.

Theorem 14. *Let B be a streaming algorithm (that might use private randomness) of length m. Then, for $\epsilon > 0$, one of two things holds:*

1) *Either $IC(B, U_m) = \Omega(m \log m \cdot \epsilon^{10})$.*
2) *Or there exists a distribution μ on m bits, such that*
   - *$\forall x \in \{-1, +1\}^m, \mu(x) \leq 2U_m(x)$,*
   - *$E_{x \sim \mu}[\sum_i x_i] \geq \Omega\left(\sqrt{\frac{m}{\log(1/\epsilon)}}\right)$, and*
   - *$\|B(U_m) - B(\mu)\|_1 < \epsilon$.*

Recall that U_m represents the uniform distribution over $\{-1, +1\}^m$. Thus, for a streaming algorithm that has low information cost, there exists a distribution μ, which has a significantly higher number of ones, on average, compared to the uniform distribution, but B cannot distinguish between μ and the uniform distribution. Informally, to construct μ, we strategically shift some weight from $-x$ to x, and show that if B could distinguish between μ and uniform, it would have "solved majority" with low information cost. Next, we show that if there is a low information cost streaming algorithm that solves P_3, then it should be able to distinguish between μ^t and U_m^t, which would contradict Theorem 14.

When the ratio of $\frac{\beta}{\alpha}$ is sufficiently close to 1 in the GapCoin(α, β) problem, which is important for our applications, we give a more direct proof of the k-OR problem, which we call the k-OR small gap problem. The proof is via a direct reduction from the SIMULTANEOUS k-COINS PROBLEM. Given $\ell \in [k]$ at the end of the stream in an instance of the SIMULTANEOUS k-COINS PROBLEM, we can add $4\sqrt{n \log k}$ updates to the ℓ-th coin. This makes the sum of the ℓ-th coin way more than the average, and by making the ratio $\frac{\beta}{\alpha}$ subconstant (though still large enough to be meaningful in the applications), determining the solution to the GapCoin(α, β) problem decides the majority bit of the ℓ-th coin.

3) Technical Overview of our Algorithms: Refer to the full version[6] for the details.

ℓ_∞-Estimation.: Our algorithm is inspired from a universe reduction technique of [24], which shows for $0 < p < 2$, there is a randomized oblivious linear map which takes a vector x and produces a vector y so that with good probability, if i is a so-called ℓ_p-heavy hitter of x, then $y_{h(i)}$ is an ℓ_p-heavy hitter of y, where h corresponds to a hash function defining y. The oblivious linear map is nothing other than a COUNTSKETCH map, with stronger randomness guarantees than what is usually needed. Here we need to argue this holds for $p = 2$, which in a certain sense simplifies the analysis of [24], but on the other hand, we also need to argue that there are no entries of y that are heavy but do not correspond to any heavy hitter in x. We need this because we need $\|y\|_\infty \approx \|x\|_\infty$, which was not necessary in [24].

After applying the universe reduction, we then feed y into a COUNTSKETCH data structure itself to find all of the ℓ_2-heavy hitters of y together with good approximations to their frequencies. Taking the maximum frequency found gives us a good approximation to $\|y\|_\infty$, and in turn to $\|x\|_\infty$. This can all be done in one pass because the universe reduction

[6]https://eccc.weizmann.ac.il/report/2020/139/.

and CountSketch maps are both linear and oblivious; hence, so is their composition. The CountSketch map uses $O(k(\log d)\log m)$ bits of space and finds the ℓ_2-heavy hitters, but since we are applying it to the vector y, which has dimension $\mathrm{poly}(k\log d)$, we obtain overall $O(k(\log k + \log\log d)\log m)$ bits of space.

ℓ_2-Heavy Hitters and ℓ_2-Point Query.: Our algorithm for ℓ_2-Point Query is a slight modification to our algorithm for the ℓ_2-Heavy Hitters problem, so we only describe the latter. Our algorithms for both problems do not use the Gaussian process and chaining-based arguments in the previous $O(\epsilon^{-2}(\log(1/\epsilon))\log(dm))$ space algorithms [21], [22]. Rather, the arguments are elementary, which could be of independent interest.

Our key subroutine Main1 assumes we know F_2 and m in advance (m is the number of elements in the stream). Our actual algorithm Main runs a constant-factor F_2-estimate on the side and invokes Main1 with this F_2-estimate. It creates a new instance of Main1 each time the stream length doubles, and only maintains two instances of Main1 at a time, the current one starting from when m last doubled, and the previous one which completed the last time m doubled. This enables us to reuse space. Since Main1 ends up being run on a stream of length 2^i for some known value of i with $2^i = \Theta(m)$, and since our F_2-estimate is a good enough constant factor approximation, by using the random order property of the stream we can argue that the heavy hitters found by invoking Main1 on this substream contain the actual heavy hitters for the entire stream, and no items that are sufficiently far from being heavy in the actual stream.

The idea behind our Main1 algorithm is to partition the stream into consecutive and contiguous equal-sized blocks $B_1, \ldots, B_t$, where $t \approx \epsilon\sqrt{F_2}$. This means that each B_i has size about $\epsilon^{-1}m/\sqrt{F_2}$. An important idea is that for each item $j \in [d]$, we randomly assign $O(\log(1/\epsilon))$ numbers $h^1(j), \ldots, h^{O(\ln(1/\epsilon))}(j) \in \{1, 2, \ldots, t\}$ to it. A key observation is that if $f_j^2 \geq \epsilon^2 F_2$, or equivalently $f_j \geq \epsilon\sqrt{F_2}$, then because the stream is randomly ordered, j has a constant probability of appearing in each block B_i. Moreover, it has probability $1 - O(\epsilon^2)$ of appearing in a B_i for which i is one of the $O(\log(1/\epsilon))$ numbers which is assigned to j. We say:

Definition 2. *An item $j \in [d]$ is* **excited** *in B_i if j occurs in block B_i and if i is one of the $O(\log(1/\epsilon))$ numbers which is assigned to j.*

Since there are only $O(\epsilon^{-2})$ total heavy hitters, by a union bound each heavy hitter is excited in some B_i. Moreover, one can also show that the set S^i of items j for which both $j \in B_i$ and $h^\ell(j) = i$ for some ℓ, $1 \leq \ell \leq O(\log(1/\epsilon))$, has size $O(\epsilon^{-2}\log(1/\epsilon))$ with good probability; in particular by a union bound with constant probability, simultaneously every heavy hitter is in such a set.

We next reduce S^i to at most a single element by looking at the next block B_{i+1}. We cannot afford to store S^i because it has size $O(\epsilon^{-2}\log(1/\epsilon))$ and each item identifier requires $O(\log d)$ bits. Instead, we choose another hash function g which maps the elements of S^i to a range of size $\mathrm{poly}(\epsilon^{-1}\log m)$, and we only store the hash, under g, of the items in S^i. For each of these hashed identities $g(j)$, we check if there is a unique item j' in B_{i+1} for which $g(j') = g(j)$ and for which $h^q(j') = i$ for some $q \in [10\log(1/\epsilon)]$. It is crucial that we also filter by ensuring that $h^q(j') = i$ for some $q \in [10\log(1/\epsilon)]$, as otherwise since the blocks are polynomially (in m) large there will be elements j' for which $g(j') = g(j)$, since the range of g is only $\mathrm{poly}(\epsilon^{-1}\log m)$. However, almost all of the items j in these next blocks are not excited in B_i.

A crucial property here is that for an actual heavy hitter j, when it is excited in a block B_i, it will also have good probability of occurring in the next block. Since there are $O(\log(1/\epsilon))$ values of q, we can make sure this happens with probability $1 - \mathrm{poly}(\epsilon)$ for one of the blocks that j is excited in. Further, we can ensure with good probability j is the unique item with this property. The intuition is that items with very low frequency are very unlikely to be excited in a block B_i, and then occur in the very next block. On the other hand, there are only a small number of items with large frequency, and these items will all be excited in blocks far away from each other since they are each only excited in a few random blocks.

An issue is that since there are $m^{\Omega(1)}$ blocks for which we start tracking the S^i, there will still be blocks not containing a heavy hitter which pass the above test, meaning that after inspecting B_{i+1}, there will be exactly one hashed identity with count equal to 1. Indeed, this happens with $\mathrm{poly}(\epsilon)$ probability. To counter this, we run a subroutine Identify to take the single hashed identifier which passed the above test for a given block and track the actual (unhashed) identifier over the next $O(\epsilon^{-2}\log m)$ blocks, counting the number of times the item occurs. The first issue is we do not know the actual identifier at this point, since we could not afford to store it when creating the set S^i corresponding to the block B_i where this identifier occurred. We first look at the next $100\log(1/\epsilon)$ blocks in our Identify procedure to recover the actual identifier with probability $1-\mathrm{poly}(\epsilon)$. This probability is large enough to ensure for all blocks which actually passed the test and contained a single heavy hitter, that we recover the corresponding heavy hitter. However, this probability is not enough to rule out false positives. However, now that we have an actual identifier, we count its number of occurrences over the next $4000\epsilon^{-2}\log m = O(\epsilon^{-2}\log m)$ blocks. At this point, the probability a non-heavy hitter occurs in a significantly large fraction of these blocks is $1/\mathrm{poly}(m)$, which is small enough that we can ensure there are no false positives.

One issue which arises with tracking an actual $O(\log d)$-bit identifier over the next $O(\epsilon^{-2}\log m)$ blocks, is that we may be tracking up to $O(\epsilon^{-2}\log m)$ actual identifiers at any given time, and this requires $\Theta(\epsilon^{-2}(\log d)\log m)$ bits of space, which is too large. We fix this by enforcing that if we are ever already tracking an actual identifier, then we do not begin tracking a new one. This may affect correctness now, since we decide to not track some actual identifiers, and therefore may not find one of the heavy hitters. Fortunately, because of the random order property, the locations of the heavy hitters are spread out in the stream, and one can show under our condition that $m \geq \mathrm{poly}(\epsilon^{-1}\log(d))$ (as otherwise there is a simpler algorithm we describe), that we never run into this problem.

One last problem is that we may start tracking a non-heavy hitter in the $4000\epsilon^{-2}\log m$ blocks preceding where we start tracking a heavy hitter j, and therefore since we have already invoked IDENTIFY, we may decide not to start tracking item j. Note that if the probability of tracking the actual identifier of a non-heavy hitter were only $\mathrm{poly}(\epsilon)$, and say, ϵ were not too small, then we almost certainly would start tracking a non-heavy hitter among the $4000\epsilon^{-2}\log m$ blocks preceding that of a heavy hitter, and therefore decide not to start tracking the heavy hitter. Fortunately though, the probability of tracking the actual identifier of a non-heavy hitter can be shown to be at most $\mathrm{poly}(\epsilon/\log(dm))$, for a large enough polynomial, and this probability is small enough to argue that for every heavy hitter, for any of the $4000\epsilon^{-2}\log m$ blocks preceding it, we do not start tracking an actual identifier of some other item. Thus, when we process the block containing the heavy hitter, we are free to start tracking its actual identifier.

II. Preliminaries

We define some notation and formalize certain concepts to better understand the introduction. For $a, b \geq 0$, $\sqrt[a]{b}$ represents $b^{1/a}$. For a positive integer n, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. Mostly through the paper, we will use capital letters to denote random variables, for example, X and Y. Given a sequence of random variables, $X_1, X_2, \ldots, X_n$, we use the notation $X_{a..b}$ (for $1 \leq a < b \leq n$) to represent the set $\{X_a, X_{a+1}, \ldots, X_b\}$ of random variables. Further, we use $X_{\leq i}$, $X_{>i}$, X_{-i} and $X_{a..b\cap S}$ to represent the set $\{X_j\}_{j\leq i}$, $\{X_j\}_{j>i}$, $\{X_1, \ldots, X_n\} \setminus \{i\}$ and $\{X_j\}_{j\in[a,b]\cap S}$ of random variables respectively (if the random variable X_0 is non empty, then the sets are defined according starting with X_0). We sometimes use these notations in superscripts when the indexing of the random variables is done in the subscript. Given a random variable X, we use the notation $x \sim X$ to denote the process that X takes value x with probability $\Pr[X = x]$. We use the notation $x \in_R S$ to represent that x is drawn uniformly at random from the set S. We use $\mathrm{Var}[X|Y]$ and $\mathrm{Var}[X|Y = y]$ to denote the expected variance of the random variable X conditioned on Y, and conditioned on the event $Y = y$, respectively. To represent the distribution of X conditioned on $Y = y$, we will use the notation X_y. Throughout the paper, we will denote bits by the sets $\{0, 1\}$ and $\{-1, +1\}$ interchangeably. We assume $\log$ is base 2 unless specified otherwise.

We use Stirling's approximation [25] for factorials in the paper, which states that[7]

$$\sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n} \leq n! \leq en^{n+\frac{1}{2}}e^{-n}.$$

We use U_n to represent the uniform distribution on $\{-1, +1\}^n$. $\mathrm{Ber}(p)$ represents the distribution on $\{0, 1\}$ with the probability of 1 being p. $\mathrm{Bin}(k, p)$ denotes the Binomial distribution over k bits where each bit is 1 with probability p and 0 otherwise. $\mathrm{Bin}(k, p)$ is just the distribution of the sum of k i.i.d. (independently and identically distributed) random variables from $\mathrm{Ber}(p)$. We use the following facts about the binomial distribution in the paper.

We also use the following fact about the entropy of the $\mathrm{Bin}(n, 1/2)$ distribution [26]:

$$H(\mathrm{Bin}(n, 1/2)) = \frac{1}{2}\log_2(\pi \cdot e \cdot n/2) + O(1/n).$$

Next, we give a brief prelude to the tools used from information theory. For a probability distribution $P\colon \mathcal{X} \to [0, 1]$, $H(P) = \sum_{x\in\mathcal{X}} P(x)\log_2(1/P(x))$ represents the entropy of the distribution P. We also use $H(X)$ to denote the entropy of the the random variable X. $I(X; Y|Z)$ represents the mutual information between X and Y conditioned on the random variable Z. $I(X; Y|Z) = H(X|Z) - H(X|Y, Z)$, where $H(X|Y) = \mathbb{E}_{y\sim Y}H(X|Y = y)$. Next, we describe some of the properties of mutual information used in the paper.

1) (Chain Rule) $I(AB; C) = I(A; C) + I(B; C|A)$.
2) If $I(D; B|A, C) = 0$, then $I(A; B|C) \geq I(A; B|C, D)$.
3) If $I(D; B|C) = 0$, then $I(A; B|C) \leq I(A; B|C, D)$.

Property 1 follows from the chain rule for entropy (H). Properties 2 and 3 follow from the observation that

$$\begin{aligned} I(A; B|C) + I(D; B|A, C) &= I(AD; B|C) \\ &= I(D; B|C) + I(A; B|C, D). \end{aligned}$$

As mutual information is non-negative, if $I(D; B|A, C) = 0$, then $I(A; B|C) \geq I(A; B|C, D)$ (because $I(D; B|C) \geq 0$) and if $I(D; B|C) = 0$, then $I(A; B|C) \leq I(A; B|C, D)$.

Given two distributions $P, Q\colon \mathcal{X} \to [0, 1]$, the KL-Divergence from Q to P is defined as

$$\mathbb{D}\left(P\|Q\right) = \sum_{x\in\mathcal{X}} P(x)\log_2\left(\frac{P(x)}{Q(x)}\right).$$

Given two distributions $P, Q\colon \mathcal{X} \to [0, 1]$, $\|P - Q\|_1 = \sum_{x\in\mathcal{X}} |P(x) - Q(x)|$ represents the distance between the

[7]Here, e, known as the Euler's number, is a mathematical constant, and the base of the natural logarithm.

two distributions. We use Pinsker's inequality[8], which states that,

$$\|P - Q\|_1 \leq \sqrt{2\,\mathbb{D}\,(P\|Q)}.$$

We will also use the following relationship between mutual information $I(X;Y)$ and the KL-Divergence, which can be easily verified by plugging in the definitions:

$$I(X;Y) = \mathbb{E}_{y\sim Y}\mathbb{D}\,(X_y\|X)\,.$$

In words, the mutual information between random variables X and Y is the expected KL-Divergence from the distribution X to the distribution of X conditioned on $Y = y$.

We will also make use of the following chaining inequality from [22]:

Lemma 1. *(Lemma 9 of [22], restated) If $k = \Omega(1/\epsilon^2)$, and Π is a $k \times n$ matrix of i.i.d. random variables uniform in $\{+1, -1\}$ (or even 8-wise independence suffices), then*

$$\mathbf{E}[sup_t\ |\|\Pi f^{(t)}\|_2^2 - k\|f^{(t)}\|_2^2|] \leq \epsilon k \|f^{(m)}\|_2^2,$$

where $f^{(1)}, \ldots, f^{(m)}$ are the frequency vectors of an n-dimentional insertion-only stream. Here $f^{(t)}$ represents the frequency vector after the first t updates.

Here, $\|x\|_2$ represents the ℓ_2-norm of x, that is, $\|x\|_2^2 = \sum_i x_i^2$.

Next, we define some of the terms used in the paper in relation to streaming algorithms. Let M denote the sequence of states of a streaming algorithm on a stream of length n, defined as follows: M_i represents the memory state of the algorithm after i steps. We can treat M_i as a random variable that depends on the input stream and the randomness used by the streaming algorithm. We say M uses *private randomness*, if at all time steps, it only has access to independent sources of randomness. In fact, we prove lower bounds for all sequences of messages (random variables) $M = M_0, M_1, ..., M_n$ of the form $M_i = \mathcal{M}_i(M_{i-1}, X_i, R_i)$ where R_i is the private randomness used at step i. Here, $\mathcal{M}_i$ can be an arbitrary function of M_{i-1}, X_i and the R_is, and does not need to be defined by a low-space streaming algorithm. For the sake of our lower bounds, we will refer to streaming algorithms as encompassing such sequences of messages. We simulate the streaming algorithm M by an n-party communication protocol, where the i-th party, denoted $\mathcal{P}_i$, receives the i-th input of the stream. Communication is allowed only from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$ for all $i \in [n-1]$. To obtain an n-party protocol from a streaming algorithm M, given X_1 (the first input of the stream), $\mathcal{P}_1$ can simulate the first step of the streaming algorithm by using private randomness and X_1. After the simulation, $\mathcal{P}_1$ sends the memory state m_1 as its message to $\mathcal{P}_2$, and so on. Given m_i and X_{i+1}, $\mathcal{P}_{i+1}$ can simulate the $(i+1)$-st step of M using private randomness and send m_{i+1} as its message to $\mathcal{P}_{i+2}$. $\mathcal{P}_n$ outputs what the streaming algorithm outputs. We define a **new notion of information cost** for such an n-party communication protocol (B):

$$IC(B,\mu) := \sum_{i=1}^{n}\sum_{j=1}^{i} I(B_i; X_j | B_{j-1}),$$

where X_i represents the input to party $\mathcal{P}_i$, $X = (X_1, \ldots, X_n)$ is drawn from the distribution μ, and B_i represents the message sent by $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$. B_i depends on B_{i-1}, X_i and the private randomness used by $\mathcal{P}_i$ (we will refer to B_0 as the random variable generating the starting message for party $\mathcal{P}_1$). We define further notions of information cost for n-party protocols as needed subsequently in the paper. For streaming algorithms M that only use private randomness, we show that when $X = (X_1, \ldots, X_n)$ is drawn from the uniform distribution over $\{0,1\}^n$, then

$$\sum_{i=1}^{n}\sum_{j=1}^{i} I(M_i; X_j | M_{j-1}) \leq \sum_{i=1}^{n} |M_i| \leq n \cdot (\text{space used by } M),$$

where $|M_i|$ is the length of the i-th message, which is at most the memory used by M. Thus, we can *lower bound the space required by M to perform a task by lower bounding the information cost of any such n-party protocol that performs the same task.*

III. Link to the Full Version

Full version of the paper with detailed theorems, proofs and algorithms can be found at

https://eccc.weizmann.ac.il/report/2020/139/.

Acknowledgment

We would like to thank Jelani Nelson and Huacheng Yu for their many useful comments and discussions.

References

[1] G. Cormode, "Sketch techniques for massive data," in *Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches*, ser. Foundations and Trends in Databases, G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, Eds. Hanover, MA, USA: Now Publishers Inc., Jan. 2012, vol. 4, pp. 1–294. [Online]. Available: http://dx.doi.org/10.1561/1900000004

[2] S. Muthukrishnan *et al.*, "Data streams: Algorithms and applications," *Foundations and Trends® in Theoretical Computer Science*, vol. 1, no. 2, pp. 117–236, 2005.

[3] J. Nelson, "Sketching and streaming algorithms for processing massive data," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 19, no. 1, pp. 14–19, 2012.

[8]https://www.cs.bgu.ac.il/~asml162/wiki.files/pollard-pinsker.pdf.

[4] R. Morris, "Counting large numbers of events in small registers," *Communications of the ACM*, vol. 21, no. 10, pp. 840–842, 1978.

[5] D. M. Kane, J. Nelson, and D. P. Woodruff, "On the exact space complexity of sketching and streaming small norms," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2010, pp. 1161–1178.

[6] J. Brody and E. Verbin, "The coin problem and pseudo-randomness for branching programs," in *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, 2010, pp. 30–39.

[7] J. P. Steinberger, "The distinguishability of product distributions by read-once branching programs," in *Proceedings of the 28th Conference on Computational Complexity, CCC 2013, K.lo Alto, California, USA, 5-7 June, 2013*, 2013, pp. 248–254.

[8] G. Cohen, A. Ganor, and R. Raz, "Two sides of the coin problem," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[9] C. H. Lee and E. Viola, "The coin problem for product tests," *TOCT*, vol. 10, no. 3, pp. 14:1–14:10, 2018.

[10] R. Jayaram and D. P. Woodruff, "Towards optimal moment estimation in streaming and distributed models," *CoRR*, vol. abs/1907.05816, 2019.

[11] A. Chattopadhyay, J. Radhakrishnan, and A. Rudra, "Topology matters in communication," in *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. IEEE, 2014, pp. 631–640.

[12] A. Chattopadhyay and A. Rudra, "The range of topological effects on communication," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2015, pp. 540–551.

[13] A. Chakrabarti, G. Cormode, and A. McGregor, "Robust lower bounds for communication and stream computation," *Theory Comput.*, vol. 12, no. 1, pp. 1–35, 2016.

[14] J. Misra and D. Gries, "Finding repeated elements," *Science of computer programming*, vol. 2, no. 2, pp. 143–152, 1982.

[15] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 6, pp. 1219–1232, 2005.

[16] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International Conference on Database Theory*. Springer, 2005, pp. 398–412.

[17] R. Jayaram and D. P. Woodruff, "Data streams with bounded deletions," in *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, 2018, pp. 341–354.

[18] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[19] G. Cormode, "Sketch techniques for approximate query processing."

[20] M. Charikar, K. C. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theor. Comput. Sci.*, vol. 312, no. 1, pp. 3–15, 2004. [Online]. Available: https://doi.org/10.1016/S0304-3975(03)00400-6

[21] V. Braverman, S. R. Chestnut, N. Ivkin, and D. P. Woodruff, "Beating countsketch for heavy hitters in insertion streams," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, 2016, pp. 740–753.

[22] V. Braverman, S. R. Chestnut, N. Ivkin, J. Nelson, Z. Wang, and D. P. Woodruff, "BPTree: An L2 heavy hitters algorithm using constant memory," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, 2017, pp. 361–376.

[23] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar, "An information statistics approach to data stream and communication complexity," *Journal of Computer and System Sciences*, vol. 68, no. 4, pp. 702–732, 2004.

[24] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff, "Fast moment estimation in data streams in optimal space," in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011, pp. 745–754.

[25] H. Robbins, "A remark on Stirling's formula," *The American mathematical monthly*, vol. 62, no. 1, pp. 26–29, 1955.

[26] J. A. Adell, A. Lekuona, and Y. Yu, "Sharp bounds on the entropy of the Poisson law and related quantities," *IEEE transactions on information theory*, vol. 56, no. 5, pp. 2299–2306, 2010.