Semi-Streaming Algorithms for Annotated Graph Streams*

Justin Thaler†

Yahoo Labs, New York, USA

Abstract

Considerable effort has been devoted to the development of streaming algorithms for analyzing massive graphs. Unfortunately, many results have been negative, establishing that a wide variety of problems require $\Omega(n^2)$ space to solve. One of the few bright spots has been the development of *semi-streaming* algorithms for a handful of graph problems – these algorithms use space $O(n \cdot \text{polylog}(n))$.

In the annotated data streaming model of Chakrabarti et al. [7], a computationally limited client wants to compute some property of a massive input, but lacks the resources to store even a small fraction of the input, and hence cannot perform the desired computation locally. The client therefore accesses a powerful but untrusted service provider, who not only performs the requested computation, but also *proves* that the answer is correct.

We consider the notion of *semi-streaming algorithms for annotated graph streams* (semi-streaming annotation schemes for short). These are protocols in which both the client's space usage and the length of the proof are $O(n \cdot \text{polylog}(n))$. We give evidence that semi-streaming annotation schemes represent a more robust solution concept than does the standard semi-streaming model. On the positive side, we give semi-streaming annotation schemes for two dynamic graph problems that are intractable in the standard model: (exactly) counting triangles, and (exactly) computing maximum matchings. The former scheme answers a question of Cormode [22]. On the negative side, we identify for the first time two natural graph problems (connectivity and bipartiteness in a certain edge update model) that can be solved in the standard semi-streaming model, but cannot be solved by annotation schemes of "sub-semi-streaming" cost. That is, these problems are as hard in the annotations model as they are in the standard model.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph streams, stream verification, annotated data streams, probabilistic proof systems

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.59

1 Introduction

The rise of cloud computing has motivated substantial interest in protocols for *verifiable data stream computation*. These protocols allow a computationally weak client (or *verifier*), who lacks the resources to locally store a massive input, to outsource the storage and processing of that input to a powerful but untrusted service provider (or *prover*). Such protocols provide a guarantee that the answer returned by the prover is correct, while allowing the verifier to make only a single streaming pass over the input.

* The full version of this paper is available at `http://arxiv.org/abs/1407.3462`.

† The majority of this work was performed while the author was at the Simons Institute for the Theory of Computing, UC Berkeley. Supported by a Research Fellowship from the Simons Institute for the Theory of Computing.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 59; pp. 59:1–59:14

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

Several recent works have introduced closely related models capturing the above scenario [18, 7, 8, 19, 6, 20, 12, 13, 11]. Collectively, these works have begun to reveal a rich theory, leveraging algebraic techniques developed in the classical theory of interactive proofs [23, 29, 4, 16] to obtain efficient verification protocols for a variety of problems that require linear space in the standard streaming model (sans prover).

The primary point of difference among the various models of verifiable stream computation that have been proposed is the amount of interaction that is permitted between the verifier and prover. The *annotated data streaming* model of Chakrabarti et al. [7] (subsequently studied in [12, 19, 6, 11]) is non-interactive, requiring the correctness proof to consist of just a single message from the prover to the verifier, while other models, such as the *Arthur–Merlin streaming protocols* of Gur and Raz [18, 6] and the *streaming interactive proofs* of Cormode et al. [13, 8] permit the prover and verifier to exchange two or more messages. Our focus in this paper is on the annotated data streaming model of Chakrabarti et al. – owing to their non-interactive nature, protocols in this model possess a number of desirable properties not shared by their interactive counterparts, such as reusability (see Section 2 for details). We are concerned with protocols for problems on graph streams, described below.

Graph Streams. The ubiquity of massive relational data sets (derived, e.g., from the Internet and social networks) has led to detailed studies of data streaming algorithms for analyzing graphs. In this setting, the data stream consists of a sequence of edges, defining a graph G on n nodes, and the goal is to compute various properties of G (is G connected? How many triangles does G contain?). Unfortunately, many results on graph streaming have been negative: essentially any graph problem of the slightest practical interest requires $\Omega(n)$ space to solve in the standard streaming model, and many require $\Omega(n^2)$ space even to approximate. Due to their prohibitive cost in the standard streaming model, many basic graph problems are ripe for outsourcing.

One of the few success stories in the study of graph streams has been the identification of the *semi-streaming* model as something of a "sweet spot" for streaming algorithms[26, 24]. The semi-streaming model is characterized by an $O(n \cdot \operatorname{polylog} n)$ space restriction, i.e., space proportional to the number of nodes rather than the number of edges. For dense graphs this represents considerably less space than that required to store the entire graph. It has long been known that problems like connectivity and bipartiteness possess semi-streaming algorithms when the stream consists only of edge insertions, with no deletions. Recently, semi-streaming algorithms have been provided for these and other problems even for dynamic graph streams, which contain edge deletions as well as insertions [1, 2, 14]. We direct the interested reader toward the recent survey of McGregor [25] on graph stream algorithms.

In this work, we consider the notion of *semi-streaming annotation schemes* for graph problems. Here, the term "scheme" refers to a protocol in the annotated data streaming model. A scheme's total cost is defined to be the sum of the verifier's space usage (referred to as the *space cost* of the scheme) and the length of the proof (referred to as the scheme's *help cost*). A scheme is said to be semi-streaming if its total cost is $O(n \cdot \operatorname{polylog} n)$.

We give evidence that semi-streaming annotation schemes represent a substantially more robust solution concept (i.e., a "sweeter spot") for graph problems than does the standard semi-streaming model. First, we give novel semi-streaming annotation schemes for two challenging dynamic graph problems, counting triangles and maximum matching, that require $\Omega(n^2)$ space in the standard streaming model. The total cost of these schemes is provably optimal up to a logarithmic factor. Second, we show that two canonical problems that do possess semi-streaming algorithms in the standard streaming model (connectivity

Table 1 Comparison of our new scheme for TRIANGLES to prior work.

Reference	TRIANGLES Scheme Costs (help cost, space cost)	Total Cost Achieved
[7]	$(n^2 \log n, \log n)$	$O(n^2 \log n)$
[7]	$(x \log n, y \log n)$ for any $x \cdot y \geq n^3$	$O(n^{3/2} \log n)$
Theorem 1	$(n \log n, n \log n)$	$O(n \log n)$

and bipartiteness in a certain edge update model) are *just as hard* in the annotations model. Formally, we show that any scheme for these problems with space cost $O(n^{1-\delta})$ requires a proof of length $\Omega(n^{1+\delta})$ for any $\delta > 0$. Thus, for these problems, giving a streaming algorithm access to an untrusted prover does not allow for a significant reduction in cost relative to what is achievable without a prover. This gives further evidence for the robustness of semi-streaming annotation schemes as a solution concept: while several fundamental problems that cannot be solved by standard semi-streaming algorithms can be solved by semi-streaming annotation schemes, there are problems that do have semi-streaming solutions in the standard model that cannot be solved by schemes of sub-semi-streaming cost.

1.1 Summary of Contributions and Techniques

Throughout this informal overview, n will denote the number of nodes in the graph defined by the data stream, and m the number of edges. To avoid boundary cases in the statement of our lower bounds, we assume that the help cost of any scheme is always at least 1 bit.

1.1.1 New Semi-Streaming Annotation Schemes

Prior work has given semi-streaming annotation schemes for two graph problems that require $\Omega(n^2)$ space in the standard semi-streaming model: bipartite perfect matching [7], and shortest s-t path in graphs of polylogarithmic diameter [12]. As discussed above, we give semi-streaming schemes for two more challenging problems: maximum matching (MAXMATCHING) and counting triangles (TRIANGLES). Both schemes apply to dynamic graph streams. We begin by describing our result for TRIANGLES.

▶ **Theorem 1** (Informal Version of Theorem 4). *There is a scheme for* TRIANGLES *with total cost $O(n \log n)$. Every scheme requires the product of the space and help costs to be $\Omega(n^2)$, and hence has total cost $\Omega(n)$.*

Theorem 1 affirmatively answers a question of Cormode [22], resolves the Merlin-Arthur communication complexity of the problem up to a logarithmic factor, and improves over the best previous bound of $O(n^{3/2} \log n)$, due to [7] (see Table 1 for a comparison).

As is the case for essentially all non-trivial protocols for verifiable stream computation, the scheme of Theorem 1 uses algebraic techniques related to the famous *sum-check protocol* of Lund et al. [23] from the classical theory of interactive proofs. Yet, our scheme deviates in a significant way from all earlier annotated data stream and interactive proof protocols [15, 8, 7, 29]. Roughly speaking, in previous protocols, the verifier's updates to her memory state were commutative, in the sense that reordering the stream tokens would not change the final state reached by the verifier. However, our new verifier is inherently non-commutative: her update to her state at time i depends on her actual state at time i, and reordering the stream tokens can change the final state reached by the verifier. The full version of the paper contains further discussion of this point.

Table 2 Comparison of our new scheme for MaxMatching to prior work.

Reference	MaxMatching Scheme Costs (help cost, space cost)	Total Cost Achieved
[12]	$(m \log n, \log n)$	$O(m \log n)$
Theorem 5	$(n \log n, n \log n)$	$O(n \log n)$

▶ **Theorem 2** (Informal Version of Theorem 5). *There is a scheme for* MaxMatching *with total cost $O(n \log n)$. Every scheme for* MaxMatching *requires the product of the space and help costs to be $\Omega(n^2)$, and hence has total cost $\Omega(n)$.*

Our scheme combines the Tutte-Berge formula with algebraic techniques to allow the prover to establish matching upper and lower bounds on the size of a maximum matching in the input graph. Schemes for maximum matching had previously been studied by Cormode et al. [12], but this prior work only gave schemes with help cost proportional to the number of edges, which is $\Omega(n^2)$ in dense graphs (like us, Cormode et al. exploited the Tutte-Berge formula, but did not do so in a way that achieved help cost sublinear in the input size). Prior work had also given a scheme achieving optimal tradeoffs between help and space costs for *bipartite perfect matching* [7, Theorem 7.5] – our scheme for MaxMatching can be seen as a broad generalization of [7, Theorem 7.5].

1.1.2 New Lower Bounds

On the other hand, we identify, for the first time, natural graph problems that possess standard semi-streaming algorithms, but in a formal sense are just as hard in the annotations model as they are in the standard streaming model. The problems that we consider are connectivity and bipartiteness in a certain edge update model that we call the XOR update model. In this update model, the stream $\langle e_1, \ldots, e_m \rangle$ is a sequence of edges from $[n] \times [n]$, which define a graph $G = (V, E)$ via: $e \in E \iff |\{i : e_i = e\}| = 1 \mod 2$. Intuitively, each stream update e_i is interpreted as changing the status of edge e_i: if it is currently in the graph, then the update causes e_i to be deleted; otherwise e_i is inserted. Our lower bound holds for schemes for connectivity and bipartiteness in the XOR update model, even under the promise that $e_1, \ldots, e_{m-n}$ are all unique (hence, all but the last n stream updates correspond to edge insertions), and the last n updates are all incident to a single node.

▶ **Theorem 3** (Informal Version of Corollary 7). *Consider any scheme for* Connectivity *or* Bipartiteness *in the XOR update model with help cost c_a and and space cost c_v. Then $(c_a + n) \cdot c_v \geq n^2$, even under the promise that the first $m - n$ stream updates are all unique, and the last n stream updates are all incident to a single node. In particular, the total cost of any annotation scheme for these problems is $\Omega(n)$.*

Both connectivity and bipartiteness in the XOR update model possess standard semi-streaming algorithms [1].[1] Hence, Theorem 3 implies that the total cost of any annotation scheme is at most a polylogarithmic factor smaller than the problems' space complexity in the standard streaming model. Like all prior work establishing lower bounds on the cost of

[1] The algorithms of [1] are described in the turnstile update model, in which each stream update explicitly specifies whether to insert or delete a (copy of) an edge. However, these algorithms are easily modified to apply to the XOR update model as well. In brief, these algorithms have L_0-sampling algorithms at their core. Existing L_0-samplers are in turn built on top of sparse recovery algorithms (see, e.g., [10]), and many sparse recovery algorithms can be implemented in the XOR update model directly (see, e.g., [17]).

protocols for verifiable stream computation, our lower bounds are established using notions of *Merlin-Arthur communication complexity* [7, 12, 19, 8, 18].

Prior to this work, only one other problem was known to be as hard (up to logarithmic factors) in the annotations model as in the standard streaming model [6, Corollary 3.3]. The problem considered in [6, Corollary 3.3] was an "exponentially sparse" variant of the classic INDEX problem, in which the data stream consists of a vector $x \in \{0,1\}^n$ promised to have Hamming weight $O(\log n)$, followed by an index $i \in [n]$, and the goal is to output the value x_i. Connectivity and bipartiteness are arguably more natural problems, and are qualitatively different, as we now explain.

On an informal level, the reason the exponentially sparse INDEX problem is hard in the annotations model is that any "useful" annotation must at least specify a single index into the vector x, which requires $\log n$ bits of annotation. And since x is exponentially sparse, $\log n$ is actually equal (up to a constant factor) to the space complexity of a standard streaming algorithm for the problem. In our view, BIPARTITENESS and CONNECTIVITY are hard in the annotations for a different reason – roughly speaking, any useful annotation for these problems must at least specify, for each node u, the side of the bipartition or the connected component in which u resides.

Overview of the Proof. Our proof of Theorem 3 works by specifying a reduction from the INDEX problem on inputs of length n^2, for which a lower bound of $\Omega(n^2)$ on the product of the help and space costs of any annotation scheme was established in [7], to CONNECTIVITY and BIPARTITENESS on graphs with n nodes and $\Theta(n^2)$ edges.

Notice that in the standard (sans prover) streaming model, the INDEX problem on n^2 variables is strictly harder than connectivity and bipartiteness problems on graphs with n nodes, as the former requires $\Omega(n^2)$ space, while the latter two problems require only $O(n \cdot \text{polylog}(n))$ space. Yet Theorem 3 establishes that in the annotations model, all three problems are of essentially equivalent difficulty (in particular, schemes of total cost $\tilde{O}(n)$ are necessary and sufficient to solve all three problems). To establish such a result, it is necessary to use a reduction that is specifically tailored to the annotations model, in the sense that the reduction must not apply in the standard streaming model (since INDEX and CONNECTIVITY are not of equivalent difficulty in the standard setting). Namely, in our reduction from INDEX to connectivity, the prover *helps the verifier* transform an instance of the INDEX problem into a CONNECTIVITY instance. This "help" consists of $\Theta(n)$ bits, and this is why our lower bound is of the form $(c_a + n) \cdot c_v \geq n^2$. This is in contrast to prior lower bounds, which, with the exception of [6, Corollary 3.3], were of the form $c_a \cdot c_v = \Omega(C)$ for some quantity C.

1.2 Other Related Work

As discussed above, several recent papers [11, 8, 13, 19, 20, 18, 6, 7, 12] have all studied annotated data streams and closely related models for verifiable stream computation. Refinements and implementations [11, 31] have demonstrated genuine practicality for many of the protocols developed in this line of work. Protocols for verifiable stream computation have also been studied in the cryptography community [9, 27]. These works only require security against cheating provers that run in polynomial time, whereas we require security to hold even against computationally unbounded provers. In exchange for the weaker security guarantee, these protocols may achieve properties that are unattainable in our information-theoretic setting. For example, some of these protocols achieve a stronger form of reusability than we do (see Section 2 for our definition of reusability) – they remain secure for many uses even if the prover learns all of the verifier's accept/reject decisions. The work of Chung et al. [9]

uses fully homomorphic encryption (FHE), which remains far from practical at this time. Papamanthou et al. [27] avoid the use of FHE, but handle only much simpler queries (e.g., point queries and range search) than the graph problems we consider here.

2 Models of Streaming Computation

Our presentation of data streaming models closely follows Chakrabarti et al. [6]. Recall that a (standard) data stream algorithm computes a function f of an input sequence $\mathbf{x} \in \mathcal{U}^m$, where m is the number of stream updates, and $\mathcal{U}$ is some data universe. The algorithm has only sequential access to $\mathbf{x}$, uses a limited amount of space, and has access to a random string. The function f may or may not be Boolean. An annotated data stream algorithm, or a *scheme*, is a pair $\mathcal{A} = (\mathfrak{h}, V)$, consisting of a help function $\mathfrak{h} : \mathcal{U}^m \times \{0,1\}^* \to \{0,1\}^*$ used by a *prover* and a data stream algorithm run by a *verifier*, V. The prover provides $\mathfrak{h}(\mathbf{x})$ as annotation to be read by the verifier. We think of $\mathfrak{h}$ as being decomposed into $(\mathfrak{h}_1, \ldots, \mathfrak{h}_m)$, where the function $\mathfrak{h}_i : \mathcal{U}^m \to \{0,1\}^*$ specifies the annotation supplied after the arrival of 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 streaming verifier, who has access to a (private) random string r, then processes this annotated stream, eventually giving an output $\text{out}^V(\mathbf{x}^{\mathfrak{h}}, r)$.

We say a scheme is *online* if each function $\mathfrak{h}_i$ depends only on $(x_1, \ldots, x_i)$. The scheme $\mathcal{A} = (\mathfrak{h}, V)$ is said to be δ_s-sound and δ_c-complete for the function F if the following hold:

1. For all $\mathbf{x} \in \mathcal{U}^m$, we have $\Pr_r[\text{out}^V(\mathbf{x}^{\mathfrak{h}}, r) \neq F(\mathbf{x})] \leq \delta_c$.
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 $\Pr_r[\text{out}^V(\mathbf{x}^{\mathfrak{h}'}, r) \notin \{F(\mathbf{x})\} \cup \{\perp\}] \leq \delta_s$.

An output of "$\perp$" indicates that the verifier rejects the prover's claims in trying to convince the verifier to output a particular value for $F(\mathbf{x})$. We define $\text{err}(\mathcal{A})$ to be the minimum value of $\max\{\delta_s, \delta_c\}$ such that the above conditions are satisfied. We define the *annotation length* $\text{hc}(\mathcal{A}) = \max_{\mathbf{x}} \sum_i |\mathfrak{h}_i(\mathbf{x})|$, the total size of the prover's communications, and the *verification space cost* $\text{vc}(\mathcal{A})$ to be the space used by the verifier. We say that $\mathcal{A}$ is an online (c_a, c_v) scheme if $\text{hc}(\mathcal{A}) = O(c_a)$, $\text{vc}(\mathcal{A}) = O(c_v)$, and $\text{err}(\mathcal{A}) \leq \frac{1}{3}$ (the constant 1/3 is arbitrary and chosen by convention).

Chakrabarti et al. [7] also define the notion of a *prescient* scheme, which is the same as an online scheme, except the annotation at any time i is allowed to depend on data which the verifier has not seen yet. Prescient schemes have the undesirable property that the prover may need to "see into the future" to convince the verifier to produce the correct output. Note that even though our Triangles and MaxMatching protocols are online, they are optimal up to logarithmic factors *even among prescient schemes* (see Theorems 4 and 5).

While the annotated data streams model allows the prover to interleave the annotation with the stream, in all of the schemes we present in this paper, all of the annotation comes at the end of the stream. This property avoids any need for fine-grained coordination between the annotation and the stream, and permits the prover to send the annotation as a single email attachment, or post it to a website. We clarify that the *lower bounds* for Connectivity and Bipartiteness that we establish in Section 4 apply to any online scheme, even those which interleave the annotation with the stream.

The schemes we present in this work permit a natural form of *reusability*: if the verifier wants to compute a function f on a given dataset $\mathbf{x}$, the verifier can receive $f(\mathbf{x})$ (with

a correctness proof), and check the validity of the proof using a "secret state" that she computed while observing the stream $\mathbf{x}$. Further updates to the stream $\mathbf{x}$ can then occur, yielding a (longer) stream $\mathbf{x}'$, and the verifier can update her secret in a streaming fashion. The verifier may then receive the answer $f(\mathbf{x}')$ (with a correctness proof) on the updated dataset, and check its correctness using the updated secret state. The probability that the verifier gets fooled into outputting an incorrect answer on even a single query grows only linearly with the number of times the prover sends the verifier an answer. Such reusability is not possible with interactive solutions [13, 8, 20], which require the verifier to reveal information about r over the course of the protocol.

3 Upper Bounds

Graph Streams in the Strict Turnstile Model. The annotation schemes of this section apply to graph streams in the strict turnstile update model. In this model, a data stream σ consists of a sequence of undirected edges, accompanied by (signed) multiplicities: $\langle (e_1, \Delta_1), \ldots, (e_m, \Delta_m) \rangle$. Each edge $e_i \in [n] \times [n]$, and each $\Delta_i \in \mathbb{Z}$. An update (e_i, Δ_i) with $\Delta_i > 0$ is interpreted as an insertion of Δ_i copies of edge e_i into graph G. If $\Delta_i < 0$, the update is interpreted as a deletion of Δ_i copies of edge e_i. It is assumed that at the end of the stream, no edge has been deleted more times than it has been inserted (all of our protocols work even if this property does not hold at *intermediate* time steps, as long as the property holds after the final stream update has been processed).[2] When analyzing the time costs of our schemes, we assume that any addition or multiplication in a finite field of size poly(n) takes one unit of time.

3.1 A Semi-Streaming Scheme for Counting Triangles

In the TRIANGLESproblem, the goal is to determine the number of unordered triples of distinct vertices (u, v, z) such that edges (u, v), (v, z), and (z, u) all appear in G. More generally, if these edges appear with respective multiplicities M_1, M_2, and M_3, we view triple (u, v, z) as contributing $M_1 \cdot M_2 \cdot M_3$ triangles to the total count.[3] Computing the number of triangles is a well-studied problem [3] and there has been considerable interest in designing algorithms in a variety of models including the data stream model [5, 28], MapReduce [30], and the quantum query model [21]. One motivation is the study of social networks where important statistics such as the clustering coefficient and transitivity coefficient are based on the number of triangles. Understanding the complexity of counting triangles captures the ability of a model to perform a non-trivial correlation within large graphs. Chakrabarti et al. [7] gave two annotated data streaming protocols for this problem. The first protocol had help cost $O(n^2 \log n)$, and space cost $O(\log n)$. The second protocol achieved help cost $O(x \log n)$ and space cost $O(y \log n)$ for any x, y such that $x \cdot y \geq n^3$. In particular, by setting $x = y = n^{3/2}$, the second protocol of Chakrabarti et al. ensured that both help cost and space cost equaled $O\left(n^{3/2} \log n\right)$. Cormode [22] asked whether it is possible to achieve an annotated data streaming protocol in which both the help cost and space cost are $\tilde{O}(n)$. We

[2] We do not consider edges with negative weights at the end of the stream because it is unclear what is the most meaningful way to define problems like TRIANGLES and MAXMATCHING in this setting. See Footnotes 3 and 4.

[3] The TRIANGLESscheme of Theorem 3.1 gives meaningful results even if the M_i's may be negative: a triangle with an odd (even) number of edges of negative multiplicity contributes a negative (positive) number to the total triangle count.

answer this question in the affirmative.

▶ **Theorem 4** (Formal Statement of Theorem 1)**.** *Assume there is an a priori upper bound $B \leq \text{poly}(n)$ on the multiplicity of any edge in G. There is an online scheme for* Triangles *with space and help costs $O(n \log n)$. Every scheme (online or prescient) requires the product of the space and help costs to be $\Omega(n^2)$, and hence total cost $\Omega(n)$, even for $B = 1$, and even if G is promised to have exactly 0 or 1 triangles.*

Discussion. Before proving Theorem 4, it is instructive to consider a simple *interactive* streaming verification protocol for Triangles(described in detail in the full version of the paper). At a high level, the interactive protocol applies the sum-check protocol of Lund et al. [23] to a suitably defined multivariate polynomial h. The space and communication costs of this protocol are comparable to that of Theorem 4; the advantage of Theorem 4 over this simple interactive solution is that Theorem 4 gives a protocol that is *non-interactive*, and comes with the associated reusability benefits described in Section 2. Theorem 4 below effectively removes the interaction from the simple interactive protocol as follows. We identify a *univariate* polynomial $g(Z)$ of degree $O(n)$ such that the number of triangles in G equals $\sum_{z \in [n]} g(Z)$. Moreover, we show that the verifier can evaluate $g(r)$ for any point $r \in \mathbb{F}$ in space $O(n \log |\mathbb{F}|)$ with a single streaming pass over the input. It follows that applying the sum-check protocol to g yields a scheme with costs claimed in Theorem 4. The polynomial g that we identify is defined as a sum of m constituent polynomials, one per stream update.

Proof of Theorem 4. The lower bound was proved in [7, Theorem 7.1]. Details of the upper bound follow. Let G_i denote the graph defined by the first i stream updates $\langle (e_1, \Delta_1), \ldots, (e_i, \Delta_i) \rangle$, and let $E_i \colon [n] \times [n] \to \mathbb{Z}$ denote the function that outputs the multiplicity of the edge (u, v) in graph G_{i-1}. On edge update $e_i = (u_i, v_i)$, notice that the number of triangles that e_i *completes* in E_i is precisely $\Delta_i \cdot \sum_{z \in [n]} E_i(u_i, z) E_i(v_i, z)$. Thus, the total number of triangles in the graph $G = G_m$ is precisely $\sum_{i \leq m} \Delta_i \sum_{z \in [n]} E_i(u_i, z) E_i(v_i, z)$. Let $\mathbb{F}$ denote a field of prime order $6(B \cdot n)^3 \leq |\mathbb{F}| \leq 12(B \cdot n)^3$, and let $\tilde{E}_i(X, Y)$ denote the unique polynomial over $\mathbb{F}$ of degree at most n in each variable X, Y such that $\tilde{E}_i(u, v) = E_i(u, v)$ for all $(u, v) \in [n] \times [n]$. Then the number of triangles in G equals

$$\sum_{i \leq m} \Delta_i \sum_{z \in [n]} E_i(u_i, z) E_i(v_i, z) = \sum_{i \leq m} \Delta_i \sum_{z \in [n]} \tilde{E}_i(u_i, z) \tilde{E}_i(v_i, z) = \sum_{z \in [n]} \sum_{i \leq m} \Delta_i \tilde{E}_i(u_i, z) \tilde{E}_i(v_i, z). \tag{1}$$

In turn, the right hand side of Equation (1) can be written as $\sum_{z \in [n]} g(z)$, where g denotes the *univariate* polynomial defined via: $g(Z) = \sum_{i \leq m} \Delta_i \cdot \tilde{E}_i(u_i, Z) \tilde{E}_i(v_i, Z)$. Notice $g(Z)$ is a univariate polynomial of degree at most $2n$. Our scheme proceeds as follows.

Prover's computation. At the end of the stream, the prover sends a univariate polynomial $s(Z)$ of degree at most $2n$, where $s(Z)$ is claimed to equal $g(Z)$. Notice that since $s(Z)$ has degree at most $2n$, $s(Z)$ can be specified by sending its values on all inputs in $\{0, \ldots, 2n\}$ – this requires help cost $O(n \log |\mathbb{F}|) = O(n \log n)$.

Verifier's computation. At the start of the stream, the verifier picks a random field element $r \in \mathbb{F}$, and keeps the value of r secret from the prover. We will show below that the verifier can evaluate $g(r)$ with a single streaming pass over the input, using space $O(n \log n)$. The

verifier checks whether $s(r) = g(r)$. If this check fails, the verifier halts and rejects. If the check passes, the verifier outputs $\sum_{z \in [n]} s(z)$ as the correct answer.

We now explain how the verifier can evaluate $g(r)$ with a single streaming pass over the input. The high-level idea is as follows. $g(r)$ is defined as a sum of m terms, where the ith term equals $\Delta_i \cdot \tilde{E}_i(u_i, r)\tilde{E}_i(v_i, r)$. For each $u \in [n]$, we will show how the verifier can incrementally maintain the quantity $\tilde{E}_i(u, r)$ at all times i. The verifier will maintain all n of these quantities, resulting in a total space cost of $O(n \log |\mathbb{F}|) = O(n \log n)$. With these quantities in hand, it is straightforward for the verifier to incrementally maintain the sum $\sum_{j \leq i} \Delta_j \cdot \tilde{E}_j(u_j, r)\tilde{E}_j(v_j, r)$ at all times i: upon the ith stream update, the verifier simply adds $\Delta_i \cdot \tilde{E}_i(u_i, r) \cdot \tilde{E}_i(v_i, r)$ to the running sum.

To maintain the quantity $\tilde{E}_i(u, r)$, we begin by writing the bivariate polynomial $\tilde{E}_i(X, Y)$ in a convenient form. Given a pair $(u, v) \in [n] \times [n]$, let $\tilde{\delta}_{(u,v)}$ denote the following (Lagrange) polynomial: $\tilde{\delta}_{(u,v)}(X, Y) = \left(\frac{\prod_{1 \leq u' \leq n : u' \neq u}(X - u')}{\prod_{1 \leq u' \leq n : u' \neq u}(u - u')}\right)\left(\frac{\prod_{1 \leq v' \leq n : v' \neq v}(Y - v')}{\prod_{1 \leq v' \leq n : v' \neq v}(v - v')}\right)$. Notice that $\tilde{\delta}_{(u,v)}$ evaluates to 1 on input (u, v), and evaluates to 0 on all other inputs $(x, y) \in [n] \times [n]$. Thus, we may write $\tilde{E}_i(X, Y) = \sum_{j \leq i} \tilde{\delta}_{(u_j, v_j)}(X, Y)$. In particular, for each node $u \in [n]$,

$$\tilde{E}_i(u, r) = \tilde{E}_{i-1}(u, r) + \tilde{\delta}_{(u_i, v_i)}(u, r) + \tilde{\delta}_{(v_i, u_i)}(u, r).$$

Thus, the verifier can incrementally maintain the quantity $\tilde{E}_i(u, r)$ in a streaming manner using space $O(\log |\mathbb{F}|)$: while processing the ith stream update, the verifier simply adds $\tilde{\delta}_{(u_i, v_i)}(u, r) + \tilde{\delta}_{(v_i, u_i)}(u, r)$ to the running sum tracking $\tilde{E}_i(u, r)$.

Completeness. It is evident that if the prover sends the true polynomial $g(Z)$, then the verifier's check will pass, and the verifier will output the correct number of triangles.

Soundness. If the prover sends a polynomial $s(Z) \neq g(Z)$, then with probability at least $1 - 2n/|\mathbb{F}| \geq 1 - 1/(3n^2)$ over the verifier's random choice of $r \in \mathbb{F}$, it will hold that $s(r) \neq g(r)$. Hence, with probability at least $1 - 1/(3n^2) \geq 2/3$, the verifier's check will fail. ◀

Several remarks regarding Theorem 4 are in order.

- **Verifier Time.** The verifier in the protocol of Theorem 4 can process each stream update in constant time as follows. On stream update $e_i = (u_i, v_i)$, the verifier must add $\tilde{\delta}_{(u_i, v_i)}(u, r) + \tilde{\delta}_{(v_i, u_i)}(u, r)$ to each of the $\tilde{E}_i(u, r)$ values. However, it is straightforward to check that $\tilde{\delta}_{(u_i, v_i)}(u, r) = 0$ for all $u \neq u_i$, so the verifier need only update two quantities at time i: $\tilde{E}_i(u_i, r)$ and $\tilde{E}_i(v_i, r)$. We explain how both of these updates can be computed in constant time. It can be seen that $\tilde{\delta}_{(u_i, v_i)}(u_i, r) = \frac{\prod_{1 \leq v' \leq n : v' \neq v_i}(r - v')}{\prod_{1 \leq v' \leq n : v' \neq v_i}(v_i - v')}$. The right hand side of this equation can be computed in $O(1)$ time if the verifier maintains a pre-computed lookup table consisting of $O(n)$ field elements. Specifically, for each $v \in [n]$, it suffices for the verifier to maintain the quantities $q_1(v) := \prod_{1 \leq v' \leq n : v' \neq v}(r - v')$ and $q_2(v) = \left(\prod_{1 \leq v' \leq n : v' \neq v}(v - v')\right)^{-1}$. All $O(n)$ of these quantities can be computed in pre-processing in total time $O(n \log n)$, where the $\log n$ term is due to the time required to compute a multiplicative inverse in the field $\mathbb{F}$. Indeed, $q_1(1)$ and $q_2(1)$ can be computed naively in $O(n)$ time, and then for any $v > 1$, $q_1(v)$ and $q_2(v)$ can be computed in $O(\log n)$ time from $q_1(v-1)$ and $q_2(v-1)$ via the identities $q_1(v) = q_1(v-1) \cdot (r - v)^{-1} \cdot (r - v + 1)$ and $q_2(v) = q_2(v-1) \cdot (v-1)^{-1} \cdot (v - 1 - n)$.

Table 3 Statement of Time Costs For Our TrianglesScheme (Theorem 4).

Verifier Pre-Processing Time	Verifier Time Per Stream Update	Verifier Time to Process Proof	Prover Total Time
$O(n \log n)$	$O(1)$	$O(n)$	$O(m \cdot n)$

Finally, the verifier can process the proof in time $O(n)$. Recall that the proof consists of the values $s(x)$ for $x \in \{0, \ldots, 2n\}$, and the verifier simply needs to compute $\sum_{1 \leq x \leq n} s(x)$ as well as $s(r)$. The first quantity can trivially be computed in time $O(n)$, and the second can be computed in time $O(n)$ as well using standard techniques (see, e.g., [11]).

- **Prover Time.** The honest prover in the protocol of Theorem 4 can be implemented to run in time $O(m \cdot n)$. Indeed, the honest prover needs to evaluate $g(x)$ for $O(n)$ points $x \in \mathbb{F}$, and we have explained above how $g(x)$ can be computed in $O(m)$ time (in fact, in $O(1)$ time per stream update). This is comparable to the naive triangle counting algorithm that, for each edge and node combination, tests whether the two edges incident on the edge and node exist in the graph.
  The time costs for both the prover and verifier are summarized in Table 3.
- **MA communication.** Theorem 4 implies that the (online) MA communication complexity of counting triangles is $O(n \log n)$ (see Section 4.1 for the definition of the (online) MA communication model). This essentially matches an $\Omega(n)$ lower bound on the (even non-online) MA communication complexity of the problem, proved by Chakrabarti et al. [7] via a standard reduction to Set-Disjointness, and answers a question of Cormode [22].
- **Extensions: Counting Structures Other Than Triangles.** Let H be a graph on k vertices. It is possible to extend the protocol of Theorem 4 to count the number of occurrences of H as a subgraph of G. The protocol requires $k - 2$ rounds, and its help and space costs are $O(k^3 n \log n)$ and $O(kn \log n)$. Details are deferred to the full version.

3.2 A Semi-Streaming Scheme for Maximum Matching

We give a semi-streaming scheme for the MaxMatching problem in general graphs.[4] Due to space constraints, the proof of Theorem 5 is deferred to the full version.

▶ **Theorem 5** (Formal Version of Theorem 2). *Assume there is an a priori upper bound $B \leq \text{poly}(n)$ on the multiplicity of any edge in G. There is an online scheme for* MaxMatching *of total cost $O(B \cdot n \log n)$. Every scheme for* MaxMatching *(online or prescient) requires the product of the space and help costs to be $\Omega(n^2)$, and hence total cost $\Omega(n)$, even for $B = 1$.*

4 Lower Bounds for Connectivity and Bipartiteness

In this section, we establish our lower bounds on the cost of online schemes for Connectivity and Bipartiteness in the XOR update models. Like almost all previous lower bounds for data stream computations, our lower bounds use reductions from problems in communication complexity. To model the prover in a scheme, the appropriate communication setting is Merlin-Arthur communication, which we now introduce.

[4] It is possible to modify our scheme to give meaningful answers on graphs with edges of negative multiplicity. Specifically, the modified scheme can treat edges of negative multiplicity as having strictly positive multiplicity. We omit the details for brevity.

4.1 Merlin-Arthur Communication

Consider a communication game involving three parties, named Alice, Bob, and Merlin. Alice holds an input $x \in \mathcal{X}$, Bob and input $y \in \mathcal{Y}$, and Merlin is omniscient (he sees both x and y) but untrusted. Alice and Bob's goal is to compute $f(x,y)$ for some agreed upon function $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$. In an MA communication protocol $\mathcal{P}$, Merlin first broadcasts a message m_M to both Alice and Bob. Alice and Bob then engage in a randomized communication protocol, before outputting a single bit. To clarify, Merlin does not learn the randomness that Alice and Bob use until after sending the message $m_M(x,y)$. For each input (x,y), the protocol $\mathcal{P}$ defines a game between Merlin, Alice, and Bob, in which Merlin's goal is to make Alice and Bob output 1. We define $\mathbf{Val}^{\mathcal{P}}(x,y)$ to be Merlin's probability of winning with optimal play. Given a Boolean function f, we say that $\mathcal{P}$ computes f if, for all (x,y) we have (1) $f(x,y)=0 \Longrightarrow \mathbf{Val}^{\mathcal{P}}(x,y) \leq 1/3$, and (2) $f(x,y)=1 \Longrightarrow \mathbf{Val}^{\mathcal{P}}(x,y) \geq 2/3$. We refer to the Property (1) as *soundness* and Property (2) as *completeness*.

The *help cost*, or $\mathrm{hc}(\mathcal{P})$, of $\mathcal{P}$ is $\max_{(x,y)} |m_M(x,y)|$, i.e., the maximum length of Merlin's message in bits. The *verification cost*, or $\mathrm{vc}(\mathcal{P})$, of $\mathcal{P}$ is the maximum number of bits that Alice and Bob exchange, where the maximum is taken over all inputs (x,y), all possible Merlin messages m_M, and all choices of Alice and Bob's randomness. The *total cost* of $\mathcal{P}$ is the sum of the help and verification costs of $\mathcal{P}$.

In an *online* MA (OMA) communication protocol, neither Merlin nor Bob can talk to Alice. Given any online scheme for a function f, we naturally obtain an OMA protocol $\mathcal{P}$ for the communication problem in which Alice holds a prefix of a stream, Bob holds a suffix, and the goal is to evaluate f on the concatenated stream $x \circ y$. Hence, if we establish lower bounds on the help and verification costs of any OMA protocol for f, an equivalent lower bound on the help and space costs of any online scheme for f follows.

In this section, we establish lower bounds on the help and verification costs of any OMA protocol for the Disconnectivity and Bipartiteness problems in the XOR update model. More precisely, we consider the communication problems Disconnectivity$^{\mathrm{cc}}$ and Bipartiteness$^{\mathrm{cc}}$ in which Alice holds the first $m-n$ tuples in a graph stream in the XOR update model, Bob holds the length n tuples, and the output function evaluates to 1 if and only if the resulting graph is disconnected or bipartite, respectively.[5]

4.2 The Lower Bound

▶ **Theorem 6.** *Consider any OMA protocol $\mathcal{P}$ for* Disconnectivity$^{\mathrm{cc}}$ *or* Bipartiteness$^{\mathrm{cc}}$. *Then it holds that* $(\mathrm{hc}(\mathcal{P})+n)\cdot \mathrm{vc}(\mathcal{P}) = \Omega(n^2)$. *This holds even under the promise that the first $m-n$ stream updates (i.e., Alice's input) are all unique, and the last n stream updates (i.e., Bob's input) are all incident to a single node. In particular, the total cost of $\mathcal{P}$ is $\Omega(n)$.*

Proof. We prove the lower bound Disconnectivity$^{\mathrm{cc}}$problem. The proof for Bipartiteness$^{\mathrm{cc}}$is similar, and is deferred to the full version of the paper. Let $\mathcal{P}$ denote any OMA protocol for Disconnectivity$^{\mathrm{cc}}$that works on graphs with $n+1$ nodes, under the promise

[5] The reason that we consider Disconnectivity$^{\mathrm{cc}}$ rather than Connectivity$^{\mathrm{cc}}$ is the asymmetric way that inputs in $F^{-1}(0)$ and $F^{-1}(1)$ in the definition of OMA communication complexity. Recall that the OMA communication problem for a decision problem F requires only that if $F(x)=1$ then there is some prover that will cause the verifier to accept with high probability, and if $F(x)=0$ then there is no such prover. (By contrast, our definition of a scheme for a function F requires there to be a convincing proof of the value of $F(x)$ for all values $F(x)$.) Hence, the OMA communication complexities of Disconnectivity$^{\mathrm{cc}}$ and Connectivity$^{\mathrm{cc}}$ may not be equal, and indeed our lower bound argument applies only to Disconnectivity$^{\mathrm{cc}}$.

described in the theorem hypothesis. As discussed in the outline of Section 1.1.2, our proof will use a reduction from the INDEX problem on $\binom{n}{2}$ inputs. In this problem, Alice's input consists of a bitstring $\mathbf{x}$ of length $\binom{n}{2}$, Bob's input is an index $i^* \in [\binom{n}{2}]$, and the goal is to output $\mathbf{x}_{i^*}$. It was established in [7] that any OMA protocol $\mathcal{Q}$ for INDEX on $\binom{n}{2}$ inputs requires $\text{hc}(\mathcal{Q}) \cdot \text{vc}(\mathcal{Q}) = \Omega(n^2)$. We show how to use $\mathcal{P}$ to construct a protocol $\mathcal{Q}$ for INDEX on $\binom{n}{2}$ inputs with $\text{hc}(\mathcal{Q}) = n + \text{hc}(\mathcal{P})$ and $\text{vc}(\mathcal{Q}) = \text{vc}(\mathcal{P})$. It will then follow from the lower bound for INDEX that $(\text{hc}(\mathcal{P}) + n) \cdot \text{vc}(\mathcal{P}) \geq n^2$.

Description of $\mathcal{Q}$. In $\mathcal{Q}$, Alice interprets her input $\mathbf{x} \in \{0,1\}^{\binom{n}{2}}$ as an undirected graph G_1 with n nodes as follows. She associates each index $i \in \binom{n}{2}$ with a unique edge (u_i, v_i) out of the set of all $\binom{n}{2}$ possible edges that could appear in G_1. Alice also adds to G_1 a special node v^*, and connects v^* to every other node in G_1. Denote the resulting graph on $n+1$ nodes by G_2. Notice that G_2 is always connected, as every node is connected to v^* by design.

Likewise, Bob interprets his input $i^* \in [\binom{n}{2}]$ as an edge (u_{i^*}, v_{i^*}). Clearly, determining whether $\mathbf{x}_{i^*} = 1$ is equivalent to determining whether edge (u_{i^*}, v_{i^*}) appears in Alice's graph G_2. Merlin sends Bob a list L claimed to equal all edges incident to node u_{i^*} in G_2. This requires only n bits of "help", since there are only n nodes to which u_{i^*} might be adjacent. Bob treats L as his input to the DISCONNECTIVITY$^{\text{cc}}$problem.

Alice, Bob, and Merlin now run the DISCONNECTIVITY$^{\text{cc}}$ protocol $\mathcal{P}$ (with Alice's input equal to G_2 and Bob's input equal to L). Bob outputs 1 iff the protocol $\mathcal{P}$ outputs 1, and L contains the edge (u_{i^*}, v_{i^*}).

Costs of $\mathcal{Q}$. The help cost of $\mathcal{Q}$ is equal to $n + \text{hc}(\mathcal{P})$, since the honest Merlin sends Bob the list L, and then behaves as he would in the protocol $\mathcal{P}$. The verification cost of $\mathcal{Q}$ is just $\text{vc}(\mathcal{Q})$, since the only message Alice sends to Bob is the message she would send in $\mathcal{P}$.

Completeness and Soundness of $\mathcal{Q}$. Let G_3 denote the graph obtained from G_2 by XORing all the edges in the list L. Let $I(u_{i^*})$ denote the set of edges incident to u_{i^*} in G_3. We claim that G_3 is disconnected if and only if L is equal to $I(u_{i^*})$. For the first direction, suppose that L is equal to $I(u_{i^*})$. Then by XORing the edges in G_3 with the edges in L, every edge incident to node u_{i^*} is deleted from the graph. Hence, u_{i^*} is an isolated vertex in G_3, implying that G_3 is disconnected. For the second direction, suppose that L is not equal to $I(u_{i^*})$. Let (u_{i^*}, v) denote an edge in $L \setminus I(u_{i^*})$. Then (u_{i^*}, v) is in G_3. Moreover, v is adjacent to node v^*, as are all nodes in G_3 other than u_{i^*}. Hence G_3 is connected.

To finish the proof of completeness of $\mathcal{Q}$, note that if $\mathbf{x}_{i^*} = 1$, then the edge (u_{i^*}, v_{i^*}) is in G_3. If Merlin sends $L = I(u_{i^*})$, then G_3 will be disconnected, and by the the completeness of $\mathcal{P}$, Merlin can convince Bob that G_3 is disconnected with probability at least 2/3. In this event, Bob will output 1, because (u_{i^*}, v_{i^*}) will be in the list L.

To finish the proof of soundness of $\mathcal{Q}$, note that if $\mathbf{x}_{i^*} = 0$, then the edge (u_{i^*}, v_{i^*}) is not in G_3. Hence, if Merlin sends $L = I(u_{i^*})$, then Bob will reject automatically, because (u_{i^*}, v_{i^*}) will not be in the list L. On the other hand, if Merlin sends a list L that is *not* equal to $I(u_{i^*})$, then G_3 will be connected. By the the soundness of $\mathcal{P}$, Merlin can convince Bob that G_3 is disconnected with probability at most 1/3. Hence, Bob will output 1 in $\mathcal{Q}$ with probability at most 1/3, completing the proof for DISCONNECTIVITY$^{\text{cc}}$. ◀

Because any online scheme for CONNECTIVITY or BIPARTITENESS can be simulated by an OMA communication protocol, we obtain the following corollary.

▶ **Corollary 7** (Formal Version of Theorem 3). *Consider any online scheme for* Connectivity *or* Bipartiteness *in the XOR update model with help cost c_a and and space cost c_v. Then $(c_a + n) \cdot c_v \geq n^2$, even under the promise that the first $m - n$ stream updates are all unique, and the last n stream updates are all incident to a single node. In particular, the total cost of any annotation scheme is $\Omega(n)$.*

Acknowledgments. The author is grateful to Graham Cormode, Amit Chakrabarti, Andrew McGregor, and Suresh Venkatasubramanian for many valuable discussions regarding this work.

References

1. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *SODA*, pages 459–467, 2012. URL: http://dl.acm.org/citation.cfm?id=2095156.
2. Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14, 2012. doi:10.1145/2213556.2213560.
3. Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
4. László Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985. doi:10.1145/22145.22192.
5. Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545464.
6. Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In *SODA*, pages 687–706, 2014. doi:10.1137/1.9781611973402.52.
7. Amit Chakrabarti, Graham Cormode, Andrew McGregor, and Justin Thaler. Annotations in data streams. *ACM Trans. Algorithms*, 11(1):7:1–7:30, 2014. doi:10.1145/2636924.
8. Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and arthur-merlin communication. In *30th Conference on Computational Complexity, CCC 2015, June 17-19, 2015, Portland, Oregon, USA*, pages 217–243, 2015. doi:10.4230/LIPIcs.CCC.2015.217.
9. Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In *CRYPTO*, pages 151–168, 2011. doi:10.1007/978-3-642-22792-9_9.
10. Graham Cormode and Donatella Firmani. A unifying framework for l_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014. doi:10.1007/s10619-013-7131-9.
11. Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, pages 90–112, 2012. doi:10.1145/2090236.2090245.
12. Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013. doi:10.1007/s00453-011-9598-y.
13. Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *PVLDB*, 5(1):25–36, 2011. URL: http://www.vldb.org/pvldb/vol5/p025_grahamcormode_vldb2012.pdf.
14. Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012. URL: http://arxiv.org/abs/1203.4900.

15 Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, STOC'08, pages 113–122, New York, NY, USA, 2008. ACM. doi:10.1145/1374376.1374396.

16 Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. doi:10.1137/0218012.

17 Michael T. Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *Allerton*, pages 792–799, 2011. doi:10.1109/Allerton.2011.6120248.

18 Tom Gur and Ran Raz. Arthur-Merlin streaming complexity. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming: Part I*, ICALP'13, Berlin, Heidelberg, 2013. Springer-Verlag.

19 Hartmut Klauck and Ved Prakash. Streaming computations with a loquacious prover. In *ITCS*, pages 305–320, 2013. doi:10.1145/2422436.2422471.

20 Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem. In *ICALP (1)*, pages 919–930, 2014. doi:10.1007/978-3-662-43948-7_76.

21 Troy Lee, Frédéric Magniez, and Miklos Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *SODA*, pages 1486–1502, 2013. doi:10.1137/1.9781611973105.107.

22 List of open problems in sublinear algorithms: Problem 47. http://sublinear.info/47.

23 Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39:859–868, October 1992. doi:10.1145/146585.146605.

24 Andrew McGregor. Graph mining on streams. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 1271–1275. Springer US, 2009. doi:10.1007/978-0-387-39940-9_184.

25 Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, May 2014. doi:10.1145/2627692.2627694.

26 S. Muthukrishnan. *Data Streams: Algorithms And Applications*. Foundations and Trends in Theoretical Computer Science. Now Publishers Incorporated, 2005.

27 Charalampos Papamanthou, Elaine Shi, Roberto Tamassia, and Ke Yi. Streaming authenticated data structures. In *EUROCRYPT*, pages 353–370, 2013. doi:10.1007/978-3-642-38348-9_22.

28 A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow.*, 6(14):1870–1881, September 2013. doi:10.14778/2556549.2556569.

29 Adi Shamir. IP = PSPACE. *J. ACM*, 39:869–877, October 1992. doi:10.1145/146585.146609.

30 Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pages 607–614, 2011. doi:10.1145/1963405.1963491.

31 Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO (2)*, pages 71–89, 2013. doi:10.1007/978-3-642-40084-1_5.