New Verification Schemes for Frequency-Based Functions on Data Streams

Prantar Ghosh
Dartmouth College, Hanover, NH, USA
prantar.ghosh.gr@dartmouth.edu

Abstract

We study the general problem of computing *frequency-based functions*, i.e., the sum of any given function of data stream frequencies. Special cases include fundamental data stream problems such as computing the number of distinct elements (F_0), frequency moments (F_k), and heavy-hitters. It can also be applied to calculate the maximum frequency of an element (F_∞).

Given that exact computation of most of these special cases provably do not admit any sublinear space algorithm, a natural approach is to consider them in an enhanced data streaming model, where we have a computationally unbounded but untrusted *prover* that can send *proofs* or help messages to ease the computation. Think of a memory-restricted client delegating the computation to a powerful cloud service. The client does not blindly trust the cloud, and with its limited memory, it wants to *verify* the proof that the cloud sends. Chakrabarti et al. (ICALP '09) introduced this model as the *annotated data streaming model* and showed that multiple problems including exact computation of frequency-based functions – that have no sublinear algorithms in basic streaming – do have algorithms, also called *schemes*, in the annotated streaming model with both space and proof-length sublinear in the input size.

We give a general scheme for computing any frequency-based function with both space usage and proof-size of $O(n^{2/3} \log n)$ bits, where n is the size of the universe. This improves upon the best known bound of $O(n^{2/3} \log^{4/3} n)$ given by the seminal paper of Chakrabarti et al. and as a result, also improves upon the best known bounds for the important special cases of computing F_0 and F_∞. We emphasize that while being quantitatively better, our scheme is also qualitatively better in the sense that it is simpler than the previously best scheme that uses intricate data structures and elaborate subroutines. Our scheme uses a simple technique tailored for this model: the verifier solves the problem partially by running an algorithm known to be helpful for it in the basic (sans prover) streaming model and then takes the prover's help to solve the remaining part.

1 Introduction

Interactive proof systems have contributed a very important conceptual message to computer science: it is possible for a computationally bounded entity to reduce its computational cost for a problem if it is only required to verify a proof of the solution instead of finding a solution on its own. This concept led to celebrated results such as IP = PSPACE [28] and the PCP Theorems [3, 4]. It is natural to incorporate this idea to deal with challenging problems in

massive data streams so as to reduce the impractical computational costs for such problems. This incorporation led to the following setting: a space-restricted client reading a huge data stream outsources the computation to a more powerful entity, such as a cloud service, with unbounded space. The cloud sends the result of the computation to the client who refuses to blindly trust it since it might be malicious or might have incurred some hardware failure. Therefore, the cloud (henceforth named "Prover") also sends the client (henceforth named "Verifier") a *proof* in support of its results. Verifier needs to use his limited space to collect sufficient information from the stream so as to verify the proof. In the case that Prover is honest, Verifier can use it as a help message to find the solution to the underlying problem. Otherwise, he rejects the proof. This combination of data streaming with prover-verifier systems has been fruitful: multiple works [1, 6, 7, 8, 9, 10, 12, 14, 21, 22, 30] have shown that several problems that are provably intractable in the basic data streaming model turn out to be solvable in prover-enhanced models using verification space and proof-length sublinear in the input size.

Chakrabarti et al. [7] formally defined this enhanced data streaming model as the *annotated data streaming model*. An algorithm in this model is called a *scheme*. In designing a scheme, the two important complexity parameters that we need to focus on are the *space* used by Verifier and the *size of the proof* sent by Prover. A scheme that has a proof-length of $O(h)$ bits and uses $O(v)$ bits of space is called an (h, v)-*scheme*.

Since its inception, data streaming algorithms have been extensively studied for fundamental statistical problems such as counting the number of distinct elements in a stream (F_0) [2, 5, 15, 20], the kth frequency moment for $k > 0$ (F_k) [2, 16, 18, 32], the maximum frequency of an element (F_∞) [2, 19], and the ℓ_p-norm of the frequency vector for some $p \geqslant 0$ [19, 20, 26]. All of these problems are special cases of (or can be solved by easily applying) the general problem of computing *frequency-based functions*: given a function $g : \mathbb{Z} \to \mathbb{Z}^+$, find $\sum_{j=1}^n g(f_j)$, where, for each j in the universe $\{1, \ldots, n\}$, f_j is the frequency of the jth element. This general problem was notably addressed by the celebrated seminal paper by Alon, Matias, and Szegedy [2]: they asked for a characterization of precisely which frequency-based functions can be approximated efficiently in the basic streaming model. The aforementioned paper by Chakrabarti et al. [7] studied such statistical problems in the annotated streaming setting and gave several interesting schemes. In particular, for the general problem of computing frequency-based functions, they gave an $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$-scheme. Their scheme uses an intricate data structure with binary trees and calls upon a subroutine for heavy-hitters that uses an elaborate framework called *hierarchical heavy hitters*.

Given how general the problem is, with several important special cases having numerous applications, it is important and beneficial to have a *simple* scheme for the general problem. In this work, we design such a simple scheme that uses the most basic and classical data structure for frequency estimation: the Misra-Gries summary [25]. Our scheme ends up improving the best known complexity bounds for the problem: we give an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme. No better bounds or simpler algorithms were known even for the special cases of computing F_0 or F_∞. Our result thus simplifies and improves the bounds for these problems as well.

The aforementioned scheme works for streams of length $m = O(n)$, an assumption that was also made by Chakrabarti et al. [7]. However, their scheme can be made to work for longer turnstile streams as long as $\|\mathbf{f}\|_1 = O(n)$. We show how to use the Count-Median Sketch [13], an estimation algorithm with stronger guarantees than Misra-Gries, to get a scheme with similar complexity bounds for these long streams. But since the Count-Median Sketch is randomized (contrary to Misra-Gries), we incur a non-zero completeness error for this scheme. The high-level idea of both our schemes is the following: we use the estimation algorithm as a primitive to "partially" solve the problem. Prover then helps Verifier with the "remaining" unsolved part.

1.1   Setup and Terminology

We formalize the setting described above. A *scheme* for computing a function $g(\sigma)$ of the input stream σ is a triple $(\mathcal{H}, \mathcal{A}, \text{out})$, where $\mathcal{H}$ is a function that Prover uses to generate the help message or proof-stream for σ, given by $\mathcal{H}(\sigma)$, $\mathcal{A}$ is a data streaming algorithm that Verifier runs on the stream σ using a random string R to produce a summary $\mathcal{A}_R(\sigma)$, and out is a streaming algorithm that Verifier runs on the proof-stream $\mathcal{H}(\sigma)$ and also uses $\mathcal{A}_R(\sigma)$ and R to generate an output $\text{out}_R(\mathcal{H}(\sigma), \mathcal{A}_R(\sigma))$ in $\text{range}(g) \cup \perp$, where the symbol $\perp$ denotes rejection of the proof. Note that if the proof-length $|\mathcal{H}(\sigma)|$ is larger than the memory of Verifier, then he needs to process $\mathcal{H}(\sigma)$ as a stream.

A scheme $(\mathcal{H}, \mathcal{A}, \text{out})$ has completeness error ε_c and soundness error ε_s if it satisfies

- (completeness) $\forall \sigma : \Pr_R[\text{out}_R(\mathcal{A}_R(\sigma), \mathcal{H}(\sigma)) = g(\sigma)] \geqslant 1 - \varepsilon_c$;
- (soundness) $\forall \sigma, H : \Pr_R[\text{out}_R(\mathcal{A}_R(\sigma), H) \notin \{g(\sigma), \perp\}] \leqslant \varepsilon_s$.

Informally, this means that an honest Prover can convince Verifier to produce the correct output with high probability. Again, if Prover is dishonest, then, with high probability, Verifier rejects the proof. We usually aim for $\varepsilon_c, \varepsilon_s \leqslant 1/3$ (they can be boosted down using standard techniques incurring a small increase in the space usage). A scheme is said to have *perfect completeness* if $\varepsilon_c = 0$.

The *hcost* (short for "help cost") of a scheme $(\mathcal{H}, \mathcal{A}, \text{out})$ is defined as $\max_\sigma |\mathcal{H}(\sigma)|$, i.e., the maximum number of bits required to express a proof. The *vcost* (short for "verification cost") is the maximum bits of space used by the algorithms $\mathcal{A}_R(\sigma)$ and $\text{out}_R(\sigma)$, where the maximum is taken over all inputs σ and possible random strings R. A scheme with hcost $O(h)$ and vcost $O(v)$ is called an (h, v)-scheme. An (h, v)-scheme is interesting if $h > 0$ and v is asymptotically smaller than the best bound achievable for $h = 0$, i.e., in the basic (sans prover) streaming model.

1.2   Our Results and Techniques

In this section, we state our results and give an overview of our techniques.

Results.   Given a stream with elements in $[n]$, let $\mathbf{f}$ denote its frequency vector $\langle f_1, f_2, \ldots, f_n \rangle$, where f_j is the frequency of the jth element. A *frequency-based function* is a function $G(\mathbf{f})$ of the form $G(\mathbf{f}) := \sum_{j=1}^{n} g(f_j)$ for some function $g : \mathbb{Z} \to \mathbb{Z}^+$.

Our main result is captured in the following theorem which we prove in Section 3.2.1.

▶ **Theorem 1.** *There is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing any frequency-based function in any turnstile stream of length $m = O(n)$. The scheme is perfectly complete and has soundness error at most $1/poly(n)$.*

With some modifications, we obtain a similar scheme for longer streams at the cost of imperfect completeness. This is given by the following theorem which we prove in Section 3.2.2.

▶ **Theorem 2.** *There is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing any frequency-based function in any turnstile stream with $\|\mathbf{f}\|_1 = O(n)$. The scheme has completeness and soundness errors at most $1/3$.*

As a consequence, we get schemes with the same complexity bounds for the problems of computing F_0, F_∞, and checking multiset inclusion (see Corollary 5 for formal definition). Just as for frequency-based functions, our schemes also improve upon the best known bounds for these special cases and applications[1]. We discuss these results in detail in Section 3.3.

---

[1] Computing F_k for constant $k > 0$ is a well-studied special case for which better bounds are known [7].

▶ **Corollary 3.** *For any turnstile stream with $\|\mathbf{f}\|_1 = O(n)$, there is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing F_0, the number of distinct elements with non-zero frequency, with completeness and soundness errors at most $1/3$. The scheme can be made perfectly complete with soundness error $1/poly(n)$ if the stream has length $m = O(n)$.*

▶ **Corollary 4.** *For any turnstile stream with $\|\mathbf{f}\|_1 = O(n)$, there is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing F_∞, the maximum frequency of an element, with completeness and soundness errors at most $1/3$. The scheme can be made perfectly complete with soundness error $1/poly(n)$ if the stream has length $m = O(n)$.*

▶ **Corollary 5.** *Let $X, Y \subseteq [n]$ be multisets of size $O(n)$. Given a stream where elements of X and Y arrive in interleaved manner, there is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for determining whether $X \subseteq Y$.*

Techniques.   Computing frequency-based functions is challenging simply because we don't have enough space to store all the exact frequencies. However, there are efficient small-space algorithms – e.g., Misra-Gries algorithm [25], Count-Median Sketch [13] – that return reasonably good *estimates* of the frequencies. We use such an algorithm as a primitive in our schemes. The estimates returned partially solve the problem by helping us identify the "heavy-hitters" or the most frequent items. There cannot be too many heavy-hitters and hence, the all-powerful Prover can send Verifier the exact frequencies of these elements (which of course need to be verified) without too much communication. On the other hand, the rest of the elements, though large in number, have relatively small frequency. We show a way to encode the answer in terms of a low-degree polynomial when the frequencies are small. Prover can then send us this polynomial using few bits, enabling us to solve the problem with small communication overall.

We remark that the high-level technique used in our first scheme – using Misra-Gries as a subroutine – might be more widely applicable than that used in the second one, i.e., using Count-Median Sketch. This is because Misra-Gries is deterministic while Count-Median is randomized. In general, both Prover and Verifier can locally run a *deterministic* algorithm on the input, and then, Prover can send messages based on the final state of that algorithm. Note that it isn't clear if a *randomized* algorithm can always help in this regard since we assume that Prover and Verifer do not have access to shared randomness[2]. Hence, the final states of the algorithm might vary drastically for Prover and Verifier if they run it locally with their own private randomness. For our problem, we don't run into this issue since we don't require Prover to know the exact output of the Verifier's local estimation algorithm.

Other techniques used are pretty standard in this area. We use techniques based on the famous *sum-check protocol* of Lund et al. [24] that encodes answers as sum of low-degree polynomials. In our case, where Prover sends only a single message to Verifier, a quantity of interest is expressed as the sum of evaluations of a low-degree univariate polynomial. Since the polynomial has low-degree, it can be expressed with a small number of monomials. Thus, Prover needs only a few bits to express the set of coefficients that describe the polynomial, leading to short proof-length. Moreover, to verify the authenticity of the polynomial, Verifier needs to evaluate it at just a single random point, the space for which he can afford. The main challenge in this technique is to find the proper low-degree polynomials to encode the answer,

---

[2] This assumption is made so that it corresponds to the MA communication model. Access to shared randomness corresponds to the AMA communication model where better bounds are known [17].

and in this work, we give such new polynomial encodings for the underlying sub-problems. Another standard technique we use is the *shaping technique* that transforms a one-dimensional vector into a two-dimensional array. On a high level, this helps in "distributing" the work between Prover and Verifier as they each "take care of" a single dimension. Pertaining to the streaming model, we exploit the popular technique of *linear sketching* where we express a quantity of interest as a linear combination of the stream updates, which helps us to maintain the quantity dynamically as the stream arrives.

1.3 Related Work

Early works on the concept of stream outsourcing and verification were done by the database community [23, 27, 31, 33]. Motivated by these works, Chakrabarti et al. [7] abstracted out and formalised the theoretical aspects of the settings. They defined two types of stream verification settings: (i) the *annotated data streaming* setting – calling the schemes as *online schemes* – where Prover and Verfier read the input stream together and Prover sends help messages during and/or after the stream arrival based on the part of the stream she has seen so far, and (ii) the *prescient* setting where Prover knows the entire stream upfront, i.e., before Verifies sees it, and can send help messages accordingly. Several subsequent works [6, 9, 10, 12, 21, 30] studied these non-interactive models. Natural generalizations of the model, where we allow multiple rounds of interaction between Prover and Verifier, have also been explored. These include *Arthur-Merlin streaming protocols* (Prover is named "Merlin" and Verifier is named "Arthur" following a long-standing tradition in complexity theory) of Gur and Raz [17] and the *streaming interactive proofs* (SIP) of Cormode et al. [14]. The latter setting was further studied by multiple works [1, 8, 22]. We refer the reader to the expository article by Thaler [29] for a detailed survey of this area.

We state the results with the standard assumption [7, 14] that $m = O(n)$. Chakrabarti et al. [7] gave two schemes for computing any general frequency-based function: an online $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$-scheme and a prescient $(n^{2/3} \log n, n^{2/3} \log n)$-scheme. They noted that the schemes apply to get best known schemes for the special cases of computing the number of distinct elements (F_0), the maximum frequency (F_∞), and for checking multiset inclusion. They also showed a lower bound that any online or prescient (h, v)-scheme for the problem (even for the aforementioned special cases) requires $hv \geqslant n$. They designed schemes with better bounds for certain other frequency-based functions, often matching this lower bound up to polylogarithmic factors. For instance, for any $hv = n$, they gave an online $(k^2 h \log n, kv \log n)$-scheme for calculating the kth frequency moment F_k for any positive integer k, and a $(\phi^{-1} \log^2 n + h \log n, v \log n)$-scheme for computing the ϕ-heavy hitters (elements with frequency of at least a ϕ-fraction of the stream length).

The specific problem of computing F_0 has been studied by multiple works in various stream verification models. Cormode, Mitzenmacher, and Thaler [11] studied the problem in the stronger SIP-model and gave a $(\log^3 n, \log^2 n)$-SIP with $O(\log^2 n)$ rounds of communication. For the case where we restrict the number of rounds to $O(\log n)$, Cormode, Thaler, and Yi [14] gave a $(\sqrt{n} \log^2 n, \log^2 n)$-SIP. Klauck and Prakash [22] improved this to a $(\log^4 n \log \log n, \log^2 n \log \log n)$-SIP. Gur and Raz [17] designed an $(\tilde{O}(\sqrt{n}), \tilde{O}(\sqrt{n}))$-AMA-streaming protocol[3] (the $\tilde{O}(\cdot)$ notation hides polylog(n) factors) for F_0.

---

[3] AMA stands for the communication pattern Arthur-Merlin-Arthur

2   Preliminaries

Here, we discuss the streaming models we study and some standard results that we use in our schemes.

Throughout this paper, the stream elements come from the universe $[n] := \{1, \ldots, n\}$ and the stream length is m. In the *turnstile* streaming model, tokens are of the form $(j, \Delta) \in [n] \times \mathbb{Z}$, which means Δ copies of the element j are inserted (resp. deleted) if $\Delta > 0$ (resp. $\Delta < 0$). The *cash register* or *insert-only* streaming model is the special case when Δ is always positive. In this paper, for simplicity, we assume unit updates, i.e., $\Delta \in \{-1, 1\}$ always. The assumption can be easily removed by looking at an update as a collection of multiple unit updates.

For a stream $\sigma = \langle (a_1, \Delta_1), \ldots, (a_m, \Delta_m) \rangle$, the *frequency vector* $\mathbf{f}(\sigma)$ is defined as $\langle f_1, \ldots, f_n \rangle$ where f_j is the *frequency* of element j, given by $f_j := \sum_{\substack{i \in [m]: \\ a_i = j}} \Delta_i$. We denote *estimates* of f_j by $\hat{f}_j$. We drop the argument σ from $\mathbf{f}(\sigma)$ when the stream is clear from the context.

In our schemes, we use the standard technique of *sketching* a frequency vector by evaluating its *low-degree extension* at a random point. We explain what this means. We transform (or *shape*) our frequency vector of length n into a 2-dimensional $d_1 \times d_2$ array f, where $d_1 d_2 = n$, using some canonical bijection from $[n]$ to $[d_1] \times [d_2]$. This means that the domain of the function f can now be seen as $[d_1] \times [d_2]$. We work on a finite field $\mathbb{F}$ with large enough characteristic such that the values don't "wrap around" under operations in $\mathbb{F}$. By Lagrange's interpolation, there is a unique polynomial $\tilde{f}(X, Y) \in \mathbb{F}[X, Y]$ with $\deg_X(\tilde{f}) = d_1 - 1$ and $\deg_Y(\tilde{f}) = d_2 - 1$ such that $\tilde{f}(x, y) = f(x, y)$ for all $(x, y) \in [d_1] \times [d_2]$. We call $\tilde{f}$ the *low-degree $\mathbb{F}$-extension* of f. For each $(x, y) \in [d_1] \times [d_2]$, we have "Lagrange basis polynomials" defined as

$$\delta_{x,y}(X, Y) := \left(\prod_{x_i \in [d_1] \setminus \{x\}} \frac{X - x_i}{x - x_i} \right) \cdot \left(\prod_{y_i \in [d_2] \setminus \{y\}} \frac{Y - y_i}{y - x_i} \right) \tag{1}$$

We can write $\tilde{f}$ as a linear combination of these polynomials as follows:

$$\tilde{f}(X, Y) = \sum_{(x,y) \in [d_1] \times [d_2]} f(x, y) \, \delta_{x,y}(X, Y)$$

In particular, if f is built up from a stream of turnstile updates $\langle ((x, y)_j, \Delta_j) \rangle$, then

$$\tilde{f}(X, Y) = \sum_j \Delta_j \, \delta_{(x,y)_j}(X, Y) \,. \tag{2}$$

Thus, we can use Equation (2) to *maintain* $\tilde{f}(x^*, y^*)$ at some fixed point (x^*, y^*) dynamically with stream updates. We formalize this in the following fact.

▶ **Fact 6.** *Given a point $(x^*, y^*) \in \mathbb{F}^2$ and a stream of updates to an initially-zero $d_1 \times d_2$-dimensional array f, we can maintain $\tilde{f}(x^*, y^*)$ using $O(\log |\mathbb{F}|)$ space. For implementation details and generalizations, see Cormode et al. [14].*

An important subroutine in one of our schemes is the classic Misra-Gries algorithm for frequency estimation [25] which, given an input stream of m elements and a fraction ϕ, estimates the frequency of the stream elements within an additive factor of ϕm. We recall this algorithm in Algorithm 1.

Informally, the algorithm does the following: it keeps an array or "dictionary" K indexed by "keys" that are elements of the stream and each of them has an associated counter $K[i]$. At any point of time, the array has at most $\lceil \phi^{-1} \rceil$ keys. When a stream element arrives, it increments the counter for the element if it's present in the keys (it includes it in the keys if there are less than $\lceil \phi^{-1} \rceil$ keys), and otherwise decrements the counter of every key. If a counter for a key becomes 0, it is removed from K. Finally, the estimate $\hat{f}_j$ is given by $K[j]$ (which is 0 if j is not in the keys). The guarantees of the algorithm is given in Fact 7.

▌ **Algorithm 1** [25] Misra-Gries algorithm for frequency estimates in insert-only streams.

Input: Stream σ; $\phi \leqslant 1$
 1: Initialize $K \leftarrow$ empty array
Process(token $j \in \sigma$):
 2: **if** $j \in \text{keys}(K)$ **then**
 3:      $K[j] \leftarrow K[j] + 1$
 4: **else**
 5:      **if** $|\text{keys}(K)| < \lceil \phi^{-1} \rceil$ **then**
 6:          $K[j] \leftarrow 1$
 7:      **else**
 8:          **for** $i \in \text{keys}(K)$ **do**:
 9:              $K[i] \leftarrow K[i] - 1$
10:              if $K[i] = 0$ then remove i from $\text{keys}(K)$
Output:
11: **for** $j \in [n]$ **do**:
12:      if $j \in \text{keys}(K)$ then **return** $\hat{f}_j = K[j]$; else **return** $\hat{f}_j = 0$

▶ **Fact 7** ([25]). *For an insert-only stream of m elements in $[n]$, given any $\phi \leqslant 1$, Algorithm 1 uses $O(\phi^{-1}(\log n + \log m))$ space and returns frequency estimates $\langle \hat{f}_j : j \in [n] \rangle$ such that, for all tokens $j \in [n]$, we have $f_j - \phi m \leqslant \hat{f}_j \leqslant f_j$.*

Note that this algorithm was designed for insert-only streams and doesn't work for turnstile streams. To use it for turnstile streams, we need to make appropriate modifications (which we do in Section 3.1).

3 Computing Frequency-based Functions in Turnstile Streams

Let $\mathbf{f}$ be the frequency vector of a stream as defined in Section 2. Recall that a *frequency-based function* is a function $G(\mathbf{f})$ of the form $G(\mathbf{f}) := \sum_{j \in [n]} g(f_j)$ for some function $g : \mathbb{Z} \to \mathbb{Z}^+$. In this section, we obtain an improved $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing any frequency-based function for some predetermined function g. As stated earlier, we design a scheme exploiting the Misra-Gries algorithm (Algorithm 1). We want to use it as a subroutine in our problem for turnstile streams, but it works only in the insert-only model. Therefore, in Section 3.1, we provide a simple extension of the algorithm that attains a similar guarantee for turnstile streams. In Section 3.2, first, we use this extended Misra-Gries (EMG) algorithm as a subroutine for our scheme for computing frequency-based functions. Next, we show that we can instead use the Count-Median Sketch [13] to make our scheme work for longer streams. In Section 3.3, we discuss some important applications of our schemes.

3.1 Extension of Misra-Gries Algorithm for Turnstile Streams

The extended Misra-Gries algorithm (henceforth called "EMG algorithm") works as follows: we process the positive and negative updates separately in two parallel copies of Algorithm 1 to estimate the total positive update and (absolute value of) the total negative update. In the second copy, we can actually think of the updates as "increments" since only negative updates are processed there. Thus, what we are actually estimating is the absolute value of the total negative update.

For each j, let the total positive update be f_j^+ and (absolute value of) the total negative update f_j^-. Then, the actual frequency is $f_j = f_j^+ - f_j^-$. Denote the corresponding estimates given by the copies of Algorithm 1 by $\hat{f}_j^+$ and $\hat{f}_j^-$. Then $\hat{f}_j := \hat{f}_j^+ - \hat{f}_j^-$ gives a similar guarantee as Fact 7 for turnstile streams; this time, we also incur an additive error of ϕm on the upper bound.

To see this, note that by Fact 7, we have, $\forall j \in [n]$,

$$f_j^+ - \phi m \leqslant \hat{f}_j^+ \leqslant f_j^+ \tag{3}$$

$$f_j^- - \phi m \leqslant \hat{f}_j^- \leqslant f_j^- \tag{4}$$

Thus, Equations (3) and (4) give $f_j^+ - f_j^- - \phi m \leqslant \hat{f}_j^+ - \hat{f}_j^- \leqslant f_j^+ - f_j^- + \phi m$, i.e.,

$$f_j - \phi m \leqslant \hat{f}_j \leqslant f_j + \phi m \tag{5}$$

Hence, this time we get double sided error. This estimate would suffice for getting our desired scheme. Therefore, we get the following lemma.

▶ **Lemma 8.** *Given a turnstile stream of m elements in $[n]$, the EMG algorithm uses $O(\phi^{-1}(\log n + \log m))$ space and returns a summary $\langle \hat{f}_j : j \in [n] \rangle$ such that, for all $j \in [n]$, we have $f_j - \phi m \leqslant \hat{f}_j \leqslant f_j + \phi m$.*

▶ Remark 9. The guarantee given by the EMG algorithm may not be very useful in general for turnstile streams. This is because the total number of stream updates m can be huge, whereas the frequency of each token can be small since we allow both increments and decrements in the turnstile model. The classic Misra-Gries algorithm for insert-only model, on the other hand, has a good guarantee (Fact 7) since $m = \|\mathbf{f}\|_1$ in this model. However, for our purpose, the guarantee in Lemma 8 is good enough since we assume that $m = O(n)$.

3.2 Schemes for Frequency-based Functions

First, in Section 3.2.1, we describe a protocol for computing frequency-based functions in turnstile streams of length $O(n)$ and prove Theorem 1. Next, in Section 3.2.2, we show that the scheme can be modified to work for any turnstile stream with $\|\mathbf{f}\|_1 = O(n)$, proving Theorem 2. The completeness error in the latter scheme is, however, non-zero.

3.2.1 Perfectly Complete Scheme for $O(n)$-Length Streams

As in prior works [7, 14], we solve the problem for stream length $m = O(n)$. Hence, by Lemma 8, the EMG algorithm takes $O(\phi^{-1} \log n)$ space and gives, for some constant c,

$$\forall j \in [n] : \ f_j - \phi cn \leqslant \hat{f}_j \leqslant f_j + \phi cn \,. \tag{6}$$

Set $\phi = (cn^{2/3})^{-1}$. Therefore, we have an $O(n^{2/3} \log n)$ space algorithm that guarantees

$$\forall j \in [n] : \ f_j - n^{1/3} \leqslant \hat{f}_j \leqslant f_j + n^{1/3} \,.$$

Let K denote the set of keys in the final state of the EMG algorithm for the setting of $\phi = 1/(cn^{2/3})$. Observe that if $\hat{f}_j = 0$ for some j (i.e., $j \notin K$), we know that $f_j \in [-n^{1/3}, n^{1/3}]$.

Define $h(j) = \mathbb{I}\{j \notin K\}$ where $\mathbb{I}$ is the 0-1 indicator function. We have

$$\sum_{j \in [n]} g(f(j)) = \sum_{j \in K} g(f(j)) + \sum_{j \notin K} g(f(j)) = \sum_{j \in K} g(f(j)) + \sum_{j \in [n]} g(f(j))h(j)$$

Let $L := \sum_{j \in K} g(f(j))$ and $R := \sum_{j \in [n]} g(f(j))h(j)$. We shall compute L and R separately and add them to get the desired answer.

We *shape* (see Section 2) the 1D array $[n]$ into a 2D $n^{1/3} \times n^{2/3}$ array. Thus, we get

$$R = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} g(f(x,y))h(x,y)$$

As is standard [7], we assume that the range of the function g is upper bounded by some polynomial in n, say n^p. Pick a prime q such that $n^{p+1} < q < 2n^{p+1}$. We will work in the finite field $\mathbb{F}_q$ and the upper bound on the range of g ensures that $G(\mathbf{f})$ will not "wrap around" under arithmetic in $\mathbb{F}_q$.

Let $\tilde{f}, \tilde{h}$ be polynomials of lowest degree over the finite field $\mathbb{F}_q$ that agree with f, h respectively at all values in $[n^{1/3}] \times [n^{2/3}]$. Note that, by Lagrange's interpolation, both $\tilde{f}$ and $\tilde{h}$ have degrees $n^{1/3} - 1$ and $n^{2/3} - 1$ in the two variables (see Section 2). Again, let $\tilde{g}$ denote the polynomial of lowest degree that agrees with g at all values in $[-n^{1/3}, n^{1/3}]$. Thus, $\tilde{g}$ has degree $2n^{1/3}$.

Therefore, we have

$$R = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} \tilde{g}(\tilde{f}(x,y))\tilde{h}(x,y)$$

i.e., we can write

$$R = \sum_{x \in [n^{1/3}]} P(x), \tag{7}$$

where the polynomial P is given by

$$P(X) = \sum_{y \in [n^{2/3}]} \tilde{g}(\tilde{f}(X,y))\tilde{h}(X,y) \tag{8}$$

To compute L, it suffices to obtain the values f_j for all $j \in K$ since g is predetermined. In our protocol, Prover would send values f_j' that she claims to be f_j for all $j \in K$. Define

$$T := \sum_{j \in K} (f_j - f_j')^2$$

Note that we have $f_j = f_j'$ for each j if and only if $T = 0$. Set $f_j' := 0$ for all $j \notin K$. Thus, we can rewrite T as

$$T = \sum_{j \in [n]} (f_j - f_j')^2 (1 - h(j))$$

Using shaping as before, we get

$$T = \sum_{x \in [n^{1/3}]} \sum_{y \in [n^{2/3}]} (f(x,y) - f'(x,y))^2 (1 - h(x,y)) \tag{9}$$

Let $\tilde{f}'$ denote the polynomial of lowest degree over $\mathbb{F}_q$ that agrees with f' at all values in $[n^{1/3}] \times [n^{2/3}]$. Therefore, we have

$$T = \sum_{x \in [n^{1/3}]} Q(x) \tag{10}$$

where the polynomial Q is given by

$$Q(X) = \sum_{y \in [n^{2/3}]} (\tilde{f}(X, y) - \tilde{f}'(X, y))^2 (1 - \tilde{h}(X, y)). \tag{11}$$

We are now ready to describe the protocol.

Stream processing. Verifier picks $r \in \mathbb{F}_q$ uniformly at random. As the stream arrives, he maintains $\tilde{f}(r, y)$ for all $y \in [n^{2/3}]$ (Fact 6). In parallel, he runs the EMG algorithm setting $\phi = (cn^{2/3})^{-1}$.

Help message. Prover sends polynomials P' and Q', and values f'_j for all $j \in K$. She claims that P', Q', f' are identical to P, Q, f respectively. The polynomials are sent as streams of their coefficients following some canonical order of their monomials. Verifier evaluates $P'(r)$ and $Q'(r)$ as the polynomials are streamed.

Verification and output. Looking at the final state of the EMG subroutine, Verifier constructs $\tilde{h}(r, y)$ for all $y \in [n^{2/3}]$ (he can treat the keys as a stream and use Fact 6). Also, from the values f'_j, he constructs $\tilde{f}'(r, y)$ for all $y \in [n^{2/3}]$. The $O(n^{1/3})$-degree polynomial $\tilde{g}$ is computed and stored in advance (we need to evaluate g at all points in $[-n^{1/3}, n^{1/3}]$ and then use Lagrange interpolation to get $\tilde{g}$).

Thus, Verifier can now use Equation (8) to compute $P(r)$ and Equation (11) to compute $Q(r)$. He checks whether $P(r) = P'(r)$ and $Q(r) = Q'(r)$. If the checks pass, he believes P', Q' are correct. He further checks whether $\sum_{x \in [n^{1/3}]} Q'(x) = 0$, i.e., by Equation (10), whether $T = 0$. If so, he believes that $f'_j = f_j$ for all $j \in K$. Next, he computes $L = \sum_{j \in K} g(f'(j))$, and using Equation (7), he computes $R = \sum_{x \in [n^{1/3}]} P'(x)$. Finally, $L + R$ gives the answer.

Error probability. The correctness analysis follows along standard lines of sum-check protocols. The scheme is perfectly complete since it follows from above that we always output correctly if Prover is honest. For soundness, note that the protocol fails if either $P \neq P'$ or $Q \neq Q'$, but $P(r) = P'(r)$ and $Q(r) = Q'(r)$. Then, r is a root of the non-zero polynomial $P - P'$ or $Q - Q'$. Since degree of $P - P'$ is $O(n^{2/3})$ and that of $Q - Q'$ is $O(n^{1/3})$, they have at most $O(n^{2/3})$ roots in total. Since r is drawn uniformly at random from $\mathbb{F}_q$, where $q > n^{p+1}$, the probability that r is such a root is at most $O(n^{2/3})/n^{p+1} \leqslant 1/\text{poly}(n)$ for sufficiently large n. Thus, the soundness error is at most $1/\text{poly}(n)$.

Help and Verification costs. The polynomials P and Q have degree $O(n^{2/3})$ and $O(n^{1/3})$ respectively. Thus, it requires $O(n^{2/3} \log n)$ bits in total to express their coefficients since each coefficient comes from $\mathbb{F}_q$ that has size $\text{poly}(n)$. Recall that for the setting of $\phi = (cn^{2/3})^{-1}$, there are $O(\phi^{-1}) = O(n^{2/3})$ keys in the EMG algorithm. Prover sends f'_j for each $j \in K$, and since each frequency is at most $m = O(n)$, this requires $O(n^{2/3} \log n)$ bits to communicate. Therefore, the total hcost is $O(n^{2/3} \log n)$.

As noted above, the invocation of EMG algorithm takes $O(n^{2/3} \log n)$ space. Verifier maintains $\tilde{f}(r, y)$ and stores the values $\tilde{h}(r, y)$ and $\tilde{f}'(r, y)$ for all $y \in [n^{2/3}]$. Each value is an element in $\mathbb{F}_q$, and hence they take up $O(n^{2/3} \log n)$ space in total. The $O(n^{1/3})$-degree polynomial $\tilde{g}$ takes $O(n^{1/3} \log n)$ space to store. Hence, the total vcost is $O(n^{2/3} \log n)$.

Thus, we have proved the following theorem.

▶ **Theorem 1.** *There is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing any frequency-based function in any turnstile stream of length $m = O(n)$. The scheme is perfectly complete and has soundness error at most $1/poly(n)$.*

3.2.2 Handling longer streams at the cost of imperfect completeness

The scheme in Section 3.2.1 requires stream length $m = O(n)$. Note that a turnstile stream with massive cancellations can have length $m \gg n$, but $\|\mathbf{f}\|_1$ can still be $O(n)$. Chakrabarti et al. [7] presented their scheme under the assumption of $m = O(n)$, but their scheme can be made to work for longer streams as long as $\|\mathbf{f}\|_1 = O(n)$. We can modify our scheme to handle such streams as well without increasing the costs, but we no longer have perfect completeness. We give a sketch of this scheme below highlighting the modifications.

We cannot use the EMG algorithm anymore because it doesn't give a strong guarantee with respect to $\|\mathbf{f}\|_1$ for turnstile streams. We use the Count-Median Sketch instead which gives the following guarantee.

▶ **Fact 10** (Count-Median Sketch [13]). *For all $\phi, \varepsilon > 0$, there exists an algorithm that, given a turnstile stream of elements in $[n]$ with $\|\mathbf{f}\|_1 = O(n)$, uses $O(\phi^{-1} \log(\varepsilon^{-1}) \log n)$ space and returns frequency estimates $\langle \hat{f}_j : j \in [n] \rangle$ such that, with probability at least $1 - \varepsilon$, for all tokens $j \in [n]$, we have $f_j - \phi\|\mathbf{f}\|_1 \leqslant \hat{f}_j \leqslant f_j + \phi\|\mathbf{f}\|_1$.*

If $\|\mathbf{f}\|_1 \leqslant cn$ for some constant c, then setting $\phi = (4cn^{2/3})^{-1}$ and $\varepsilon = 1/4$, we get that there is an $O(n^{2/3} \log n)$ space algorithm that, with probability at least $3/4$, gives

$$\forall j \in [n] : f_j - n^{1/3}/4 \leqslant \hat{f}_j \leqslant f_j + n^{1/3}/4 \tag{12}$$

For this protocol, redefine the set K as $K := \{j : |f_j| \geqslant n^{1/3}/2\}$. Prover sends a set K' that she claims is identical to K. Let M denote the set $\{j : |\hat{f}_j| \geqslant 3n^{1/3}/4\}$. Verifier checks whether $M \subseteq K'$, and if the check passes, he computes $\sum_{j \in K'} g(f_j)$ and $\sum_{j \notin K'} g(f_j)$ separately, similar to the earlier protocol, and adds them to obtain the answer.

Error probability. For completeness, note that if Prover is honest and $K' = K$, then with probability at least $3/4$, we have $M \subseteq K'$. To see this, observe that, by the guarantees of the Count-Median Sketch (Equation (12)), for all $j \in [n]$ with $|\hat{f}_j| \geqslant 3n^{1/3}/4$, we have $|f_j| \geqslant n^{1/3}/2$ with probability at least $3/4$. The rest of the completeness analysis is as before, and hence, there is no additional completeness error. Thus, the total completeness error of the scheme is at most $1/4$.

For soundness, suppose that $K' \neq K$. By the guarantees of the Count-Median Sketch, for all $j \in [n]$ with $|f_j| \geqslant n^{1/3}$, we have $|\hat{f}_j| \geqslant 3n^{1/3}/4$ with probability at least $3/4$. Thus, $\{j : |f_j| \geqslant n^{1/3}\} \subseteq M$. Hence, if the check $M \subseteq K'$ passes, then with probability at least $3/4$, we have $\{j : |f_j| \geqslant n^{1/3}\} \subseteq K'$. Thus, if $j \notin K'$, we have $|f_j| < n^{1/3}$. Therefore, the computation of $\sum_{j \notin K'} g(f_j)$ goes through as before. The additional soundness error is at most $1/poly(n)$ as analyzed earlier. Thus, the total soundness error of the protocol is at most $1/4 + 1/poly(n) < 1/3$.

Help and Verification costs. Clearly, since $\|\mathbf{f}\|_1 \leqslant cn$, we have $|K| = O(n^{2/3})$ which adds $O(n^{2/3} \log n)$ bits to the hcost. The Count-Median Sketch takes space $O(n^{2/3} \log n)$, similar to the EMG algorithm. The rest of the cost analysis is as before, and hence we have an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme.

Thus, we have the following theorem.

▶ **Theorem 2.** *There is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing any frequency-based function in any turnstile stream with $\|\mathbf{f}\|_1 = O(n)$. The scheme has completeness and soundness errors at most $1/3$.*

▶ Remark 11. We compare the schemes for Theorem 1 and Theorem 2 (call them Scheme 1 and Scheme 2 respectively). Scheme 2 works for streams of length $m \gg n$ as long as $\|\mathbf{f}\|_1 = O(n)$, while Scheme 1 requires $m = O(n)$. On the negative side, Scheme 2 has imperfect completeness contrary to Scheme 1. Furthermore, the space dependence on the error ε for Scheme 2 is worse than Scheme 1: given any ε, Scheme 2 uses $O(n^{2/3} \log n \log(\varepsilon^{-1}))$ space to bound the completeness and soundness errors by at most ε, while Scheme 1 takes $O\left(n^{2/3}(\log n + \log(\varepsilon^{-1}))\right)$ space to bound the soundness error by ε. This means that to bound the error by $1/\mathrm{poly}(n)$, Scheme 2 takes $O(n^{2/3} \log^2 n)$ space, making it weaker (though simpler) than the scheme of Chakrabarti et al. [7], which takes $O(n^{2/3} \log^{4/3} n)$ space for the same and is also perfectly complete. For this, Scheme 1 takes only $O(n^{2/3} \log n)$ space.

3.3 Special Instances and Applications

Here, we note important implications of Theorems 1 and 2. They can be applied to get similar results for multiple well-studied problems such as computing the number of distinct elements in the stream (F_0), the highest frequency of an element in the stream (F_∞), and checking multiset inclusions. Note that for these problems, to the best of our knowledge, the best-known schemes were $(n^{2/3} \log^{4/3} n, n^{2/3} \log^{4/3} n)$-schemes obtained by direct application of the general scheme. Hence, we improve the bounds and simplify the schemes for these problems as well.

As a direct corollary of Theorems 1 and 2, we get the same bounds for F_0. It is an extensively studied problem in both basic streaming and stream verification. It is the special case of frequency-based functions where the function g is defined as $g(x) = 0$ if $x = 0$, and $g(x) = 1$ otherwise. Therefore, we obtain the following result.

▶ **Corollary 3.** *For any turnstile stream with $\|\mathbf{f}\|_1 = O(n)$, there is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing F_0, the number of distinct elements with non-zero frequency, with completeness and soundness errors at most $1/3$. The scheme can be made perfectly complete with soundness error $1/poly(n)$ if the stream has length $m = O(n)$.*

Another well-studied problem related to frequency-based functions is computing F_∞. Unlike F_0, it is not a direct special case, but a protocol for it follows by easily applying a scheme for frequency-based functions. Chakrabarti et al. [7] noted one way in which it can be applied to solve F_∞. Here, we note a slightly alternate way which doesn't use a subroutine that their scheme uses and is tailored to our protocols: Prover sends the element $j^* \in [n]$ that she claims has the highest frequency and a value f'_{j^*} that she claims to be equal to f_{j^*}. By the above protocols, Verifier can check whether $f'_{j^*} = f_{j^*}$. If the check passes, he computes $G(\mathbf{f}) := \sum_{j=1}^{n} g(f_j)$ using the scheme above, where g is defined as $g(x) = 0$ if $x \leqslant f'_{j^*}$ and $g(x) = 1$ otherwise. He accepts Prover's claim if $G(\mathbf{f}) = 0$. Thus, we get the following result.

▶ **Corollary 4.** *For any turnstile stream with $\|\mathbf{f}\|_1 = O(n)$, there is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for computing F_∞, the maximum frequency of an element, with completeness and soundness errors at most $1/3$. The scheme can be made perfectly complete with soundness error $1/poly(n)$ if the stream has length $m = O(n)$.*

The problem of checking multiset inclusion has two multisets arriving in a stream arbitrarily interleaved between each other, and we need to check if one of them is contained in the other. This abstract problem is used as a subroutine in several other problems, e.g.,

some graph problems considered in the annotated settings [7, 9, 10]. Thus, an improved scheme for multiset inclusion implies improved subroutines for the corresponding problems. It can be solved by easy application of frequency-based functions. The reduction is already noted in Chakrabarti et al. [7], but we repeat it here for the sake of completeness.

▶ **Corollary 5.** *Let $X, Y \subseteq [n]$ be multisets of size $O(n)$. Given a stream where elements of X and Y arrive in interleaved manner, there is an $(n^{2/3} \log n, n^{2/3} \log n)$-scheme for determining whether $X \subseteq Y$.*

Proof. Think of X and Y as n-length characteristic vector representations of the multisets (with an entry denoting the multiplicity of the corresponding element). Then, $X \subseteq Y$ if and only if $X_j \leqslant Y_j$ for each $j \in [n]$. As the elements arrive, we increment an entry if belongs to Y and decrement it if it belongs to X. Thus, the vector $\mathbf{f}$ is given by $f_j = Y_j - X_j$. Define g as $g(x) = 0$ if $x \geqslant 0$ and $g(x) = 1$ otherwise. Therefore, computing $G(\mathbf{f}) := \sum_{j=1}^{n} g(f_j)$ and checking if it equals 0 solves the problem. The multisets having size $O(n)$ ensures that the length of the stream is $O(n)$, and so we can safely apply our scheme. ◀

4 Conclusions and Open Problems

In this work, we designed two new schemes for the broad class of frequency-based functions. These schemes are much simpler than the previously best scheme known for the problem, and furthermore, they even improve upon the complexity bound for space usage and proof-size from $O(n^{2/3} \log^{4/3} n)$ to $O(n^{2/3} \log n)$. The best known upper bound for prescient schemes, given by Chakrabarti et al. [7], is $O(n^{2/3} \log n)$ for both of these complexity parameters. Hence, we also close this small gap between the prescient and online scheme complexities, and show that prescience is not necessary here to bring the polylogarithmic factor down to $\log n$. Additionally, the high-level framework used in our schemes is generic enough to be applicable for multiple other problems.

An important open problem in this area is to determine the asymptotic complexity of the general problem of computing frequency-based functions in the stream verification settings. Chakrabarti et al. [7] showed that any online or prescient (h, v)-scheme for the problem requires $hv \geqslant n$. Note that this lower bound leaves open the possibility of a $(\sqrt{n}, \sqrt{n})$-scheme, while the best known scheme achieves $(\tilde{O}(n^{2/3}), \tilde{O}(n^{2/3}))$ for both online and prescient settings. Can we match the lower bound (up to polylogarithmic factors) and get an $(\tilde{O}(\sqrt{n}), \tilde{O}(\sqrt{n}))$-scheme for the problem, even if prescient? What about for even special cases like F_0 or F_∞? Recall that there exist such online schemes for the kth frequency moment for some constant $k \in \mathbb{Z}^+$ [7]. Also, it is possible to get such a scheme for F_0 if we allow multiple rounds of interaction [17]. Any strict improvement on the lower bound would be extremely interesting and a breakthrough. Currently, we don't know of a function in the turnstile streaming model for which any online (h, v)-scheme must have total cost $h + v \geqslant \omega(\sqrt{N})$ where N is the lower bound on its basic streaming complexity. This is related to the major open question of breaking the "$\sqrt{N}$ barrier" for the Merlin-Arthur (MA) communication model.

References

1 Amirali Abdullah, Samira Daruki, Chitradeep Dutta Roy, and Suresh Venkatasubramanian. Streaming verification of graph properties. In *Proc. 27th International Symposium on Algorithms and Computation*, pages 3:1–3:14, 2016.

2   Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. Preliminary version in *Proc. 28th Annual ACM Symposium on the Theory of Computing*, pages 20–29, 1996.

3   Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. Preliminary version in *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.

4   Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998. Preliminary version in *Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 2–13, 1992.

5   Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proc. 6th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 128–137, 2002.

6   Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 687–706, 2014.

7   Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. Annotations in data streams. *ACM Trans. Alg.*, 11(1):Article 7, 2014.

8   Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkata-subramanian. Verifiable stream computation and Arthur-Merlin communication. In *Proc. 30th Annual IEEE Conference on Computational Complexity*, pages 217–243, 2015.

9   Amit Chakrabarti and Prantar Ghosh. Streaming verification of graph computations via graph structure. In *Proc. 33rd International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 70:1–70:20, 2019.

10 Amit Chakrabarti, Prantar Ghosh, and Justin Thaler. Streaming verification for graph problems: Optimal tradeoffs and nonlinear sketches. *To appear in RANDOM*, 2020. `arXiv: 2007.03039`.

11 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. In *Proc. 18th Annual European Symposium on Algorithms*, pages 231–242, 2010.

12 Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013.

13 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Alg.*, 55(1):58–75, 2005. Preliminary version in *Proc. 6th Latin American Theoretical Informatics Symposium*, pages 29–38, 2004.

14 Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *Proc. VLDB Endowment*, 5(1):25–36, 2011.

15 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

16 Sumit Ganguly and Graham Cormode. On estimating frequency moments of data streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th International Workshop, APPROX 2007, and 11th International Workshop, RANDOM 2007*, volume 4627 of *Lecture Notes in Computer Science*, pages 479–493, 2007.

17 Tom Gur and Ran Raz. Arthur–Merlin streaming complexity. In *Proc. 40th International Colloquium on Automata, Languages and Programming*, pages 528–539, 2013.

18 Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proc. 37th Annual ACM Symposium on the Theory of Computing*, pages 202–208, 2005.

19 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1161–1178, 2010.

20   Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proc. 29th ACM Symposium on Principles of Database Systems*, pages 41–52, 2010.

21   Hartmut Klauck and Ved Prakash. Streaming computations with a loquacious prover. In *Proc. 4th Conference on Innovations in Theoretical Computer Science*, pages 305–320, 2013.

22   Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem. In *Automata, Languages, and Programming - 41st International Colloquium (ICALP)*, volume 8572 of *LNCS*, pages 919–930, 2014.

23   Feifei Li, Ke Yi, Marios Hadjieleftheriou, and George Kollios. Proof-infused streams: Enabling authentication of sliding window queries on streams. In *Proc. 33rd International Conference on Very Large Data Bases*, pages 147–158, 2007.

24   Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

25   Jayadev Misra and David Gries. Finding repeated elements. *Sci. Comput. Program.*, 2(2):143–152, 1982.

26   Jelani Nelson, Huy L. Nguyên, and David P. Woodruff. On deterministic sketching and streaming for sparse recovery and norm estimation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012*, volume 7408 of *Lecture Notes in Computer Science*, pages 627–638, 2012.

27   Stavros Papadopoulos, Yin Yang, and Dimitris Papadias. Cads: Continuous authentication on data streams. In *Proc. 33rd International Conference on Very Large Data Bases*, pages 135–146, 2007.

28   Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

29   Justin Thaler. Data stream verification. In *Encyclopedia of Algorithms*, pages 494–499. Springer Berlin Heidelberg, 2016.

30   Justin Thaler. Semi-streaming algorithms for annotated graph streams. In *Proc. 43rd International Colloquium on Automata, Languages and Programming*, pages 59:1–59:14, 2016.

31   Peter A. Tucker, David Maier, Lois M. L. Delcambre, Tim Sheard, Jennifer Widom, and Mark P. Jones. Punctuated data streams, 2005.

32   David P. Woodruff. Optimal space lower bounds for all frequency moments. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 167–175, 2004.

33   Ke Yi, Feifei Li, Marios Hadjieleftheriou, George Kollios, and Divesh Srivastava. Randomized synopses for query assurance on data streams. In *Proc. 24th International Conference on Data Engineering*, pages 416–425, 2008.