Electronic Colloquium on Computational Complexity, Report No. 22 (2012)

Annotations in Data Streams

Amit Chakrabarti* Graham Cormode† Andrew McGregor‡ Justin Thaler §

March 14, 2012

Abstract

The central goal of data stream algorithms is to process massive streams of data using *sublinear* storage space. Motivated by work in the database community on outsourcing database and data stream processing, we ask whether the space usage of such algorithms can be further reduced by enlisting a more powerful "helper" who can *annotate* the stream as it is read. We do not wish to blindly trust the helper, so we require that the algorithm be convinced of having computed a correct answer. We show upper bounds that achieve a non-trivial tradeoff between the amount of annotation used and the space required to verify it. We also prove lower bounds on such tradeoffs, often nearly matching the upper bounds, via notions related to Merlin-Arthur communication complexity. Our results cover the classic data stream problems of selection, frequency moments, and fundamental graph problems such as triangle-freeness and connectivity. Our work is also part of a growing trend — including recent studies of multi-pass streaming, read/write streams and randomly ordered streams — of asking more complexity-theoretic questions about data stream processing. It is a recognition that, in addition to practical relevance, the data stream model raises many interesting theoretical questions in its own right.

*Dartmouth College. ac@cs.dartmouth.edu

†AT&T Labs–Research. graham@research.att.com

‡University of Massachusetts Amherst. mcgregor@cs.umass.edu. Supported by NSF CAREER Award CCF-0953754.

§Harvard University, School of Engineering and Applied Sciences. jthaler@seas.harvard.edu. Supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

ISSN 1433-8092

1 Introduction

The data stream model has become a popular abstraction when designing algorithms that process network traffic and massive data sets [4, 32]. The computational restrictions that define this model are severe: algorithms must use a relatively small amount of working memory and process the input in whatever order it arrives. This captures constraints in high-throughput data processing settings. For example, network monitoring often requires (near) real-time response to anomalies and hence traffic must be processed as it arrives, rather than being stored and processed offline. For massive data sets stored in external memory, being able to process the data in any order avoids the I/O bottlenecks that may arise with algorithms that assume random access. Unfortunately, while some problems admit efficient streaming algorithms, many others provably require a lot of working memory or multiple passes over the data, which typically is not feasible.

This paper considers the potential for off-loading stream computation to a more powerful "helper" so that single pass, small-space stream computation is possible even for such "hard" functions. The additional power of the helper can arise in a variety of situations, e.g., multiple processing units, special purpose hardware, or a third party who provides a commercial stream processing service. This last case has recently garnered attention in the context of outsourcing database processing [38, 41, 46]. A key issue is that we do not want to blindly trust the helper: hardware faults or outright deception by a third-party would lead to incorrect results. So our protocols must have sufficient information contained in the help to allow the "verifier" to be convinced that they have obtained the correct answer. We think of this help as annotations augmenting the original stream. Our goal is to design protocols so that the verifier finds the correct answer with an honest helper, and is likely not fooled by a dishonest helper. The primary metrics are the amount of annotation provided by the helper and the amount of working space used by the verifier.

Our approach is naturally related to Interactive Proofs and Merlin-Arthur communication protocols [1, 5, 36] but differs in two important regards. Firstly, the verifier must process both the original data and the advice provided by the helper under the usual restrictions of the data stream model. Secondly, we focus on annotations that can be provided *online*, i.e., annotation that only depends on data that has arrived before the annotation is written. Note that in Merlin-Arthur communication, it is assumed that the helper is omniscient and that the advice he provides can take into account data held by any of the players. In the stream model, this would correspond to *prescience*, where the annotation in the stream at any particular position may depend on data that is yet to arrive. In contrast, we are primarily interested in designing algorithms with online annotation; this corresponds to a helper who sees the data concurrently with the verifier.

1.1 Our Contributions

Our first contribution is to formally define the relevant models: traditional and online Merlin-Arthur communication, and streaming models with either prescient or online annotations. We then investigate the complexity of a range of problems in these models, including selection, frequency moments, and graph problems such as triangle-counting and connectivity. Estimating frequency moments in particular has become a canonical problem when exploring variants of the data stream model such as random order streams [11] and read/write streams [7].

We now give an overview of our results. We use the shorthand "(h,v)-scheme" for an $O(v)$-space streaming algorithm that uses $O(h)$ bits of annotation; a scheme could be either prescient or online. In general, our streams have length m, and consist of tokens from the universe $[n] := \{1,2,\ldots,n\}$; to simplify the statement of bounds, we assume than m and n are polynomially related, and in particular that $\log m = \Theta(\log n)$. In the case of graph streams, we consider tokens from the universe $[n] \times [n]$. We use $\mathbb{Z}_+$ to denote the set of non-negative integers.

Selection. The problem of finding the median of m values in the range $[n]$ highlights the difference between prescient and online annotation. For arbitrary positive integers h and v, with $hv \geq m$, we present an online $(h\log n, v\log n)$-scheme. Furthermore, we show that this trade-off is optimal up to polylogarithmic factors. In contrast, a trivial $O(\log n)$-space algorithm can verify $O(\log n)$ bits of prescient annotation, implying a prescient $(\log n, \log n)$-scheme.

Frequency Moments and Frequent Items. We next consider properties of $\{f_i\}_{i\in[n]}$ where f_i is the frequency of the token "i" in the stream. For arbitrary integers h and v, with $hv \geq n$, we present an online $(\phi^{-1}h\log n, v\log n)$-scheme that computes the set of tokens whose frequency exceeds ϕm. Also, for any $0 < \varepsilon < \phi/2$, we give an online $(\varepsilon^{-1}\log^2 n, \log n)$-scheme that computes a set of tokens that includes $\{i : f_i \geq \phi m\}$ and is disjoint from $\{i : f_i \leq (\phi-\varepsilon)m\}$. This algorithm relies on a powerful way that annotation can be used in conjunction with sketch-based algorithms.

We present an online $(k^2 h\log n, kv\log n)$-scheme that computes the kth frequency moment $F_k := \sum_i f_i^k$ exactly, where k is a positive integer. This trade-off is optimal up to polylogarithmic factors even if the algorithm is allowed to use prescient annotation. To prove this, we present the first Merlin-Arthur communication bounds for multi-party set-disjointness. Additionally, we generalize the scheme for F_k to any *frequency-based function*, i.e., a function of the form $\sum_{i\in[n]} g(f_i)$ for some $g : \mathbb{Z}_+ \to \mathbb{Z}_+$. Assuming $m = O(n)$, we obtain a prescient $(n^{2/3}\log n, n^{2/3}\log n)$-scheme and an online $(n^{3/4}\log n, n^{3/4}\log n)$-scheme for this important class of functions, as well as improved schemes for functions based on low frequencies and for skewed data streams.

Graph Problems. For graphs defined by streams of m edges on n vertices, we show that only $O(\log n)$ space is needed by the verifier to determine whether a graph is connected, is triangle-free, or is bipartite, with online annotation proportional to the input size in each case. We show that our algorithms are optimal in many cases.

For any h and v, with $hv \geq n^3$, we also present an online $(h\log n, v\log n)$-scheme for counting triangles in the graph. Additionally, for any h and v, with $hv \geq n^2$, we present online $(h\log n, v\log n)$-schemes for determining whether a graph is connected or bipartite. Finally, for any h and v, with $hv \geq n^2$, we present online $(h\log n, v\log n)$-schemes for solving bipartite perfect matching. This latter scheme achieves essentially optimal tradeoffs between annotation length and space usage for the verifier.

1.2 Related Work

When multiple passes over the input are allowed, it is natural to consider annotations that can be written to the "input tape" by the stream algorithm and which are then available to the algorithm in subsequent passes [3, 15, 23, 24]. The read/write stream model, which provides both multiple passes and multiple working tapes, can be viewed as a natural extension of the multi-pass annotation model [7, 8, 31]. However, such annotations are of no use if only a single pass over the input is allowed.

Few examples of prior work have explicitly considered annotations that are provided by an (untrusted) third party. Gertner et al. [29] showed that the set of languages recognized by a verifier with logarithmic space, given annotation polynomial in the input size, is exactly NP. In contrast, our focus is on the case where the annotation is (sub)linear in the input size and can be provided online; the distinction between prescient and online annotation was not relevant in their results because with polynomial annotation, the entire input could be repeated. Feigenbaum et al. [27] observe that a logarithmic space verifier can check a linear space annotation for the disjointness problem. In communication complexity, the role of non-deterministic

advice has been studied more extensively: see, e.g., [5, 37]. The work of Aaronson and Wigderson [1] and Klauck [36] are particularly relevant: they resolve the MA complexity of two-party set disjointness. We extend some of their techniques to our streaming model. Goldwasser et al. [30] consider the question of which computations can be verified relatively efficiently while permitting multiple rounds of interaction between the parties.

There has also been more applied work which implicitly defines annotation schemes. Tucker et al. [45] considered *stream punctuations*, which, in our terminology, are simple prescient annotations, indicating facts such as that there are no more tuples relevant to timestamp t in the remainder of the stream. Yi et al. [46], in their work on stream outsourcing, study the problem of verifying that a claimed "grouping" corresponds to the input data. They solve exact and approximate versions of the problem by using a linear amount of annotation. Lastly, the work of Li et al. [38] on *proof infused streams* answers various selection and aggregation queries over sliding windows with logarithmic space and linear annotation. However, a critical difference is that Li et al. require that the helper and verifier agree on a one-way hash function, for which it is assumed the helper cannot find collisions. Our results are in a stronger model without this assumption.

Subsequent Work. The line of work described in this paper was begun when the first three authors presented a preliminary version of some of these results at ICALP 2009 [12]. Work subsequent to that paper has further studied the protocols for graph computations in this model [17]. In particular, it is observed that given any deterministic RAM algorithm with running time R, there exists an online $((m+R)\log n, \log n)$-scheme that simulates the algorithm in the annotation model. This implies alternate proofs for the existence of online $(m\log n, \log n)$-schemes for bipartite perfect matchings, bipartiteness, and connectivity.

Other subsequent works have built upon the conference version of the present work. Cormode, Thaler and Yi [21] have extended the model considered in this paper to allow for multiple rounds of interaction between helper and verifier, and provided protocols achieving exponentially smaller space and communication costs than those possible in our annotation model. Most recently, Cormode, Mitzenmacher and Thaler [19] have performed an empirical evaluation of many techniques in the literature on interactive proofs, and demonstrated genuine scalability of several of the protocols put forth in the present work as well as protocols from [17] and [21].

2 Models and Preliminaries

In this section, we first recall the definition of Merlin-Arthur communication and then present an online variant which restricts the advice given by Merlin. We then present the formal definitions of the annotated data stream models and state some basic lemmas.

2.1 Communication Models

Let $f : X_1 \times \cdots \times X_t \to \{0,1\}$ be a function, where each X_i is a finite set. This naturally gives a t-player number-in-hand communication problem, where Player i holds an input $x_i \in X_i$ and the players wish to output $f(x_1,\ldots,x_t)$ correctly, with high probability.

MA Communication. We first consider a variant of this communication model. A Merlin-Arthur protocol (henceforth, "MA protocol") for f is one that involves the usual t players, plus a "super-player," called Merlin, who knows the entire input $\mathbf{x} = (x_1,\ldots,x_t)$. The protocol works as follows: first Merlin deterministically

writes a help message $\mathfrak{h}$ on the blackboard, and then Players 1 through t run a randomized protocol $\mathcal{P}$, using a public random string R, eventually outputting a bit $\text{out}(\mathcal{P};\mathbf{x},R,\mathfrak{h})$. To clarify, R is not known to Merlin at the time he writes $\mathfrak{h}$. An MA protocol is δ-error if there exists a function $\mathfrak{h} : X_1 \times \ldots \times X_t \to \{0,1\}^*$, such that:

1. If $f(\mathbf{x}) = 1$ then $\Pr_R[\text{out}(\mathcal{P};\mathbf{x},R,\mathfrak{h}(\mathbf{x})) = 0] \leq \delta$.
2. If $f(\mathbf{x}) = 0$ then $\forall\, \mathfrak{h}' \in \{0,1\}^* : \Pr_R[\text{out}(\mathcal{P};\mathbf{x},R,\mathfrak{h}') = 1] \leq \delta$.

We define $\text{err}(\mathcal{P})$ to be the minimum δ such that the above conditions are satisfied. We also define the *help cost* $\text{hcost}(\mathcal{P})$ to be the maximum length of $\mathfrak{h}(\mathbf{x})$, over all $\mathbf{x}$, and the *verification cost* $\text{vcost}(\mathcal{P})$ to be the maximum number of bits communicated by Players 1 through t over all $\mathbf{x}$ and R. To avoid boundary cases, we insist that both of these costs are at least 1 for any protocol, i.e., we consider traditional protocols where no explicit help is provided to have $\text{hcost} = 1$, rather than 0. We define the δ-error MA-complexity of f as $\text{MA}_\delta(f) = \min\{\text{vcost}(\mathcal{P}) + \text{hcost}(\mathcal{P}) : \mathcal{P} \text{ is an MA protocol for } f \text{ with } \text{err}(\mathcal{P}) \leq \delta\}$. Further, we define $\text{MA}(f) = \text{MA}_{1/3}(f)$.

Online MA Communication. We also consider a variant of the above model, specific to *one-way protocols* (i.e., protocols where the players speak once each, in increasing order), where Merlin constructs t help messages $\mathfrak{h}_1,\ldots,\mathfrak{h}_t$ so that the ith message is a function of only the first i inputs. The message $\mathfrak{h}_i$ is revealed privately to the ith player. To make this precise we need to amend the definition of δ-error: An online MA protocol is δ-error if there exists a family of functions $\mathfrak{h}_i : X_1 \times \ldots \times X_i \to \{0,1\}^*$, such that:

1. If $f(\mathbf{x}) = 1$ then $\Pr_R[\text{out}(\mathcal{P};\mathbf{x},R,\mathfrak{h}_1(x_1),\mathfrak{h}_2(x_1,x_2),\ldots,\mathfrak{h}_t(x_1,\ldots,x_t)) = 0] \leq \delta$.
2. If $f(\mathbf{x}) = 0$ then $\forall\, \mathfrak{h}' = (\mathfrak{h}'_1,\mathfrak{h}'_2,\ldots,\mathfrak{h}'_t) \in (\{0,1\}^*)^t : \Pr_R[\text{out}(\mathcal{P};\mathbf{x},R,\mathfrak{h}') = 1] \leq \delta$.

We define the help cost, $\text{hcost}(\mathcal{P})$, to be the maximum of $\sum_{i\in[t]} |\mathfrak{h}_i(x_1,\ldots,x_i)|$, over all $\mathbf{x}$. We define $\text{err}(\mathcal{P})$, and $\text{vcost}(\mathcal{P})$ as for MA. Define $\text{MA}^{\rightarrow}_\delta(f) = \min\{\text{hcost}(\mathcal{P}) + \text{vcost}(\mathcal{P}) : \mathcal{P} \text{ is an online MA protocol for } f \text{ with } \text{err}(\mathcal{P}) \leq \delta\}$ and write $\text{MA}^{\rightarrow}(f) = \text{MA}^{\rightarrow}_{1/3}(f)$.

2.2 Data Stream Models

We now define our annotated data stream models. Recall that a (usual) data stream algorithm computes a function f of an input sequence $\mathbf{x} \in \mathcal{U}^m$, where $\mathcal{U}$ is some universe, such as $\{0,1\}$ or $[n]$: the algorithm uses a limited amount of working memory and has access to a random string. The function f may or may not be Boolean: for non-Boolean f we often consider a notion of approximation: we say f is computed correctly if the answer returned is in some pre-defined set $C(f(\mathbf{x}))$, e.g., $\{a : |a - f(\mathbf{x})| \leq \varepsilon |f(\mathbf{x})|\}$.

An annotated data stream algorithm, or a *scheme*, is a pair $\mathcal{A} = (\mathfrak{h},\mathcal{B})$, consisting of a (deterministic) help function $\mathfrak{h}$ and a data stream algorithm $\mathcal{B}$. We think of $\mathfrak{h}$ as decomposed into $(\mathfrak{h}_1,\ldots,\mathfrak{h}_m)$, where $\mathfrak{h}_i : \mathcal{U}^m \to \{0,1\}^*$; the function $\mathfrak{h}_i$ determines the annotation supplied to $\mathcal{B}$ after the ith token x_i. That is, $\mathfrak{h}$ acts on $\mathbf{x}$ to create an *annotated stream* $\mathbf{x}^{\mathfrak{h}}$ defined as follows:

$$\mathbf{x}^{\mathfrak{h}} := (x_1, \mathfrak{h}_1(\mathbf{x}), x_2, \mathfrak{h}_2(\mathbf{x}), \ldots, x_m, \mathfrak{h}_m(\mathbf{x})).$$

Note that this is a stream over $\mathcal{U} \cup \{0,1\}$, of length $m + \sum_i |\mathfrak{h}_i(\mathbf{x})|$. The algorithm $\mathcal{B}$, which uses w bits of working memory and has oracle access to a random string R, then processes this annotated stream, eventually giving an output $\text{out}(\mathcal{B};\mathbf{x}^{\mathfrak{h}},R)$.

Prescient Schemes. The scheme $\mathcal{A} = (\mathfrak{h}, \mathcal{B})$ is said to be a δ-error prescient scheme for the function f if the following conditions hold:

1. For all $\mathbf{x} \in \mathcal{U}^m$, we have $\Pr_R[\text{out}(\mathcal{B}; \mathbf{x}^{\mathfrak{h}}, R) \notin C(f(\mathbf{x}))] \leq \delta$.
2. For all $\mathbf{x} \in \mathcal{U}^m$, $\mathfrak{h}' = (\mathfrak{h}'_1, \mathfrak{h}'_2, \ldots, \mathfrak{h}'_m) \in (\{0,1\}^*)^m$, we have $\Pr_R[\text{out}(\mathcal{B}; \mathbf{x}^{\mathfrak{h}'}, R) \notin C(f(\mathbf{x})) \cup \{\perp\}] \leq \delta$.

Two things are worth noting. First, this definition allows the annotation $\mathfrak{h}_i(\mathbf{x})$ to depend on the entire stream $\mathbf{x}$, thus modelling prescience. Second, it allows (but does not force) the protocol to output $\perp$ if the annotation does not agree with $\mathfrak{h}$.

We define $\text{err}(\mathcal{A})$ to be the minimum δ such that the above conditions are satisfied. We define the *help cost* $\text{hcost}(\mathcal{A}) := \max_{\mathbf{x}} \sum_i |\mathfrak{h}_i(\mathbf{x})|$, and the *verification cost* $\text{vcost}(\mathcal{A}) = w$. We say that $\mathcal{A}$ is a prescient (h,v)-scheme if $\text{hcost}(\mathcal{A}) = O(h)$, $\text{vcost}(\mathcal{A}) = O(v)$ and $\text{err}(\mathcal{A}) \leq \frac{1}{3}$.

Online Schemes. The scheme $\mathcal{A} = (\mathfrak{h}, \mathcal{B})$ is said to be a δ-error online scheme for f if, in addition to the conditions in the previous definition, the function $\mathfrak{h}_i$ depends only on $(x_1, \ldots, x_i)$. We define hcost and vcost as above, and say that $\mathcal{A}$ is an *online* (h,v)*-scheme* if $\text{hcost}(\mathcal{A}) = O(h)$, $\text{vcost}(\mathcal{A}) = O(v)$, and $\text{err}(\mathcal{A}) \leq \frac{1}{3}$.

In order to simplify the statements of bounds, we assume throughout that universe size and stream length are polynomially related, and thus $\log m = \Theta(\log n)$. In a few cases, we use the stronger assumption that $m = O(n)$; in these cases, we state this assumption explicitly.

2.3 Background Preliminaries

In multiple places we make use of basic fingerprinting techniques which enable a verifier to test whether two large streams represent the same object, using small space. Let $\mathbb{F}_q$ denote the finite field with q elements (whenever it exists). Let $A = \langle a_1, \ldots, a_m \rangle$ denote a data stream, with each $a_i \in [n]$. Then A implicitly defines a frequency distribution $\mathbf{f}(A) := (f_1, \ldots, f_n)$, where $f_j = |\{i \in [m] : a_i = j\}|$ is the frequency of the token "j" in A. We can then fingerprint this vector by computing the following quantity.

Definition 1 (Basic Fingerprint)**.** Let $\mathbf{f} = (f_1, \ldots, f_n) \in \mathbb{Z}_+^n$ be a vector, let q be a prime, and let $r \in \mathbb{F}_q$. The quantity $\text{BF}_q(r, \mathbf{f}) := \prod_{j=1}^n (r-j)^{f_j}$, computed over $\mathbb{F}_q$, is called a *basic fingerprint* of $\mathbf{f}$.

To compute basic fingerprints, we choose q based on an *a priori* bound m on $\|\mathbf{f}\|_1$. The following lemma collects the key properties of these fingerprints.

Lemma 2.1. *Let $q \geq m$ be a prime, and choose r uniformly at random from $\mathbb{F}_q$. Given an input stream A of length m, the fingerprint $\text{BF}_q(r, \mathbf{f}(A))$ can be computed using $O(\log q)$ storage. Suppose $\mathbf{f}' \in \mathbb{Z}_+^n$ is a vector with $\mathbf{f}' \neq \mathbf{f}(A)$ and $\|\mathbf{f}'\|_1 \leq m$. Then the "collision probability" $\Pr_{r \in_R \mathbb{F}_q}[\text{BF}_q(r, \mathbf{f}') = \text{BF}_q(r, \mathbf{f}(A))] \leq m/q$.*

Proof. To compute the fingerprint in streaming fashion, express $\text{BF}_q(r, \mathbf{f}(A)) = \prod_{i=1}^m (r - a_i)$. The bound on the collision probability follows from the fact that for any $\mathbf{f} \in \mathbb{Z}_+^n$, the polynomial $\text{BF}_q(X, \mathbf{f}) \in \mathbb{F}_q[X]$ has degree at most $\|\mathbf{f}\|_1$. □

Further, on several occasions, we use the standard technique of linear sketching. We define an *integer linear sketch* broadly as any summary $\mathbf{v} \in \mathbb{Z}^s$ which can be computed as $\mathbf{v} = S\mathbf{f}(A)$, where $S \in \mathbb{Z}^{s \times n}$ is a "sketch matrix" with integral entries and $s \ll n$. Such sketches include instantiations of the Johnson-Lindenstrauss transform [33], Count-Sketch [14], and Count-Min [20]. Each stream token j increments $\mathbf{v}$ by $S\mathbf{e}_j$, where $\mathbf{e}_j \in \mathbb{Z}^n$ is the vector that is 1 in location j and 0 elsewhere. Typically, S has a compact implicit representation.

In particular, the Count-Sketch [14] defines a *basic sketch* of length w via two pairwise independent hash functions $b_\ell : [n] \to [w]$, and $c_\ell : [n] \to \{-1,+1\}$. The sketch vector $\mathbf{v}$ is defined by $\mathbf{v}_{\ell,j} = \sum_{i:b_\ell(i)=j} f_i c_\ell(i)$. A basic estimate of the frequency of item i is $\hat{f}_{i,\ell} = c_\ell(i)\mathbf{v}_{\ell,b_\ell(i)}$. This satisfies $|\hat{f}_{i,\ell} - f_i| = O((F_2(A)/w)^{1/2})$ with constant probability. To reduce the error probability, one takes the median of the basic estimates from d basic sketches with independent pairs of hash functions: $\hat{f}_i = \text{median}_{1\le\ell\le d}\, \hat{f}_{i,\ell}$. Count-Min is essentially Count-Sketch with $c_\ell(i) := 1$ for all ℓ. It promises $|\hat{f}_{i,\ell} - f_i| = O(F_1(A)/w)$ [20]. Here, $F_1(A)$ and $F_2(A)$ denote the first and second frequency moments of A, respectively.

3 Warm-Up: Index, Selection, and Frequent Items

3.1 Index and Selection

In this section, we present an online scheme for the SELECTION problem: Given desired rank $\rho \in [m]$, output an item a_k from the stream $A = \langle a_1, \ldots, a_m\rangle \in [n]^m$, such that $|\{i : a_i < a_k\}| < \rho$ and $|\{i : a_i > a_k\}| \le m - \rho$. An easy *prescient* $(\log n, \log n)$-scheme is for the helper to give an answer s that is claimed to be a_k, as annotation at the start of the stream. The verifier need only count how many items in the stream are (a) smaller than s and (b) greater than s. The verifier then outputs s if the rank of s satisfies the necessary conditions, and outputs $\perp$ otherwise.

However, our goal is to present (almost) matching upper and lower bounds when only *online* annotation is allowed. To do this, we first consider the online MA complexity of the communication problem of INDEX: Alice holds a string $x \in \{0,1\}^N$, Bob holds an integer $i \in [N]$, and the goal is for Bob to output $\text{INDEX}(x,i) := x_i$. The lower bound for SELECTION will follow from the lower bound for INDEX and a key idea for the SELECTION upper bound is taken from the communication protocol for INDEX seen in the proof of the following theorem.

Theorem 3.1 (Online MA complexity of INDEX)**.** *Let h and v be integers such that $hv \ge N$. There is an online MA protocol $\mathcal{P}$ for* INDEX*, with* $\text{hcost}(\mathcal{P}) \le h$ *and* $\text{vcost}(\mathcal{P}) = O(v \log h)$*. Futhermore, any online MA protocol $\mathcal{Q}$ for* INDEX *must have* $\text{hcost}(\mathcal{Q})\,\text{vcost}(\mathcal{Q}) = \Omega(N)$*. Thus, in particular,* $\text{MA}^{\rightarrow}(\text{INDEX}) = \tilde{\Theta}(\sqrt{N})$.

Proof. For the lower bound, we use a online MA protocol $\mathcal{Q}$ to build a randomized one-way INDEX protocol $\mathcal{Q}'$. Let $h = \text{hcost}(\mathcal{Q})$. Let $\mathcal{B}(n,p)$ denote the binomial distribution with parameters n and p, and let k be the smallest integer such that $X \sim \mathcal{B}(k, \frac{1}{3}) \Rightarrow \Pr[X > k/2] \le 2^{-h}/3$. A standard tail estimate gives $k = \Theta(h)$. Let $a(x,R)$ denote the message that Alice sends in $\mathcal{Q}$ when her random string is R, and let $b(\mathfrak{a}, i, \mathfrak{h})$ be the bit Bob outputs upon receiving message $\mathfrak{a}$ from Alice and $\mathfrak{h}$ from Merlin. In the protocol $\mathcal{Q}'$, Alice chooses k independent random strings $R_1, \ldots, R_k$ and sends Bob $a(x,R_1), \ldots, a(x,R_k)$. Bob then outputs 1 iff there exists a h-bit string $\mathfrak{h}$ such that $\text{MAJORITY}\,(b(a(x,R_1),i,\mathfrak{h}), \ldots, b(a(x,R_k),i,\mathfrak{h})) = 1$. Let C be the number of bits communicated in this protocol. Clearly, $C \le k \cdot \text{vcost}(\mathcal{Q}) = O(\text{hcost}(\mathcal{Q})\,\text{vcost}(\mathcal{Q}))$. We claim that $\mathcal{Q}'$ is a $\frac{1}{3}$-error protocol for INDEX whence, by a standard lower bound (see, e.g., Ablayev [2]), $C = \Omega(N)$.

To prove the claim, consider the case when $x_i = 1$. By the correctness of $\mathcal{Q}$ there exists a suitable help message $\mathfrak{h}$ from Merlin that causes $\Pr[b(a(x,R),i,\mathfrak{h}) = 0] \le \frac{1}{3}$. Thus, by construction and our choice of k, the probability that Bob outputs 0 in $\mathcal{Q}'$ is at most $2^{-h}/3$. Now suppose $x_i = 0$. Then, *every* possible message $\mathfrak{h}$ from Merlin satisfies $\Pr[b(a(x,R),i,\mathfrak{h}) = 1] \le \frac{1}{3}$. Arguing as before, and using a union bound over all 2^h possible messages $\mathfrak{h}$, we see that Bob outputs 1 with probability at most $2^h \cdot 2^{-h}/3 = \frac{1}{3}$.

The upper bound follows as a special case of the two-party set-disjointness protocol in [1, Theorem. 7.4] since the protocol there is actually online. We give a more direct protocol which establishes intuition for our SELECTION result. Write Alice's input string x as $x = y^{(1)} \cdots y^{(v)}$, where each $y^{(j)}$ is a string of at most h bits,

and fix a prime q with $3h < q < 6h$. Let $y^{(k)}$ be the substring that contains the desired bit x_i. Merlin sends Bob a string z of length at most h, claiming that it equals $y^{(k)}$. Alice picks a random $r \in \mathbb{F}_q$ and sends Bob r and the strings $\mathrm{BF}_q(r, y^{(1)}), \ldots, \mathrm{BF}_q(r, y^{(v)})$, thus communicating $O(v \log q) = O(v \log h)$ bits. Bob checks if $\mathrm{BF}_q(r,z) = \mathrm{BF}_q(r, y^{(k)})$, outputting 0 if not. If the check passes, Bob assumes that $z = y^{(k)}$, and outputs x_i from z under this assumption. By Lemma 2.1, the error probability is at most $h/q < 1/3$. □

It is worth making the following two remarks on the above proof.

1. The above lower bound argument in fact shows that an online MA protocol $\mathcal{P}$ for an *arbitrary* two-party communication problem f satisfies $\mathrm{hcost}(\mathcal{P})\,\mathrm{vcost}(\mathcal{P}) = \Omega(\mathrm{R}^{\rightarrow}(f))$, where $\mathrm{R}^{\rightarrow}(f)$ is the one-way, randomized communication complexity of f. Thus, $\mathrm{MA}^{\rightarrow}(f) = \Omega(\sqrt{\mathrm{R}^{\rightarrow}(f)})$.

2. The upper bound for INDEX presented above works more or less unchanged when Alice's string is in Σ^N, for an arbitrary finite alphabet Σ. In view of Lemma 2.1, one simply needs to choose the prime q such that $3|\Sigma|h < q < 6|\Sigma|h$ to bound the error probability below $1/3$. This leads to a protocol $\mathcal{P}$ with $\mathrm{hcost}(\mathcal{P}) \leq h \log |\Sigma|$ and $\mathrm{vcost}(\mathcal{P}) = O(v(\log |\Sigma| + \log h))$. Henceforth, we shall refer to this generalized protocol simply as "the INDEX protocol" — the alphabet Σ will usually be clear from the context.

Theorem 3.2. *For all h, v such that $hv \geq n$, there is an online $(h \log m, v \log m)$-scheme for* SELECTION. *Furthermore, any online (h,v)-scheme for* SELECTION *must have $hv = \Omega(m)$.*

Proof. Conceptually, the verifier builds a vector $\mathbf{r} = (r_1, \ldots, r_n) \in \mathbb{Z}_+^n$ where $r_k = |\{j \in [m] : a_j < k\}|$. This is done by inducing a new stream A' from the input stream A: each token a_j in A causes virtual tokens $a_j + 1, a_j + 2, \ldots, n$ to be inserted into A'. Then $\mathbf{r} = \mathbf{f}(A')$; note that $\|\mathbf{r}\|_1 = O(m^2)$. We apply the INDEX protocol to this vector, with $q = \Theta(m^2)$ to retrieve the ranks of elements surrounding the claimed median. This information is sufficient to check that s has the claimed rank.

For the lower bound, we use a standard reduction from the INDEX problem. Take $N = m$. Given the string $x \in \{0,1\}^m$, Alice transforms it into the stream over $[2m]$ whose jth token is $a_j = 2j - x_j$, for each j. Given the index $i \in [m]$, Bob transforms it into a stream consisting of i copies of $2m$ and $m - i$ copies of 1. Consequently, the median of the combined length-$(2m)$ stream is $2i - x_i$, from which the value of x_i can be recovered. To complete the proof, observe that any online scheme to compute this median would imply an online MA protocol for INDEX with the same cost; and that all players can perform this reduction online without extra space or annotation. □

Notice that in the above scheme the information computed by the verifier is independent of ρ, the rank of the desired element. Therefore these algorithms work even when ρ is revealed at the end of the stream.

3.2 A First Result for Frequent Items

The ϕ-heavy hitters (also known as the frequent items) are those items whose frequency of occurrence in the data stream exceeds a ϕ fraction of the total count. This problem has a long history in the data streams literature. In the traditional data stream model exact computation of heavy hitters requires linear space [40]. As a result, many algorithms have been developed which recover approximate heavy hitters from a data stream [14, 20].

In order to identify the heavy hitters, a prescient helper can list the set of claimed frequent items, along with their frequencies, for the verifier to check against the stream. But we must also ensure that the helper is not able to omit any items whose frequencies exceed the threshold.

Theorem 3.3. *For all h, v such that $hv \geq n$, there is an online $(h\phi^{-1}\log^2 n, v\log n)$-scheme and a prescient $(\phi^{-1}\log^2 n, \phi^{-1}\log^2 n)$-scheme for demonstrating the ϕ-heavy hitters.*

Proof. Given the threshold $T = \phi m$, the set of heavy hitters is $\{j : f_j > T\}$. We impose a binary tree $\mathcal{T}$ over the data, whose leaves are the elements of the universe $[n]$, and partition the $(2n-1)$ nodes of $\mathcal{T}$ into v *groups* $G_1, \ldots, G_v$, with each $|G_i| \leq 2h$. For each node w of $\mathcal{T}$, let $p(w)$ denote the parent of w, and let $L(w)$ denote the set of leaves of the subtree of $\mathcal{T}$ rooted at w. We define $\hat{f}(w) = \sum_{i \in L(w)} f_i$.

The $\hat{f}$-values for the nodes in each group G_i form a vector with entries in $\{0, 1, \ldots, m\}$. As the verifier processes the stream it maintains an $O(\log n)$-bit basic fingerprint of each such vector; this is easy to do since each token arrival simply causes a linear update to each vector. Once the end of the stream is reached, the helper can then convince the verifier of any $\hat{f}(w)$ value using the INDEX protocol: he simply supplies the vector for the group G_i that contains w, using at most $2h\log(m+1) = O(h\log n)$ bits of annotation. In particular, he can identify all the heavy hitters. But he must also convince the verifier that no heavy hitters have been omitted.

To this end, we consider a *witness set*, W, of nodes of $\mathcal{T}$ which together cover the universe. The set W, given threshold T, consists of all leaves ℓ with $\hat{f}(\ell) > T$, plus all nodes u such that $\hat{f}(u) \leq T$ but $\hat{f}(p(u)) > T$. Each node of the latter type is witness to the fact that no leaves $j \in L(u)$ can have $f_j > T$. The sets $L(u)$ for such u together with $\{j : f_j > T\}$ cover all of $[n]$. Further, because of the lower bound on $\hat{f}(p(u))$, there can be at most $2\phi^{-1}$ such nodes u at any level of $\mathcal{T}$, as the sum of $\hat{f}(w)$ over all nodes w at the parent level is exactly m. Hence $|W| = O(\phi^{-1}\log n)$.

The prover presents the verifier with each node u in W, in increasing order of $\min L(u)$, together with a convincing proof of the value of $\hat{f}(u)$. The verifier, besides checking the proofs using the stored fingerprints, checks that the sets $L(u)$ do cover all of $[n]$ (outputting $\perp$ if they do not) and outputs those u that are leaves of $\mathcal{T}$ with $\hat{f}(u) > T$. In total, hcost $= O(|W| \cdot h\log n) = O(h\phi^{-1}\log^2 n)$ and vcost $= O(v\log n)$. Note that the stated vcost does not explicitly account for the verifier storing the $O(\phi^{-1}\log n)$ claimed heavy hitters, as in some settings (e.g., Theorem 4.5, later in this paper) this is not required.

In the prescient case, the helper provides W upfront, which requires $O(|W|\log n) = O(\phi^{-1}\log^2 n)$ bits of annotation. The verifier stores it, and then computes all $\hat{f}$-values for nodes in W, checking that these satisfy the requirements on a witness set. In this case, the stated vcost does account for the verifier storing the $O(\phi^{-1}\log n)$ claimed heavy hitters. □

In Section 6, we return to this problem, and present more involved protocols with a lower cost, and consider approximate variations. Specifically, Theorem 6.1 shows how the size of the witness set W can be reduced, and Theorem 6.2 shows how the exact frequency vector can be replaced with a more compact sketched vector.

4 Frequency Moments and Generalizations

In this section we continue the study of properties of the frequency distribution $\mathbf{f}(A) = (f_1, \ldots, f_n)$ of a given stream A. In particular, we study the computation of frequency moments, which has a long history in the data streams literature, like the frequent items problem discussed earlier.

Definition 2. The kth frequency moment of the stream A is defined as $F_k = F_k(A) := \sum_{j \in [n]} f_j^k = \|\mathbf{f}(A)\|_k^k$. Slightly abusing notation, we also define $F_k(\mathbf{v}) := \|\mathbf{v}\|_k^k$ for a vector $\mathbf{v}$.

It is well known that in the traditional data stream model, exact computation of F_k ($k \neq 1$) requires $\Omega(n)$ space. Even constant factor approximation requires $\Omega(n^{1-2/k})$ space for $k \geq 2$ [13].

4.1 Schemes for Frequency Moments

We now show a family of algorithms that exhibit an optimal verification/annotation tradeoff for the exact computation of F_k. Our algorithm is inspired by the "algebrization" results of Aaronson and Wigderson [1] but the key idea can be traced back to classic interactive proof protocols of Lund et al. [39] and Shamir [43].

Theorem 4.1. *Suppose h and v are positive integers with $hv \geq n$. Then, for integers $k \geq 1$, there exists an online $(k^2 h \log m, kv \log m)$-scheme for computing F_k exactly.*

Proof. Let A be the input stream. We map the n-vector $\mathbf{f}(A)$ into an $h \times v$ matrix $(f(x,y))_{x\in[h],y\in[v]}$, using any canonical bijection between $[n]$ and $[h] \times [v]$. Pick a prime $q \geq \max\{m^k, 3kh\}$; since $m \geq n$, this can be done while ensuring that $\log q = O(k \log m)$. We shall work in the field $\mathbb{F}_q$, which is safe because q exceeds the maximum possible value of $F_k(A)$. Let $\tilde{f}(X,Y) \in \mathbb{F}_q[X,Y]$ be the unique polynomial satisfying $\deg_X(\tilde{f}) = h-1$, $\deg_Y(\tilde{f}) = v-1$ and $\tilde{f}(x,y) = f(x,y)$ for all $(x,y) \in [h] \times [v]$. The verifier picks a random $r \in \mathbb{F}_q$. As the stream is read, the verifier maintains a sketch consisting of the v quantities $\tilde{f}(r,1), \ldots, \tilde{f}(r,v)$. Clearly, this sketch fits in $O(v \log q)$ bits of storage.

At the end of the stream, the helper provides a polynomial $s'(X) \in \mathbb{F}_q[X]$ that is claimed to be equal to

$$s(X) \;:=\; \sum_{y\in[v]} \tilde{f}(X,y)^k, \tag{1}$$

which has degree at most $k(h-1)$, thus using $O(kh \log q)$ bits of annotation. The verifier evaluates $s'(r)$ from the supplied annotation and computes $s(r) = \sum_{y\in[v]} \tilde{f}(r,y)^k$ from his sketch, checks that $s'(r) = s(r)$ and outputs $\perp$ if not. If the check passes, the verifier outputs $\sum_{x\in[h]} s'(x)$ as the final answer. Clearly, this answer is correct if the annotation was honest. Further, the verifier is fooled only if $s' \neq s$, but $s'(r) = s(r)$; the probability of this is at most $k(h-1)/q \leq \frac{1}{3}$, by choice of q.

It remains to show that the sketch can be computed incrementally in $O(v \log q)$ space. To maintain each $\tilde{f}(r,y)$ for $y \in [v]$, note that upon reading a new token $i \in [n]$ that maps to $(a,b) \in [h] \times [v]$, the necessary update is of the form $\tilde{f}(r,y) \leftarrow \tilde{f}(r,y) + p_{a,b}(r,y)$, where $p_{a,b}$ is the Lagrange polynomial

$$p_{a,b}(X,Y) \;:=\; \prod_{i\in[h]\setminus\{a\}} (X-i)(a-i)^{-1} \cdot \prod_{j\in[v]\setminus\{b\}} (Y-j)(b-j)^{-1}.$$

Since $p_{a,b}(r,y) = 0$ for any $y \in [v] \setminus \{b\}$, the verifier need only update the single value $\tilde{f}(r,b)$, by adding $p_{a,b}(r,b)$, upon reading this token. Using a table of $O(v)$ appropriate precomputed values, this update can be computed quickly. For $h = v = \sqrt{n}$, this takes a constant number of arithmetic operations per update without affecting the asymptotic space cost. □

Numerous problems such as computing Hamming distances and inner products, and approximating F_2 and F_∞, can be solved using F_k as a primitive or using related techniques. We proceed to outline the relevant schemes and the results they provide.

Approximate F_2. We can approximate F_2 up to a $(1+\varepsilon)$ factor from an integer linear sketch of size $O(1/\varepsilon^2)$ (see Section 2.3 for a discussion of sketches). In particular, if $\mathrm{CS}_w(A)$ denotes a length-w Count-Sketch vector of the stream A built using 4-wise independent hash functions, then $F_2(\mathrm{CS}_w(A))$ estimates $F_2(A)$ with relative error $\varepsilon = w^{-1/2}$ with constant probability [44]. Thus, if the verifier and helper have access to a source of public randomness to define the hash functions used by the sketch (or we extend the model to allow the verifier to send the description of the randomly chosen hash functions to the helper at

the start of the protocol), the above F_2 scheme yields an online $(\varepsilon^{-2\alpha}\log m, \varepsilon^{2\alpha-2}\log m)$-scheme for any $\alpha \in [0,1]$. This follows from the combination of the algebrization approach with the observation that the verifier can track linear updates to their sketch efficiently.

Approximate F_∞. Recall that $F_\infty = \max_{j\in[n]} f_j$ and note that $F_\infty^t \le F_t \le nF_\infty^t$. Hence, if $t = \log n/\log(1+\varepsilon)$, then $(F_t)^{1/t}$ is at most a factor $1+\varepsilon$ from F_∞. This yields an online $((\frac{1}{\varepsilon}\log n)^2 h \log m,\ (\frac{1}{\varepsilon}\log n) v \log m)$-scheme for approximating F_∞ for any h, v such that $hv \ge n$. We make use of this scheme in Section 4.4.

Inner Product and Hamming Distance. Consider a stream consisting of a string $\mathbf{x} \in \{0,1\}^N$ followed by a string $\mathbf{y} \in \{0,1\}^N$. Exact computation of F_2 implies online schemes for certain functions of $\mathbf{x}$ and $\mathbf{y}$. For example, the inner product $\mathbf{x}\cdot\mathbf{y}$ is $(F_2(\mathbf{x}+\mathbf{y}) - F_2(\mathbf{x}) - F_2(\mathbf{y}))/2$ and the Hamming distance between $\mathbf{x}$ and $\mathbf{y}$ is $|\{i : x_i = 1\}| + |\{i : y_i = 1\}| - 2\mathbf{x}\cdot\mathbf{y}$. Hence we get an online $(N^\alpha \log N, N^{1-\alpha}\log N)$-scheme for each of these functions, for every $\alpha \in [0,1]$. Alternately, the approach in the proof of Theorem 4.1 can be used to more directly generate schemes for these problems with the same bounds. For example, in the case of inner product, the verifier maintains $\tilde{f}(r,1)\ldots\tilde{f}(r,v)$ and $\tilde{g}(r,1)\ldots\tilde{g}(r,v)$, where $\tilde{f}$ and $\tilde{g}$ are polynomial extensions of $\mathbf{x}$ and $\mathbf{y}$, as above. The verifier then checks that these are consistent with a helper-supplied polynomial $s'(X)$, which is claimed to be $\sum_{y\in[v]} \tilde{f}(X,y)\tilde{g}(X,y)$, by evaluation at $X = r$. The analysis follows the same lines as above.

4.2 Lower Bounds on Frequency Moments

We now present lower bounds on the tradeoffs possible for the exact and approximate computation of the nontrivial frequency moments F_k. The first part of the theorem below shows that the tradeoff given by Theorem 4.1 is nearly tight.

Theorem 4.2. *Suppose $k \ge 0$ and $k \ne 1$. Let $\mathcal{A}$ be an (h,v)-scheme (online or prescient) for computing F_k.*

(1) If $\mathcal{A}$ computes F_k exactly, then it requires $hv = \Omega(n)$.

(2) If $\mathcal{A}$ approximates F_k up to a constant factor, then it requires $hv = \Omega(n^{1-5/k})$.

Proof. Both results follow from lower bounds on the MA complexity of $\textsc{disj}_{n,t} : \{0,1\}^{nt} \to \{0,1\}$, the t-party set disjointness problem, which is defined as follows. The input is a $t \times n$ Boolean matrix, with Player i holding the ith row, for $i \in [t]$. We call an input $\mathbf{x} = (x_{ij})_{i\in[t], j\in[n]}$ *valid* if every column of $\mathbf{x}$ has weight either 0 or 1 or t, and at most one column has weight t. The desired output is

$$\textsc{disj}_{n,t}(\mathbf{x}) \ := \ \neg \vee_{j=1}^{n} \wedge_{i=1}^{t} x_{ij}\,,$$

i.e., 1 iff the subsets of $[n]$ represented by the rows of $\mathbf{x}$ are disjoint. Note that $\textsc{disj}_{n,t}$ is naturally related to frequency moments: for any valid input $\mathbf{x}$, $F_k(S) \ge t^k$ if $\textsc{disj}_{n,t}(\mathbf{x}) = 0$ and $F_k(S) \le n$ if $\textsc{disj}_{n,t}(\mathbf{x}) = 1$ where S is the multiset $\{j : x_{ij} = 1\}$. Thus, reductions from $\textsc{disj}_{n,2}$ and $\textsc{disj}_{n,O(n^{1/k})}$ establish the first and second parts of the theorem, respectively, in a straightforward manner.

To complete the proof, we need a lower bound for $\textsc{disj}_{n,t}$ itself. This is given in the next theorem, which generalizes a result by Klauck [36] and also resolves a question of Feigenbaum et al. [27]. □

Theorem 4.3. *Let $\mathcal{P}$ be an ε-error MA protocol for $\textsc{disj}_{n,t}$, where $\varepsilon \le 1/3$. Then $\mathrm{hcost}(\mathcal{P}) \cdot \mathrm{vcost}(\mathcal{P}) = \Omega(n/t^4)$. In particular, $\mathrm{MA}(\textsc{disj}_{n,t}) = \Omega(\sqrt{n}/t^2)$.*

Proof. A rectangle is defined as a subset of inputs of the form $\mathcal{X}_1 \times \cdots \times \mathcal{X}_t$, where each $\mathcal{X}_i \subseteq \{0,1\}^n$ is a subset of the set of all possible inputs for Player i. A basic fact about deterministic communication protocols is that the inverse image of any transcript of such a protocol must be a rectangle; this is usually called the *rectangle property*. Let $A = \textsc{disj}_{n,t}^{-1}(1)$ and $B = \textsc{disj}_{n,t}^{-1}(0)$. The following lemma was proved by Alon, Matias and Szegedy [4], generalizing a result due to Razborov [42].

Lemma 4.4 (Lemma 3.4 of [4])**.** *There exists a distribution μ over valid inputs such that*

(1) $\mu(A) = \mu(B) = 1/2$, and

(2) every rectangle T satisfies $\mu(T \cap B) \geq (2e)^{-1}\mu(T \cap A) - t2^{-n/2t^4}$. □

Returning to our theorem, assume $t = \omega(n^{1/4})$ since otherwise the bound is trivial. Put $h = \text{hcost}(\mathcal{P})$ and $v = \text{vcost}(\mathcal{P})$. An input $\mathbf{x} \in A$ is said to be *covered* by a message $\mathfrak{h}$ from Merlin if $\Pr_R[\text{out}(\mathcal{P}; \mathbf{x}, R, \mathfrak{h}) = 0] \leq \varepsilon$. By correctness, every such input must be covered, so there exists a help message $\mathfrak{h}^*$ that covers every input in a set $G \subseteq A$, with $\mu(G) \geq 2^{-h}\mu(A) = 2^{-h-1}$. Fix Merlin's message in $\mathcal{P}$ to $\mathfrak{h}^*$ and amplify the correctness of the resulting randomized Merlin-free protocol by repeating it $O(h)$ times and taking the majority of the outputs. This gives us a randomized protocol $\mathcal{P}'$ for $\textsc{disj}_{n,t}$ with communication cost $c = O(hv)$ whose error, on every input in $G \cup B$, is at most 2^{-2h}.

Let μ' denote the distribution μ conditioned on $G \cup B$. Note that, by condition (1) of Lemma 4.4,

$$\forall \mathbf{x} \in \{0,1\}^{nt} : \quad \text{either } \mu'(\mathbf{x}) = 0 \text{ or } \mu(\mathbf{x}) \leq \mu'(\mathbf{x}) \leq 2\mu(\mathbf{x}). \tag{2}$$

By fixing the random coins of $\mathcal{P}'$ we can obtain a deterministic protocol $\mathcal{Q}$, for $\textsc{disj}_{n,t}$, that communicates c bits and satisfies $\text{err}_{\mu'}(\mathcal{Q}) \leq 2^{-2h}$. By the rectangle property, there exist disjoint rectangles $T_1, T_2, \ldots, T_{2^c}$ such that $\text{out}(\mathcal{Q}; \mathbf{x}) = 1$ iff $\mathbf{x} \in \bigcup_{i=1}^{2^c} T_i$. Therefore

$$\sum_{i=1}^{2^c} \mu'(T_i \cap B) \;\leq\; 2^{-2h}, \quad \text{and} \tag{3}$$

$$\mu'\left(A \setminus \bigcup_{i=1}^{2^c} T_i\right) \;\leq\; 2^{-2h}. \tag{4}$$

By (2), we have $\mu'(A) = \mu'(G) \geq \mu(G) \geq 2^{-h-1}$. Using (2), and a rearrangement of (4):

$$\sum_{i=1}^{2^c} \mu(T_i \cap A) \;\geq\; \frac{1}{2}\sum_{i=1}^{2^c} \mu'(T_i \cap A) \;\geq\; \frac{1}{2}\left(\mu'(A) - 2^{-2h}\right) \;\geq\; 2^{-h-3}.$$

Suppose $c \leq n/5t^4$ and n is large enough. Applying condition (2) of Lemma 4.4 to each term in the leftmost sum above, we get

$$\sum_{i=1}^{2^c} \mu(T_i \cap B) \;\geq\; \frac{2^{-h-3}}{2e} - 2^c t \cdot 2^{-n/2t^4} \;\geq\; 2^{-h-6}.$$

However, by (2) and (3), we have $\sum_{i=1}^{2^c} \mu(T_i \cap B) \leq 2^{-2h}$, a contradiction. Hence $hv = \Omega(c) = \Omega(n/t^4)$. □

4.3 Frequency-Based Functions

It is natural to ask whether the F_k algorithm of Theorem 4.1 generalizes to more complicated functions. We demonstrate that this is indeed the case by presenting non-trivial algorithms for the class of all *frequency based functions*. A frequency based function is any function G on frequency vectors $\mathbf{f} = (f_1, \ldots, f_n)$ of the form $G(\mathbf{f}) = \sum_{j \in [n]} g(f_j)$ for some $g : \mathbb{Z}_+ \to \mathbb{Z}_+$. We assume $g(x) \leq n^c$ for some constant c, so that each value in the range of g and G can be represented using $O(\log n)$ bits. If there are constants C_1 and C_2 such that $g(x) = C_1$ for all $x \geq C_2$, then we say that G is *based on low frequencies*.[1]

Frequency-based functions have a number of important special cases, including frequency moments, F_0 (the number of distinct items in the stream), and point and range queries on the frequency distribution, and can also be used to compute F_∞, the highest frequency in the frequency vector. These functions occupy an important place in the streaming world: Alon, Matias, and Szgedy asked for a precise characterization of which frequency-based functions can be approximated efficiently in the standard streaming model in their seminal paper [4]. Braverman and Ostrovsky [9] recently gave a zero-one law for approximating monotonically increasing functions of frequencies that are zero at the origin. This can be contrasted with our result that, in the annotation model, *all* frequency-based functions have non-trivial exact schemes. We first present a natural generalization of the online scheme for F_k, which we call the polynomial-agreement protocol. This protocol was first presented by Cormode, Mitzenmacher, and Thaler in [18]; we present the details for completeness.

Polynomial-Agreement Protocol. Let A be the input stream. We wish to compute $G(\mathbf{f}(A))$, where $G(\mathbf{f}) = \sum_{j \in [n]} g(f_j)$. As in the F_k algorithm, we shall work in the field $\mathbb{F}_q$ for a sufficiently large prime q, and we map the n-vector $\mathbf{f}(A)$ into an $h \times v$ matrix $(f(x,y))_{x \in [h], y \in [v]}$, where h and v are adjustable parameters. As before, we let $\tilde{f}(X,Y) \in \mathbb{F}_q[X,Y]$ be the unique polynomial satisfying $\deg_X(\tilde{f}) = h-1$, $\deg_Y(\tilde{f}) = v-1$ and $\tilde{f}(x,y) = f(x,y)$ for all $(x,y) \in [h] \times [v]$. The verifier picks a random $r \in \mathbb{F}_q$, and maintains a sketch consisting of the v quantities $\tilde{f}(r,1), \ldots, \tilde{f}(r,v)$ as the stream is read.

Now the goal is to compute $\sum_{x,y \in [h] \times [v]} g(\tilde{f}(x,y))$. The *polynomial-agreement protocol* generalizes the F_k protocol, and has the helper send a polynomial to the verifier claimed to be

$$s_1(X) \;:=\; \sum_{y \in [v]} \tilde{g} \circ \tilde{f}(X,y)\,, \tag{5}$$

where $\tilde{g}$ is defined through interpolation as the unique degree-m polynomial that agrees with g on inputs in the set $\{0,1,\ldots,m\}$, this being the set of possible values for each entry of $\mathbf{f}(A)$. Then the verifier can compute $G(\mathbf{f}(A)) = \sum_{x \in [h]} s_1(x)$. To keep the helper honest, the verifier checks that $s_1(r) = \sum_{y \in [v]} \tilde{g}(\tilde{f}(r,y))$ by computing the sum from his sketch.

One may compare Eq. (5) with the earlier Eq. (1), and observe that setting $g(x) = x^k$ indeed yields the F_k scheme from Section 4.1.

Theorem 4.5. *Suppose $m = \Theta(n)$. Let G be any frequency-based function. Then G has a prescient $(n^{2/3}\log n, n^{2/3}\log n)$-scheme and an online $(n^{3/4}\log n, n^{3/4}\log n)$-scheme. Additionally, if G is based on low-frequencies, then G has an online $(n^{2/3}\log n, n^{2/3}\log n)$-scheme.*

Proof. We first describe the prescient scheme. It is natural to attempt to directly apply the polynomial-agreement protocol to the given function g. However, this does not yield a useful result. The problem with

[1] In full generality, we can obtain improved schemes for functions where $C_2 = o(n^{1/12})$.

this approach is that $\tilde{g} \circ \tilde{f}$ has degree $m(h-1)$, and therefore $s_1(X)$, as defined in (5), requires up to $m(h-1)$ words to represent—it would be more efficient for the helper to just repeat the stream in sorted order!

The solution is to reduce the degree of $\tilde{g}$ by removing the *heavy hitters* from A with the aid of the helper. That is, we run a prescient heavy hitters scheme to determine $H := \sum_{j \in S} g(f_j) - |S| g(0)$, where $S := \{j : f_j \geq n^\beta\}$ and $\beta < 1$ is a parameter we will fix later. Though one could use Theorem 3.3 for a heavy hitters scheme, to obtain tighter bounds we use a more efficient scheme presented later in Theorem 6.1. Note that this requires communication $O((m/n^\beta)\log n) = O(n^{1-\beta}\log n)$ since $m = \Theta(n)$ by assumption. Intuitively, H represents the contribution of the heavy hitters to the frequency-based function, and the verifier then "removes" these items from the stream by setting $f_j = 0$ for all $j \in S$. This ensures that the removed items do not contribute to the sum $\sum_{j \in [n]} g(f_j)$. The verifier and helper then run the polynomial-agreement protocol on the *modified* frequency vector, and the final result is given by $H + \sum_{j \in [n]} g(f_j)$. From now on, let $\mathbf{f}$ denote this modified vector.

When running the polynomial-agreement protocol, we exploit the fact that each entry of $\mathbf{f}$ lies in $\{0, 1, \ldots, n^\beta\}$. This lets us use a degree-n^β polynomial $\tilde{g}$ in (5). As a result, we have $\deg(s_1) \leq n^\beta(h-1)$, and so the helper requires only $O(hn^\beta \log n)$ bits to describe $s_1(X)$. For the $\frac{1}{3}$-error guarantee, the prime q need only be as large as $3n^\beta(h-1) = \text{poly}(n)$. All other details remain unchanged, and are in line with the proof of Theorem 4.1.

It remains to show that we can set the parameters h, v, and β of the above protocol to achieve $\text{hcost} = \text{vcost} = O(n^{2/3}\log n)$. The help cost is $O(n^{1-\beta}\log n)$ bits for the heavy hitters scheme plus $O(hn^\beta \log n)$ bits for the (modified) polynomial-agreement protocol. The respective verification costs are $O(n^{1-\beta}\log n)$ and $O(v \log n)$. Setting $\beta = \frac{1}{3}$, $h = n^{1/3}$, and $v = n^{2/3}$ achieves the desired costs.

A subtlety is that the verifier needs to compute the values $g(f_j)$ for all $j \in S$ in order to compute the contribution, H, of the heavy hitters. The verifier also needs to compute the values $g(i)$ for $i \in [n^\beta]$ in order to evaluate $s_1(r) = \sum_{y \in [v]} \tilde{g}(\tilde{f}(r,y))$, because the polynomial $\tilde{g}$ is defined in terms of these values. Indeed, $\tilde{g}(x) = \sum_{i \in [n^\beta]} g(i)\chi_i(x)$, where χ_i is the unique polynomial of degree at most n^β such that $\chi_i(i) = 1$ and $\chi_i(x) = 0$ for all $x \in \{0, \ldots, n^\beta\} \setminus \{i\}$. Thus, to give a space-bounded verifier, we must carefully account for the cost of storing g. However, for most natural functions of interest, g has a succinct implicit description; this is indeed the case for important examples such as F_0, F_∞, and point and range queries on the frequency distribution that are described subsequently.

In order to achieve an online $(n^{3/4}\log n, n^{3/4}\log n)$-scheme for G, observe that the only place where the above scheme used prescience was to identify heavy hitters. So we simply substitute the online heavy hitters scheme of Theorem 6.1, with parameter $\alpha \in [0,1]$, in place of the prescient version. In this case, the help cost is $O(n^{1-\beta}n^\alpha \log n)$ bits for the heavy hitters scheme and $O(hn^\beta \log n)$ bits for the polynomial-agreement protocol. The respective verification costs are $O(n^{1-\alpha}\log n)$ and $O(v \log n)$. Balancing these costs by setting $\beta = \frac{1}{2}$, $\alpha = \frac{1}{4}$, $h = n^{1/4}$, and $v = n^{3/4}$ gives the desired overall costs.

Finally, we describe how to achieve an online $(n^{2/3}\log n, n^{2/3}\log n)$-scheme if G is based on low-frequencies. Then, as described above, there are constants C_1 and C_2 such that $g(x) = C_1$ for all $x \geq C_2$. This obviates the need for a heavy hitters scheme entirely: while observing the stream, the verifier keeps a buffer of the $n^{2/3}$ most recent items observed, and "collapses down" any items appearing more than C_2 times in the buffer to an instance of the item that occurs exactly C_2 times. It is easy to see that G is the same for the collapsed stream as for the original stream, since G is based on low frequencies. As a result of the collapsing, no item in the filtered stream has frequency higher than $O(n^{1/3})$. Therefore we can obtain the desired bounds using a method similar to the polynomial-agreement protocol. □

Applications of Polynomial-Agreement. Theorem 4.5 provides annotation schemes for the problems described below.

- We can compute F_0, the number of items with non-zero count. This follows by observing that F_0 is equivalent to computing $\sum_{i\in[u]} g(f_i)$ for the function g given by $g(0)=0$ and $g(x)=1$ for $x>0$. Since F_0 is based on low frequencies, we achieve an online $(n^{2/3}\log n, n^{2/3}\log n)$-scheme.
- More generally, we can compute functions on the inverse distribution, i.e., queries of the form "How many items occur exactly k times in the stream?" We do this by setting $g(k)=1$ and $g(x)=0$ for $x\neq k$; here we think of k as being fixed. One can build on this to compute, e.g., the number of items which occurred between k and k' times, the median of this distribution, etc. If k is a constant, as in the case of *rarity* (where $k=1$) [22] we achieve an online $(n^{2/3}\log n, n^{2/3}\log n)$-scheme. Otherwise, we achieve a prescient $(n^{2/3}\log n, n^{2/3}\log n)$-scheme and an online $(n^{3/4}\log n, n^{3/4}\log n)$-scheme.
- We obtain a protocol for $F_\infty = \max_{j\in[n]} f_j$, with a little more work. The helper first claims a lower bound ℓ on F_∞ by providing the index of an item with frequency F_∞, which the verifier checks by running the generalized INDEX protocol from Section 3.1 (see Remark 2 after Theorem 3.1). Then the verifer runs the above protocol with $g(x)=0$ for $x\leq\ell$ and $g(x)=1$ for $i>\ell$; if $\sum_{j\in[n]} g(f_j)=0$, then the verifier is convinced that no item has frequency higher than ℓ, and concludes that $F_\infty=\ell$. We therefore achieve a prescient $(n^{2/3}\log n, n^{2/3}\log n)$-scheme and an online $(n^{3/4}\log n, n^{3/4}\log n)$-scheme for F_∞ (or an online $(n^{2/3}\log n, n^{2/3}\log n)$-scheme in the case that F_∞ is at most a constant).

4.4 Frequency-Based Functions for Skewed Streams

In practice, the frequency distributions of data streams are often skewed, in the sense that a small number of frequent items make up a large portion of the stream. We observe that, if the stream is sufficiently skewed, so that there are few heavy hitters, we can achieve more efficient schemes for frequency-based functions. To see this, notice that in the scheme of Theorem 4.5, the verifier, after learning the heavy hitters from the helper, only needs to know an *approximate* upper bound on $F_\infty(A')$, where A' is the stream obtained from the input stream A by deleting all the heavy hitters. That is, the helper only needs to convince the verifier that he has presented "enough" of the true heavy hitters (and their exact frequencies) so that $F_\infty(A')\leq b$ for some upper bound $b=\Theta(n^\beta)$—then we may define $\tilde{g}$ to agree with g on $[b]$, so that the degree of $\tilde{g}$ remains $O(n^\beta)$.

Observe that if there are not many heavy items, the helper can send a list L of heavy hitters and their frequencies (proving the frequencies are truthful via L parallel INDEX protocols) and then appending a proof of an approximate upper bound (within factor $1+\varepsilon$) as per Section 4.1 on the quantity $F_\infty(A')$.

It suffices to let ε be any positive constant in order to achieve $b=O(n^\beta)$. When there are fewer than ℓ items with frequency greater than n^β, the INDEX queries, if they are online, require annotation $O(\ell h\log n)$ and space $O(v\log n)$ for the verifer, while the approximate F_∞ scheme requires annotation $O(h\log^3 n)$ and space $O(v\log^2 n)$. In what follows, we will choose ℓ to be polynomial in n, so we will obtain an $(\ell h\log n, v\log^2 n)$ scheme for identifying the set of heavy hitters and an upper bound u on $F_\infty(A')$.

For concreteness, we will analyze the costs of our improved scheme under the assumption that the frequencies of items in the stream follow a Zipfian distribution, so that the ith largest frequency is (at most) mi^{-z} for parameter z. Setting this equal to n^β and rearranging, we obtain that there are at most $(m/n^\beta)^{1/z}$ heavy hitters to identify.

Therefore, if $m=\Theta(n)$, we can reduce the cost of the heavy hitters sub-protocol within the scheme of Theorem 4.5 to $(n^{(1-\beta)/z}h\,\mathrm{poly}\log n, v\,\mathrm{poly}\log n)$. Adding in the annotation cost of sending the polynomial

$\tilde{g} \circ \tilde{f}$, and the space cost of storing the verifier's sketch, the entire scheme therefore requires $\tilde{O}(n^{(1-\beta)/z}h + hn^{\beta})$ annotation and $\tilde{O}(v)$ space, where the $\tilde{O}$ notation hides factors polylogarithmic in n. Balancing exponents by setting $\beta = 1/(z+1)$, $h = n^{\frac{1}{2}+\frac{1}{2(z+1)}}$, and $v = n/h$, we obtain an $(n^{\frac{1}{2}+\frac{1}{2(z+1)}} \operatorname{poly}\log n, n^{\frac{1}{2}+\frac{1}{2(z+1)}} \operatorname{poly}\log n)$ scheme.

For example, if $z = 2$, we obtain an online $(n^{2/3} \operatorname{poly}\log n, n^{2/3} \operatorname{poly}\log n)$-scheme, which essentially matches the cost of our online scheme for functions based on low-frequencies, but applies to any frequency-based function. If $z = 3$, we obtain an online $(n^{5/8} \operatorname{poly}\log n, n^{5/8} \operatorname{poly}\log n)$-scheme.

Finally, we present a more efficient prescient scheme. If we use prescient INDEX protocols rather than online ones, our heavy hitters scheme only requires annotation $\tilde{O}(l + h_1)$ and space $\tilde{O}(l + v_1)$, provided $h_1 v_1 \geq n$. Hence, the entire scheme has annotation cost $\tilde{O}(n^{(1-\beta)/s} + h_1 + h_2 n^{\beta})$ and space cost $\tilde{O}(n^{(1-\beta)/s} + v_1 + v_2)$, where $h_1 v_1 = h_2 v_2 = n$. Assume $1 < s \leq 2$. Then setting $\beta = \frac{2-s}{2+s}$, $h_1 = v_1 = n^{1/2}$, $h_2 = n^{z/(2+z)}$, and $v_2 = n^{2/(2+z)}$, we obtain an $(n^{2/(2+z)} \operatorname{poly}\log n, n^{2/(2+z)} \operatorname{poly}\log n)$ scheme. For example, if $z = 2$, we obtain a prescient $(n^{1/2} \operatorname{poly}\log n, n^{1/2} \operatorname{poly}\log n)$-scheme. For $z > 2$, schemes with the same costs follow by setting $\beta = 0$, $h_1 = h_2 = v_1 = v_2 = n^{1/2}$.

5 Set and Multiset Inclusion

Building on some of the results and techniques in Section 4, we now address a family of abstract problems that involve a helper proving a subset (inclusion) relation to a streaming verifier. Both sets and multisets are of interest. For example, we may need to prove that $A \subseteq B$ for two sets A and B, or we may need to prove that a set A is exactly the support set of a multiset B. These abstract problems turn out to be common subproblems arising in a number of applications that we shall consider later (see, e.g., Theorems 7.5, 7.6, and 7.7).

Throughout this section, the *size* of a multiset is the number of elements in it, counting multiplicities. A fingerprint of a multiset is a basic fingerprint, as in Definition 1, of its characteristic (frequency) vector.

Lemma 5.1. *Let $A \subseteq [n]$ be a set and $B \subseteq [n]$ a multiset of size t. Let B' be the set formed by removing all duplicate elements from B. Then, given a stream which begins with the elements of A followed by the elements of B, there is a prescient $(t \log n, \log n)$-scheme that establishes whether $B' = A$.*

Proof. As the elements of A are observed in the stream, the helper annotates each $a \in A$ with the multiplicity, f_a, of a in B. Once A has been observed, the helper then lists each element b in the set difference $B' \setminus A$, along with the corresponding multiplicity f_b in B. Obviously there are no such elements iff $B' = A$. From the provided information, the verifier constructs a fingerprint of the multiset in which each $a \in A \cup B'$ appears with multiplicity f_a.

Then, while observing the elements of the multiset B, the verifier incrementally constructs a fingerprint of B, as in Lemma 2.1. The verifier accepts iff the two fingerprints match. □

In the remainder of this section, we give three schemes achieving tradeoffs between hcost and vcost for (multi)-set inclusion, in order of generality. First, we give an essentially optimal *online* $(h \log n, v \log n)$-scheme, for any h and v with $hv \geq n$, for the special case when B is a set rather than a multiset.

Theorem 5.2. *Let $X, Y \subseteq [n]$ be sets. Then given a stream with elements of X and Y arbitrarily interleaved, there is an online $(h \log n, v \log n)$-scheme for determining whether $X \subseteq Y$ for any h and v such that $hv \geq n$. Moreover, any online (h, v)-scheme requires $hv = \Omega(n)$.*

Proof. Let $x, y \in \{0,1\}^n$ be the characteristic vectors of X and Y respectively. Then $X \subseteq Y$ if and only if $F_2(y-x) = |Y| - |X|$. Consequently, the helper can run the F_2 scheme of Theorem 4.1 on the vector $y-x$ to determine if the above equality holds.

The lower bound follows from a straightforward reduction from INDEX. Take $N = n$. Given the string $x \in \{0,1\}^n$, Alice transforms it into the stream over $[n]$ representing the set $Y = \{j : x_j = 1\}$. Given the index $i \in [n]$, Bob transforms it into a stream representing the singleton set $X = \{i\}$. Then $x_i = 1$ if and only if $X \subseteq Y$. □

We now show how to use the result for frequency-based functions to handle duplicated items; in this case X and Y are multisets rather than sets. The next theorem lets us efficiently handle a small number of duplicates.

Theorem 5.3. *Let $X, Y \subseteq [n]$ be multisets. Assume k is a known upper bound on the maximum frequency of any element in X or in Y. Then given a stream with elements of X and Y arbitrarily interleaved, there is a online $(kh\log n, v\log n)$-scheme for determining whether $X \subseteq Y$, for any h and v with $hv \geq n$.*

Proof. Let x, y be the characteristic vectors of X and Y respectively. Then $X \subseteq Y$ if and only if $y_i - x_i \geq 0$ for all i. The bound on the maximum frequency implies that $-k \leq y_i - x_i \leq k$ for all $1 \leq i \leq n$. Let $\tilde{g}$ be defined through interpolation as the polynomial of degree $2k$ over the finite field $\mathbb{F}_p$ such that $\tilde{g}(x) = 0$ for $x \in \{0, 1, \ldots, k\}$, and $\tilde{g}(x) = 1$ for $x \in \{-k, -k+1, \ldots, -1\}$. Then $\sum_i \tilde{g}(y_i - x_i) = 0$ if and only if $X \subseteq Y$; intuitively, $\tilde{g}$ acts as an indicator function for the set of possible negative entries in the vector $y-x$. Applying the polynomial-agreement protocol defined in the proof of Theorem 4.5 under this definition of $\tilde{g}$, we obtain a $(kh\log n, v\log n)$-scheme for checking $X \subseteq Y$ whenever $hv \geq n$. □

Finally, we give an online $(n^{3/4}\log n, n^{3/4}\log n)$-scheme for the general multiset inclusion problem, as long as $t = O(n)$.

Theorem 5.4. *Let $X, Y \subseteq [n]$ be multisets of size at most t. Then given a stream with elements of X and Y arbitrarily interleaved, there is there is an online $(n^{3/4}\log n, n^{3/4}\log n)$-scheme for determining whether $X \subseteq Y$ assuming $t = O(n)$.*

Proof. Let x, y be the characteristic vectors of X and Y respectively. It holds that $X \subseteq Y$ if and only if $y_i - x_i \geq 0$ for all i. Define $g : \{-t, -t+1, \ldots, 0, 1, \ldots, t\} \to \{0,1\}$ by $g(x) = 0$ for $x \in \{0, \ldots, t\}$ and $g(x) = 1$ for $x \in \{-t, -t+1, \ldots, -1\}$. The theorem holds by applying the protocol of Theorem 4.5 to $G(\mathbf{f})$, where $\mathbf{f}$ is the vector $y-x$ and G is the frequency-based function defined by g. (As stated, the protocol of Theorem 4.5 applies only to $g : \mathbb{Z}_+ \to \mathbb{Z}_+$, but it applies without modification to any function g defined on a suitably small domain, such as ours). □

5.1 Application: Convex Hull on a 2D Grid

As a first illustration of the value of Theorems 5.2–5.4, consider an instance of the convex hull problem where all input points P fall on the intersection points of a two-dimensional grid defining g possible point locations. Let C be the convex hull of a stream of points. Then, for any $0 \leq \alpha \leq 1$, there exists an online $((|C| + g^{\alpha})\log g, (|C| + g^{1-\alpha})\log g)$-scheme to report the convex hull. The helper provides the claimed hull C', which the verifier can store exactly, and verify that it is indeed convex. Define $c(C)$ as the set of (grid) points contained within a convex shape C, and observe that it is easy to enumerate (but not store) $c(C)$ in space $O(|C|)$. The verifier then must establish that $C' \subseteq P$, and that $P \subseteq c(C')$. Both these subset tests can be verified efficiently using Theorem 5.2. As described, this protocol requires that P should contain

no duplicate points, however in the case where each point in P is duplicated at most a small number of times k, we can instead use the protocol of Theorem 5.3 rather than Theorem 5.2. This yields an online $((|C|+kg^{\alpha})\log g,(|C|+g^{1-\alpha})\log g)$-scheme to report the convex hull. If points are duplicated up to $O(n)$ times, we may instead apply Theorem 5.4 to obtain an online $((|C|+g^{3/4})\log g,(|C|+g^{3/4})\log g)$-scheme.

6 Frequent Items

In this section, we provide further results on finding exact and approximate frequent items. Our new results improve over Theorem 3.3 by logarithmic factors by showing a more compact witness set for the exact case, which leads to improved online schemes for the exact and approximate versions of the problem.

Theorem 6.1. *Let $T=\phi m$. There exists a prescient $(\phi^{-1}\log m,\phi^{-1}\log m)$-scheme and, for every $\alpha\in[0,1]$, an online $(\phi^{-1}n^{\alpha}\log m,\, n^{1-\alpha}\log m)$-scheme for finding $\{j : f_j > T\}$. Any online or prescient (h,v)-scheme for this problem must have $hv=\Omega(n)$.*

Proof. For the upper bound, consider a binary tree $\mathcal{T}$ whose leaves are the elements of the universe $[n]$, as in Theorem 3.3. We will specify a witness set W of size $O(\phi^{-1})$ to identify to identify all leaves j with $f_j > T$; we base W on the concept of *Hierarchical Heavy Hitters* (*HHH*s) [16]. Below, we refer to the set of Hierarchical Heavy Hitters as H.

We define H inductively, beginning with the leaves and working our way to the root. We include a leaf in H if its frequency exceeds T. Let u be a node at distance l from the root (i.e., at level l of $\mathcal{T}$), and assume inductively that we have determined all *HHH*s at levels greater than l. Let $H(u)$ denote the set of descendants of u that have been included in H, and let $L(u)$ denote the set of leaves of the subtree rooted at u. Finally, define $S(u) := L(u) \setminus \left(\cup_{v\in H(u)} L(v)\right)$. Intuitively, $S(u)$ is the set of leaves in $L(u)$ that have not already contributed their frequency to an *HHH* descendant of u. Define the *conditioned count* of u as $g(u) := \sum_{j\in S(u)} f_j$; we include u in H if $g(u) > T$. Observe there are at most ϕ^{-1} items in H since $T=\phi m$: each leaf contributes its frequency to $g(u)$ for exactly one $u\in H$, and therefore $|H|T \le \sum_{u\in H} g(u) \le m$.

We now define our witness set W as all leaves j in H in addition to all nodes u such that u's parent is in H but u is not in H. Observe that each node $u\in W$ is witness to the fact that no leaves $j\in S(u)$ can have $f_j > T$. We also include the root r in W to account for any leaves that are not descendants of any node in H. The sets $S(u)$ for $u\in W$ form a partition of $[n]$. Notice that $|W| = O(\phi^{-1})$ since $|H|\le\phi^{-1}$.

This leads to two schemes for the problem. In the first, prescient scheme, the helper lists all nodes $u\in W$ sorted by the natural order on nodes, and the verifier remembers this information. The verifier may then compute the conditioned count of each $u\in W$ using space $O(|W|\log n) = O(\phi^{-1}\log n)$: each time an item j appears in the stream, the verifier determines the unique $u\in W$ such that $j\in S(u)$ (u is simply the ancestor of j in W farthest from the root), and increments $g(u)$. The verifier checks that $g(j) > T$ for all leaf nodes $j\in W$, and that $g(u)\le T$ for all internal nodes in W and outputs $\perp$ otherwise. Since the sets $S(u)$ partition $[n]$, this latter check ensures that the helper does not omit any leaves j with $f_j > T$.

The second, online scheme is more involved. In the online setting, it is no longer possible for the verifier to track the conditioned count of each node in W while observing the stream. However, it is possible for the verifier to track (fingerprints of) a related quantity for each node v, called the *unconditioned count* of v.

For each node u in $\mathcal{T}$, recall that $L(u)$ denotes the leaves of the subtree rooted at u, and $H(u)$ denotes the descendants of u that are in H. Define the unconditioned count of u as $f(u) = \sum_{j\in L(u)} f_j$. Observe that there is a simple relationship between the conditioned and unconditioned counts of u, namely $g(u) = f(u) - \sum_{v\in H(u)} g(v)$. The verifier may exploit this relationship to force the helper to provide the true conditioned counts for each node $u\in W$.

In more detail, the $2n-1$ nodes in the tree are divided into v groups of h such that $hv \geq 2n$, as in the generalized INDEX protocol (see Remark 2 after Theorem 3.1). While observing the stream, the verifier keeps a basic fingerprint of the vector of unconditioned counts of each group. This is easily accomplished by treating each entry j in the stream as an update to the unconditioned counts of the $\log n$ ancestors of j in $\mathcal{T}$.

After the stream is seen, the helper provides the witness set W, beginning with the leaves in W and working level-by-level towards the root. For each internal node u in W, the helper also presents the (claimed) conditioned count $g_1(u)$ for u, as well as the conditioned count of u's parent, who is claimed to be in H by definition of W.

When processing $(h, g_1(h))$ for any node claimed to be in H, the verifier modifies the basic fingerprints by treating this as a deletion of $g_1(h)$ occurrences of each ancestor of h. More formally, for any node v, let $\mathbf{v}(v)$ denote the vector corresponding to v's group. For each ancestor v of h, the verifier updates

$$\mathrm{BF}_q(r, \mathbf{v}(v)) \leftarrow \mathrm{BF}_q(r, \mathbf{v}(v))\,(r-v)^{-g_1(h)},$$

where $(r-v)^{-g_1(h)}$ denotes the multiplicative inverse of $(r-v)^{g_1(h)}$ in the field $\mathbb{F}_q$. As a result, if each claimed conditioned count $g_1(h)$ is truthful, then when each node $u \in W$ is presented by the helper, the entry corresponding to u in $\mathbf{v}(u)$ is equal to $f(u) - \sum_{v \in H(u)} f(v) = g(u)$. All that remains is to ensure that the $g_1(h)$ values are as claimed.

To this end, the helper is further required to follow each pair $(u, g_1(u))$ with all the entries of the vector $\mathbf{v}(u)$, accounting for all deletions that the verifier has simulated so far. If the helper does not faithfully provide the vector $\mathbf{v}(u)$, a fingerprint of the claimed vector will fail to match the verifier's fingerprint with high probability. Consequently, the helper is forced to provide the true conditioned counts of each node u in W.

Then as in the prescient protocol, the verifier can ensure that for each $u \in W$, the conditioned count of $g(u)$ is below T, indicating that the helper did not omit any leaves $j \in S(u)$ with $f_j > T$.

In total, the verifier requires space $v \log n$ to maintain v fingerprints, and the helper needs to provide $\min\{O(|W|h), n\}$ items and (conditioned) counts, yielding an online $(\min\{n \log m, h\phi^{-1} \log m\}, v \log m)$-scheme. A subtlety here is that the output size can exceed the verifier's memory, so the verifier may output a partial result before returning $\perp$.

We prove the lower bound by an easy reduction from two-party set-disjointness, $\mathrm{DISJ}_{n,2}$. Consider Alice and Bob with respective inputs $x, y \in \{0,1\}^n$. Alice's input x induces a stream A by placing one copy of token j in the stream if $x_j = 1$. Then Bob places one copy of item j in the stream if $y_j = 1$. We may assume Bob knows $|\{j : x_j = 1\}|$, and hence knows the stream length m; if not Alice can tell Bob $|\{j : x_j = 1\}|$ at an additive cost of logarithmically many bits. Now x and y are disjoint if and only if the set $\{j : f_j > 1 = \phi m\}$ for $\phi = 1/m$ is non-empty. Thus, determining the frequent items for $T = 1$ solves two-party set disjointness, proving the bound by Theorem 4.3. □

6.1 Approximate Frequent Items

In many cases, it suffices to find a set of *approximate* frequent items: these include all items with $f_j > \phi m$ and no items with $f_j < (\phi - \varepsilon)m$ for parameters ε, ϕ. Solutions to this problem in the traditional streaming model are often based on "sketch" algorithms, as described in Section 2.3. Since a sketch v is a linear transform of the input, $v = S\mathbf{v}(A)$, a sketch can be fingerprinted: each update multiplies the fingerprint by $\mathrm{BF}_q(r, S\mathbf{e}_i)$. This observation means that the helper can annotate (parts of) $\mathbf{v}$ at the end of the stream, for verification. However, to define an efficient scheme, we also need to show: (1) the verifier can compute

$S\mathbf{e}_i$ in small space, so S must have a compact representation; and (2) the verifier must be able to extract the result from $\mathbf{v}$ in a streaming fashion, in space sublinear in the size of the *sketch*.

We use ideas from verifying exact frequent items to build a scheme for verifying approximate frequent items via sketching.

Theorem 6.2. *For $s > \phi^{-1}$, there exists an online $(s \log n \log m, \log m)$-scheme to verify the approximate frequent items found by Count-Sketch or Count-Min sketches of size s.*

Proof. Our proof proceeds by extending Theorem 3.3 to the case of sketching. The main difference is that exact counts are replaced by estimated counts drawn from the sketch, which requires a little more effort to handle. We consider an expanded set of items that includes the set of tree nodes u in $\mathcal{T}$ and their corresponding unconditioned counts $f(u)$ (recall $f(u)$ is the sum of the frequencies of all leaves in $L(u)$, the subtree rooted at u). The helper and verifier now keep a sketch $\mathbf{v}^k$ for each level k of the tree, to obtain *estimated* unconditioned counts $\hat{f}(u)$ for each node u in the tree. We henceforth assume that $\hat{f}(u) = f(u) \pm \varepsilon m$; when using sketches with $d = O(\log n)$, this holds for each i with probability at least $1 - 1/16n$, and so it holds over *all* $2n$ frequencies with probability at least $7/8$.

As in Theorem 3.3, the witness set W, given threshold T, consists of all leaves j with $\hat{f}_j > T$ in addition to pairs of nodes (u, v) such that u is the child of v, and $\hat{f}(u) \leq T$ but $\hat{f}(v) > T$. Now, there can be at most ϕ^{-1} such nodes v at any level of the binary tree, as the sum of $\hat{f}(v)$ is at most $(1+\varepsilon)m$. This bounds the size of this witness set to $|W'| = O(\phi^{-1} \log n)$ if $\varepsilon < \frac{\phi}{2}$.

The verifier can validate this witness set W over the full set of nodes and their estimated unconditioned counts as follows. By presenting the set of nodes v in W in order of $\min L(v)$, the verifier can ensure that the nodes identified do cover all of $[n]$ as required (and hence that no high frequency items are omitted). If the helper provides for each node $v \in W$ the information about v contained in the sketch, as $(v, \hat{f}_v, \hat{f}_{v,1}, \ldots \hat{f}_{v,d})$ the verifier can check that $\hat{f}_v$ is above or below T as appropriate. The verifier ensures that $\hat{f}_v$ is derived correctly from the d values of $\hat{f}_{v,\ell}$ (using $O(d)$ working space). The verifier also incrementally builds a fingerprint of the set $B = \{(v, \ell, \hat{f}_{v,\ell})\}$. At the end of the annotation, the helper lists the entries of each sketch $\mathbf{v}^k_{\ell,j}$ in order and tags each entry with the set of v's for which it has been used to make an estimate. The verifier builds a fingerprint of the tuples $(v, \ell, c_\ell(v)\mathbf{v}^k_{\ell,b_\ell(v)})$, and checks that it matches the fingerprint of B (this is essentially an instance of the multiset equality protocol in Lemma 5.1). The verifier fingerprints also the (untagged) sketch to check it matches the verifier's fingerprinted sketch built from the input stream.

The total amount of annotation is $O(s \log n)$ sketch entries, from the $\log n$ sketches of size s. The verifier needs to remember d estimated frequencies (to verify their median) and $O(\log n)$ fingerprinted sketches (one for each level). □

We mention that if $\phi \gg \varepsilon$, then the verifier only needs to inspect a small fraction of the sketch entries to verify the frequent items. In this case, one can obtain a tradeoff via the generalized protocol (Section 3.1): write the sketch as an array of $h \times v$ entries, so that $hv \geq s$. The verifier can create v fingerprints each summarizing h entries of the sketch. To verify, the helper modifies the above algorithm to only present those blocks of h entries which include a value that needs to be seen by the verifier. In total, to verify $O(|W'|)$ approximate frequencies requires verifying $O(\phi^{-1} d \log n)$ entries, giving an $(\phi^{-1} h \log m \log^2 n, v \log m)$ online scheme.

Other algorithms find all items j such that $\hat{f}_j \geq \phi F_2^{1/2}$. These can also be adapted to our setting using similar ideas, and verified in logarithmic space with annotation proportional to the sketch size.

7 Graph Problems

In this section we consider computing properties of graphs on n nodes, determined by a stream of m edges [25, 32]. We present tight results for testing connectivity of sparse graphs, determining bipartiteness, determining if a bipartite graph has a perfect matching, and counting triangles. Our bipartite perfect matching result achieves optimal tradeoffs up to logarithmic factors.

7.1 Counting Triangles via Matrix Multiplication

Estimating the number of triangles in a graph has received significant attention because of its relevance to database query optimization—knowing the degree of transitivity of a relation is useful when estimating the cost of evaluation plans for certain relational queries—and investigating structural properties of the web-graph and social graphs [6, 10, 34]. In the absence of annotation, any single-pass algorithm to determine if there is a non-zero number of triangles requires $\Omega(n^2)$ bits of space [6]. In contrast, we show that the exact number of triangles can be verified in logarithmic space, with the help of $O(n^2 \log n)$ bits of annotation. The following theorem, proved using ideas from Bar-Yossef et al. [6] coupled with Theorem 4.3, shows that this amount of annotation is nearly optimal, for a log-space verifier.

Theorem 7.1. *Any (h,v)-scheme for counting triangles must have $hv = \Omega(n^2)$.*

Proof. We show a reduction from $\text{DISJ}_{(n^2/9),2}$. We represent an instance of DISJ as a pair of $(n/3) \times (n/3)$ Boolean matrices X, Y in the natural way. We proceed to construct a graph that has a triangle iff $X_{ij} = Y_{ij} = 1$ for some $i, j \in [n/3]$. The nodes are partitioned into sets U, V, W so that $|U| = |V| = |W| = n/3$. Insert edges $\{(u_i, w_i) : i \in [n/3]\} \cup \{(u_i, v_j) : X_{ij} = 1\} \cup \{(w_i, v_j) : Y_{ij} = 1\}$. There is a triangle (u_i, v_j, w_i) iff $X_{ij} = Y_{ij} = 1$, and there is no other way to form a triangle. The result follows from Theorem 4.3. □

We now outline an online scheme with $\text{vcost} = O(\log n)$ and $\text{hcost} = O(n^2 \log n)$. A major subroutine of our algorithm is the verification of (integer) matrix multiplication in our model. That is, given $n \times n$ matrices A, B and C with integer entries, verify that $AB = C$. Our technique extends the classic result of Frievalds [28] by showing that if the helper presents the results in an appropriate order, the verifier needs only $O(\log n)$ bits to check the claim. Note that this much annotation is necessary if the helper is to provide C in his stream.

Theorem 7.2. *There exists an online $(n^2 \log n, \log n)$-scheme for verifying integer matrix multiplication.*

Proof. Let q be a prime larger than $2nm^2 + 1$, where m is an *a priori* upper bound on the absolute values of all entries of A and B. By the result of Kimbrel and Sinha [35], the verifier can check $AB = C$ by picking r uniformly from $\mathbb{F}_q$ and checking that $A(B\mathbf{r}^\mathsf{T}) = C\mathbf{r}^\mathsf{T}$, in the field $\mathbb{F}_q$, for vector $\mathbf{r} = (r^0, r^1, \ldots, r^{n-1})$. This fails to identify an incorrect product with probability at most n/q. Rather than computing $A(B\mathbf{r}^\mathsf{T})$ and $C\mathbf{r}^\mathsf{T}$ explicitly, the verifier will compare fingerprints of $C\mathbf{r}^\mathsf{T}$ and $AB\mathbf{r}^\mathsf{T}$. These are computed as $\mathbf{s}C\mathbf{r}^\mathsf{T}$ and $\mathbf{s}AB\mathbf{r}^\mathsf{T}$, for a vector $\mathbf{s} = (s^0, s^1, \ldots, s^{n-1})$ where s is picked uniformly from $\mathbb{F}_q$. This fingerprinting fails to distinguish distinct vectors with probability at most n/q.

We observe that (1) $\mathbf{s}C\mathbf{r}^\mathsf{T} = \sum_{i,j} s^i r^j C_{ij}$ can be computed easily whatever order the entries of C are presented in. (2) $\mathbf{s}AB\mathbf{r}^\mathsf{T} = (\mathbf{s}A)(B\mathbf{r}^\mathsf{T})$ is the inner product of two n-dimensional vectors, and that $(\mathbf{s}A)_i = \sum_j s^j A_{ij}$ and $(B\mathbf{r}^\mathsf{T})_i = \sum_j r^j B_{ji}$. Therefore, if the helper presents the ith column of A followed by the ith row of B for each i in turn, the verifier can easily compute $\mathbf{s}AB\mathbf{r}^\mathsf{T}$ in $O(\log q)$ space. Picking $q \geq 6n$ ensures that the verifier is fooled with probability at most $1/3$, and the total space used by the verifier to store r, s and intermediate values is $O(\log n)$. □

With this primitive, arbitrary matrix products $A_\ell A_{\ell-1}\cdots A_2A_1$ are verified with $O(\ell n^2 \log n)$ annotation by verifying $\underline{A}^{(2)} := A_2A_1$, then $\underline{A}^{(3)} := A_3\underline{A}^{(2)}$, etc. Matrix powers A^ℓ are verified with $O(n^2 \log \ell \log n)$ annotation, using repeated squaring. Here, we assume that the entries computed do not grow too large, and so can be represented within $O(\log n)$ bits.

Theorem 7.3. *There is an online $(n^2 \log n, \log n)$-scheme for counting triangles.*

Proof. Denote the graph adjacency matrix by A, with $A_{ii} := 0$. The helper lists A_{vw} and A^2_{vw} for all pairs (v,w) in some canonical order. The verifier computes $\sum_{v,w} A_{vw}A^2_{vw}$ as the number of triangles. The verifier uses fingerprints to check that A matches the original set of edges, and the scheme in Theorem 7.2 to ensure that A^2 is as claimed. □

We also show that it is possible to trade off the computation with the helper in a "smooth" manner. The approach is based on the following observation of Bar-Yossef et al. [6].

From the given stream of edges of a graph, we can create a *derived* stream, of length $m(n-2)$, by replacing each edge (u,v) with the set of triples $\{(u,v,w) : w \neq u,v\}$. The frequency moments of this derived stream can be expressed in terms of the numbers of triples of nodes with exactly zero, one, two and three edges between them. It follows that the number of triangles can be expressed in terms of the frequency moments of this derived stream, as $(F_3 - 3F_2 + 2F_1)/6$. By using the scheme of Theorem 4.1, we obtain the following theorem.

Theorem 7.4. *There is an online $(n^{3\alpha}\log n, n^{3-3\alpha}\log n)$-scheme for counting triangles for each $\alpha \in [0,1]$.*

7.2 Bipartite Perfect Matching

We present two online schemes for determining whether a bipartite graph has a perfect matching. Our first scheme is efficient for sparse graphs, while our second achieves *optimal* tradeoffs between hcost and vcost for dense graphs, up to logarithmic factors. Graph matchings have been considered in the stream model [25, 47] and it can be shown that any single-pass algorithm for determining the exact size of the maximum matching requires $\Omega(n^2)$ space. We show that we can off-load this computation to the helper such that, with only $O(n^{1+\alpha}\log n)$ annotation, the answer can be verified in $O(n^{1-\alpha}\log n)$ space, for each $\alpha \in [0,1]$. This is shown to be best possible by combining a reduction from [25] coupled with Theorem 3.1.

Theorem 7.5. *There exists an online $(m\log n, \log n)$-scheme for bipartite perfect matching, as well as an online $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme for each $\alpha \in [0,1]$. Any online (h,v)-scheme for bipartite perfect matching requires $hv = \Omega(n^2)$.*

Proof. We begin by presenting the $(m\log n, \log n)$-scheme. We consider the general case, where there may be nodes in $[n]$ with no incident edges, which are to be ignored for the matching. If there is a perfect matching M, the annotation lists all edges in M, and the degree of all nodes in $[n]$. Let x be the characteristic vector that has 1 in the vth coordinate if and only if the degree of v is non-zero, and let y be the vector of node frequencies in M. The verifier can use fingerprints to ensure that the claimed degree sequence is correct, and that x matches y.

If the graph does not have a perfect matching, Hall's Theorem provides a witness. Let (L,R) be a bipartition of the graph. Then there exists $L' \subseteq L$ such that $|L'| > |\Gamma(L')|$, where $\Gamma(L')$ is the set of neighbors of L'. The helper lists, for each node, the following information: its degree; whether it is in L or in R; and whether it is in L', $\Gamma(L')$, or neither. Then the helper presents each edge (u,v), along with the same

information on each node. By Lemma 5.1, the verifier can ensure that the sets are consistent, using a constant number of fingerprints. It remains to check that each edge is allowable and that $|L'| > |\Gamma(L')|$.

Our $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme follows the same conceptual outline as the above: if G has a perfect matching, the helper provides the matching, while if G has no perfect matching, the helper demonstrates this via Hall's Theorem. The details follow.

If there is a perfect matching M, the annotation lists all edges in M, followed by a proof that $M \subseteq E$. More specifically, for any $hv \geq n^2$, Theorem 5.2 describes how to obtain an online $(h\log n, v\log n)$-scheme for showing $M \subseteq E$, assuming no duplicate edges. This can be extended to a $(kh\log n, v\log n)$-scheme if edges may be duplicated up to k times by Theorem 5.3. The helper uses this scheme to demonstrate $M \subseteq E$, and the verifier checks that M is a matching by comparing a fingerprint of M to one of the set $\{1,2,\ldots,n\}$.

If the graph does not have a perfect matching, let (L,R) be a bipartition, as before, and let $L' \subseteq L$ be such that $|L'| > |\Gamma(L')|$. We will use the online $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme for integer $n \times n$ matrix-vector multiplication described in [17, Theorem 4]. The verifier must check that (1) L is a bipartition of n; (2) $L' \subseteq L$; and (3) $|L'| > |\Gamma(L')|$. Let $\mathbf{x} \in \{0,1\}^n$ be the indicator vector of L, and let A be the adjacency matrix of G, i.e., $A_{ij} = 1$ if there is an edge between i and j in G and $A_{ij} = 0$ otherwise. Condition (1) is equivalent to $\mathbf{x}^\mathsf{T} A\mathbf{x} = 0$, which can be checked using integer matrix-vector multiplication to verify $A\mathbf{x}$, followed by an inner-product scheme to verify $\mathbf{x}^\mathsf{T} A\mathbf{x}$. Condition (2) can be checked trivially while the helper specifies L by requiring the nodes of L' to be marked. To check (3), notice that $|\Gamma(L')|$ is equal to the number of non-zero entries in the vector $A\mathbf{x}$. This can be computed while the verifier checks (1), and that $|\Gamma(L')| < |L'|$.

The result is an online $(kn^{1+\alpha}\log n, n^{\alpha}\log n)$-scheme for $0 \leq \alpha \leq 1$, where k is an *a priori* upper bound on the number of times each edge may be duplicated. □

7.3 Bipartiteness

The problem of determining if a graph is bipartite was considered in the standard stream model [25,26], and it can be shown that any one-pass algorithm without annotations needs $\Omega(n)$ bits of space. In our model, the helper can convince a verifier with $O(\log n)$ space whether a graph is bipartite, using only $O(m\log n)$ annotation, and we show that this is essentially the best possible for sparse graphs where $m = O(n)$ using a reduction from $\text{DISJ}_{n,2}$ to bipartiteness. We also achieve tradeoffs between hcost and vcost for dense graphs, obtaining an online $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme for each $\alpha \in [0,1]$.

Theorem 7.6. *There exists an online $(m\log n, \log n)$-scheme for determining whether a graph is bipartite, as well as an online $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme for each $\alpha \in [0,1]$. Any (h,v)-scheme (online or prescient) for bipartiteness requires $hv = \Omega(n)$ even when $m = O(n)$.*

Proof. In both the $(m\log n, \log n)$-scheme and the $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme, the helper proves that a graph is *non*-bipartite by providing an odd cycle C. The verifier must check that the number of edges in C is odd, that C is a cycle, and that $C \subseteq E$. The verifier can easily perform the first two checks in logarithmic space. In the $(m\log n, \log n)$-scheme, the verifier checks that $C \subseteq E$ using Lemma 5.1, and in the $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme, the verifier checks that $C \subseteq E$ using Theorem 5.2.

In both schemes, the helper proves that a graph *is* bipartite by specifying all nodes L in the left set of a bipartition. Checking that L is indeed a bipartition of G can be done exactly as in Theorem 7.5.

For the lower bound, we reduce an instance $(x,y) \in \{0,1\}^n \times \{0,1\}^n$ of $\text{DISJ}_{n,2}$ to an instance of bipartiteness on a graph with $O(n)$ edges over nodes $(v_{ij})_{i\in[3],j\in[n]}$. For each $j \in [n]$, create edges (v_{1j}, v_{2j}); if $x_j = 1$, add the edge (v_{1j}, v_{3j}), and if $y_j = 1$, add the edge (v_{2j}, v_{3j}). The resulting graph contains an odd cycle if and only if x and y are not disjoint. □

7.4 Connectivity

The problem of determining if a graph is connected was considered in the standard stream model [25, 32] and the multi-pass W-stream model [24]. In both models, it can be shown that any constant-pass algorithm without annotations needs $\Omega(n)$ bits of space. Similar to bipartiteness, in our model the helper can convince a verifier with $O(\log n)$ space whether a graph is connected, using only $O(m \log n)$ annotation. This is essentially the best possible for sparse graphs where $m = O(n)$ by combining a reduction from [25] with Theorem 3.1. We also achieve tradeoffs between hcost and vcost for dense graphs, obtaining an online $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme.

Theorem 7.7. *There exists an online $(m\log n, \log n)$-scheme for graph connectivity, as well as an online $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme for each $\alpha \in [0,1]$. Any (h,v)-scheme (online or prescient) for connectivity requires $hv = \Omega(n)$ even when $m = O(n)$.*

Proof. We begin with the $(m\log n, \log n)$-scheme. If the graph is connected then there exists a spanning tree T directed towards the root and an injective labeling of the nodes $f : V \to [n]$ such that each non-root node with label j is linked to exactly one node with label greater than j. The helper outputs such a function f, and the verifier ensures that it is an injection. Then each (directed) edge (u,v) in T and its labels $f(u) < f(v)$ is presented in decreasing order of $f(u)$. The verifier checks this order, and ensures that it is consistent with f via fingerprinting (as per Lemma 5.1). The helper must also list all edges, so that the verifier can ensure that all T edges are from the input.

If the graph is not connected then the helper presents a connected component L of the graph. Each node is presented in lexicographic order, along with its label indicating whether or not it is in L, and each edge is presented along with the corresponding node labels. The verifier checks that $L \neq V$, uses fingerprinting to ensure no edge is omitted, and uses the multiset scheme of Lemma 5.1 to ensure that the node labels are consistent.

The $(n^{1+\alpha}\log n, n^{1-\alpha}\log n)$-scheme follows the same conceptual outline as above: if G is connected, the helper demonstrates this by providing a spanning tree; if G is disconnected, the helper identifies a connected component of the graph. In the first case, the helper provides a set of edges T claimed to be a spanning tree, and the verifier must check that (1) T is spanning and that (2) $T \subseteq E$. Checking (1) is accomplished as in the $(m\log n, 1)$ case, by appropriate labelling of the $O(n)$ edges, with $O(n)$ annotation. By Theorem 5.2, condition (2) can be checked with space $O(n^{1-\alpha}\log n)$ and annotation $O(n^{1+\alpha}\log n)$.

If G is disconnected, the helper presents a set $L \subset V$, $L \neq V$, and claims that L is disconnected from $V \setminus L$. Let A be the adjacency matrix of G, and let $\mathbf{x} \in \{0,1\}^n$ be the indicator vector of L. To check that L is as claimed, it suffices for the verifier to compute $A\mathbf{x}$, and check that the each non-zero entry of $A\mathbf{x}$ corresponds to vertices in L (intuitively, this means the set L' of vertices at distance one from L is contained in L). The first step uses the integer matrix-vector multiplication scheme of [17, Theorem 4]. This allows the verifier to ensure that the set $\{i : (A\mathbf{x})_i \neq 0\}$ matches L, via fingerprints.

For the lower bound, we reduce an instance of $\text{DISJ}_{n,2}$ to connectivity of a graph with $O(n)$ edges over nodes $v_{0,0} \ldots v_{3,n}$: create edges $(v_{j,0}, v_{j,i})$ for $j \in \{0,2,3\}$ and $i \in [n]$. Then if $x_i = 1$, add edge $(v_{0,i}, v_{1,i})$, else add edge $(v_{1,i}, v_{2,i})$; and if $y_i = 1$, add edge $(v_{1,i}, v_{3,i})$ else add edge $(v_{2,i}, v_{3,i})$. The resulting graph is connected only if x and y are not disjoint. The result follows from Theorem 4.3. □

Acknowledgements

We thank Yael Gertner, Sampath Kannan, and Mahesh Viswanathan for sharing [29]. We also thank Sudipto Guha and T. S. Jayram for helpful discussions.

References

[1] S. Aaronson and A. Wigderson. Algebrization: a new barrier in complexity theory. In *ACM Symposium on Theory of Computing*, pages 731–740, 2008.

[2] F. Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theoretical Computer Science*, 175(2):139–159, 1996.

[3] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *IEEE Symposium on Foundations of Computer Science*, pages 540–549, 2004.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[5] L. Babai, P. Frankl, and J. Simon. Complexity classes in communication complexity theory (preliminary version). In *IEEE Symposium on Foundations of Computer Science*, pages 337–347, 1986.

[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.

[7] P. Beame and D.-T. Huynh-Ngoc. On the value of multiple read/write streams for approximating frequency moments. In *IEEE Symposium on Foundations of Computer Science*, 2008.

[8] P. Beame, T. S. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. In *ACM Symposium on Theory of Computing*, pages 689–698, 2007.

[9] V. Braverman and R. Ostrovsky. Zero-one frequency laws. In *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*, pages 281–290, New York, NY, USA, 2010. ACM.

[10] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *ACM Principles of Database Systems*, pages 253–262, 2006.

[11] A. Chakrabarti, G. Cormode, and A. McGregor. Robust lower bounds for communication and stream computation. In *ACM Symposium on Theory of Computing*, pages 641–650, 2008.

[12] A. Chakrabarti, G. Cormode, and A. McGregor. Annotations in data streams. In *Proc. 36th International Colloquium on Automata, Languages and Programming*, pages 222–234, 2009.

[13] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *IEEE Conference on Computational Complexity*, pages 107–117, 2003.

[14] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15, 2004.

[15] E. Y. Chen. Geometric streaming algorithms with a sorting primitive. In *ISAAC*, pages 512–524, 2007.

[16] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in data streams. In *International Conference on Very Large Data Bases*, pages 464–475, 2003.

[17] G. Cormode, M. Mitzenmacher, and J. Thaler. Streaming graph computations with a helpful advisor. In *European Symposium on Algorithms*, 2010.

[18] G. Cormode, M. Mitzenmacher, and J. Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, pages 1–34, 2011. 10.1007/s00453-011-9598-y.

[19] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *ITCS '12: Proceedings of the 3rd Symposium on Innovations in Theoretical Computer Science*, 2012.

[20] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[21] G. Cormode, J. Thaler, and K. Yi. Verifying computations with streaming interactive proofs. *PVLDB*, 5(1):25–36, 2011.

[22] M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 323–334, 2002.

[23] C. Demetrescu, B. Escoffier, G. Moruz, and A. Ribichini. Adapting parallel algorithms to the w-stream model, with applications to graph problems. In *MFCS*, pages 194–205, 2007.

[24] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 714–723, 2006.

[25] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

[26] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008.

[27] J. Feigenbaum, S. Kannan, and J. Zhang. Annotation and computational geometry in the streaming model. Technical Report YALEU/DCS/TR-1249, Yale University, 2003.

[28] R. Freivalds. Fast probabilistic algorithms. In *MFCS*, pages 57–69, 1979.

[29] Y. Gertner, S. Kannan, and M. Viswanathan. NP and streaming verifiers. *Manuscript*, 2002.

[30] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. In *ACM Symposium on Theory of Computing*, pages 113–122, 2008.

[31] M. Grohe, A. Hernich, and N. Schweikardt. Randomized computations on large data sets: tight lower bounds. In *ACM Principles of Database Systems*, pages 243–252, 2006.

[32] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999.

[33] W. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[34] H. Jowhari and M. Ghodsi. New streaming algorithms for counting triangles in graphs. In *International Conference on Computing and Combinatorics*, pages 710–716, 2005.

[35] T. Kimbrel and R. K. Sinha. A probabilistic algorithm for verifying matrix products using $o(n^2)$ time and $\log_2 n + o(1)$ random bits. *Inf. Process. Lett.*, 45(2):107–110, 1993.

[36] H. Klauck. Rectangle size bounds and threshold covers in communication complexity. In *IEEE Conference on Computational Complexity*, pages 118–134, 2003.

[37] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[38] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios. Proof-infused streams: Enabling authentication of sliding window queries on streams. In *VLDB*, pages 147–158, 2007.

[39] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[40] S. Muthukrishnan. Data streams: algorithms and applications. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pages 413–413, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[41] S. Papadopoulos, Y. Yang, and D. Papadias. Cads: Continuous authentication on data streams. In *VLDB*, pages 135–146, 2007.

[42] A. Razborov. On the distributional complexity of disjontness. In *International Colloquium on Automata, Languages and Programming*, pages 249–253, 1990.

[43] A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

[44] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[45] P. A. Tucker, D. Maier, L. M. L. Delcambre, T. Sheard, J. Widom, and M. P. Jones. Punctuated data streams, 2005.

[46] K. Yi, F. Li, M. Hadjieleftheriou, G. Kollios, and D. Srivastava. Randomized synopses for query assurance on data streams. In *IEEE International Conference on Data Engineering*, 2008.

[47] M. Zelke. Weighted matching in the semi-streaming model. In *STACS*, pages 669–680, 2008.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de