Semi-Streaming Algorithms for Annotated Graph Streams

Justin Thaler*

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 $O(n\,\mathrm{polylog}(n))$.

In the annotated data streaming model of Chakrabarti et al. [CCMT14], 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 put forth 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 $O(n\,\mathrm{polylog}(n))$. We give evidence that semi-streaming annotation schemes represent a substantially 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 [ope]. 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 just as hard in the annotations model as they are in the standard model.

*Yahoo! Labs. 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.

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.

Several recent works have introduced closely related models capturing the above scenario [GR13,CCMT14, CCM$^+$13, KP13, CCGT14, KP14, CMT13, CTY11, CMT12]. Collectively, these works have begun to reveal a rich theory, leveraging algebraic techniques developed in the classical theory of interactive proofs [LFKN92, Sha92, Bab85, GMR89] 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. [CCMT14] (subsequently studied in [CMT13, KP13, CCGT14, CMT12]) 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 [GR13, CCGT14] and the *streaming interactive proofs* of Cormode et al. [CTY11, CCM$^+$13] 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 specifically 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 [Mut05, McG09]. The semi-streaming model is characterized by an $O(n\,\mathrm{polylog}\,n)$ 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 [AGM12a, AGM12b, GKP12]. We direct the interested reader toward the recent survey of McGregor [McG14] on graph stream algorithms.

In this work, we put forth 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 $O(n\,\mathrm{polylog}\,n)$.

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,

Reference	TRIANGLES Scheme Costs (help cost, space cost)	Total Cost Achieved
[CCMT14]	$(n^2 \log n, \log n)$	$O(n^2 \log n)$
[CCMT14]	$(x \log n, y \log n)$ for any $x \cdot y \geq n^3$	$O(n^{3/2} \log n)$
Theorem 1.1	$(n \log n, n \log n)$	$O(n \log n)$

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

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 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 "easy" problems (i.e., 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 specific graph problems that require $\Omega(n^2)$ space in the standard semi-streaming model: bipartite perfect matching [CCMT14], and shortest *s-t* path in graphs of polylogarithmic diameter [CMT13]. As discussed above, we give semi-streaming annotation schemes for two more challenging graph problems: maximum matching (MAXMATCHING) and counting triangles (TRIANGLES). Both schemes apply to dynamic graph streams.

Scheme for Counting Triangles.

Theorem 1.1 (Informal Version of Theorem 3.1)**.** *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 requires total cost* $\Omega(n)$*.*

Theorem 1.1 affirmatively answers a question of Cormode [ope], resolves the Merlin-Arthur communication complexity of the problem up to a logarithmic factor, and improves over the best previous upper bound of $O(n^{3/2} \log n)$, due to Chakrabarti et al. [CCMT14] (see Table 1 for a detailed comparison to prior work).

As is the case for essentially all non-trivial protocols for verifiable stream computation, the scheme of Theorem 1.1 uses algebraic techniques related to the famous *sum-check protocol* of Lund et al. [LFKN92] 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 [GKR08, CCM+13, CCMT14, Sha92]. Roughly

Reference	MAXMATCHING Scheme Costs (help cost, space cost)	Total Cost Achieved
[CMT13]	$(m \log n, \log n)$	$O(m \log n)$
Theorem 3.2	$(n \log n, n \log n)$	$O(n \log n)$

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

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. See Section 3.1.1 for further discussion of this point.

Like our scheme for MAXMATCHING below, our scheme for TRIANGLES does not achieve smooth tradeoffs between space and help costs: we do not know how to reduce the space usage to $o(n \log n)$ without blowing the help cost up to $\Omega(n^2)$, or vice versa. This is in contrast to prior work on annotated data streams [CCMT14, CMT13, CCGT14, GR13], which typically achieved any combination of space and help costs subject to the product of these two costs being above some threshold. We conjecture that achieving such smooth tradeoffs for either problem is impossible.

Scheme for Maximum Matching.

Theorem 1.2. *[Informal Version of Theorem 3.2] 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 requires 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. [CMT13], 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* [CCMT14, Theorem 7.5] – our scheme for MAXMATCHING can be seen as a broad generalization of [CCMT14, 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 1.3 (Informal Version of Corollary 4.2)**.** *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 [AGM12a].[1] Hence, Theorem 1.3 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 protocols for verifiable stream computation, our lower bounds are established using notions of *Merlin-Arthur communication complexity* [CCMT14,CMT13, KP13, CCM+13, GR13].

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 [CCGT14, Corollary 3.3]. The problem considered in [CCGT14, 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. Clearly, connectivity and bipartiteness are qualitatively very different from this problem, and arguably more natural.

Overview of the Proof. Our proof of Theorem 1.3 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 [CCMT14], 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 $O(n\,\text{polylog}(n))$ space. Yet Theorem 1.3 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 [CCGT14, Corollary 3.3], were all 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 [CMT12,CCM+13,CTY11,KP13,KP14,GR13,CCGT14,CCMT14, CMT13] have all studied annotated data streams and closely related models for verifiable stream computation. Refinements and implementations [VSBW13, CMT12, Tha13] 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 [CKLR11, PSTY13, SS12]. These works only require security against cheating provers that run in polynomial time, as compared to the setting we consider, where security holds even against computationally unbounded provers. In exchange for the weaker security guarantee, these protocols can achieve properties that are impossible 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. [CKLR11] uses fully homomorphic encryption (FHE), which remains far from practical at this time.

[1] The algorithms of [AGM12a] 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., [CF14]), and many sparse recovery algorithms can be implemented in the XOR update model directly (see, e.g., [GM11]).

Schröder and Schröder [SS12], and Papamanthou et al. [PSTY13] avoid the use of FHE, but handle only much simpler queries (such as 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. [CCGT14]. 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 $\text{out}^V(\mathbf{x}^{\mathfrak{h}}, r)$.

Online Schemes. 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 conditions 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).

Prescient Schemes. Chakrabarti et al. [CCMT14] 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 3.1 and 3.2 for details).

Additional Properties of Our Schemes. While the annotated data streams model allows the prover to in leave 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 for the verifier to retrieve at her convenience. 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 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. This kind of reusability is not possible with many interactive solutions [CTY11, CCM+13, KP14], which typically require the verifier to reveal information about r to the prover 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 $\text{poly}(n)$ takes one unit of time.

3.1 A Semi-Streaming Scheme for Counting Triangles

In the TRIANGLES problem, 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 [AYZ97] and there has been considerable interest in designing algorithms in a variety of models including the data stream model [BYKS02, PTTW13], MapReduce [SV11], and the quantum query model [LMS13]. 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. [CCMT14] 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 [ope] 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 answer this question in the affirmative.

Theorem 3.1 (Formal Statement of Theorem 1.1)**.** *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.*

Proof. The lower bound was proved in [CCMT14, 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

[2] The primary reason that we do not consider the case where edges may have negative weights at the end of the stream is that it is not clear what is the most meaningful way to define problems like TRIANGLES and MAXMATCHING in this setting. See Footnotes 3 and 4 for details.

[3] The TRIANGLES scheme 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.

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, at the end of the stream, the total number of triangles in the graph $G = G_m$ is precisely

$$\sum_{i\le m} \Delta_i \sum_{z\in[n]} E_i(u_i,z)E_i(v_i,z). \tag{1}$$

We use a novel variant of the sum-check protocol [LFKN92, AW09] to compute this quantity. To this end, let $\mathbb{F}$ denote a field of prime order $6(B\cdot n)^3 \le |\mathbb{F}| \le 12(B\cdot n)^3$, 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 Quantity (1) equals

$$\sum_{i\le m} \Delta_i \sum_{z\in[n]} E_i(u_i,z)E_i(v_i,z) = \sum_{i\le m} \Delta_i \sum_{z\in[n]} \tilde{E}_i(u_i,z)\tilde{E}_i(v_i,z) = \sum_{z\in[n]} \sum_{i\le m} \Delta_i \cdot \tilde{E}_i(u_i,z)\tilde{E}_i(v_i,z). \tag{2}$$

In turn, the right hand side of Equation (2) can be written as $\sum_{z\in[n]} g(z)$, where g denotes the *univariate* polynomial defined via:

$$g(Z) = \sum_{i\le m} \Delta_i \cdot \tilde{E}_i(u_i,Z)\tilde{E}_i(v_i,Z). \tag{3}$$

Notice $g(Z)$ is a univariate polynomial of degree at most $2n$. Our annotated data streaming protocol proceeds as follows.

Prover's computation. At the end of the stream, the prover is required to send 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. Equation (3) expresses $g(r)$ 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 $\sum_{j\le i}\Delta_j \cdot \tilde{E}_j(u_j,r)\tilde{E}_j(v_j,r)$ 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:

$$\tilde{\delta}_{(u,v)}(X,Y) = \left(\frac{\prod_{1\le u'\le n: u'\neq u}(X-u')}{\prod_{1\le u'\le n: u'\neq u}(u-u')}\right)\left(\frac{\prod_{1\le v'\le n: v'\neq v}(Y-v')}{\prod_{1\le v'\le n: v'\neq v}(v-v')}\right). \tag{4}$$

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 $\tilde{E}_i(X,Y) = \sum_{j\le i} \tilde{\delta}_{(u_j,v_j)}(X,Y)$. 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 in the final graph G.

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 and the verifier will reject. □

Several remarks regarding Theorem 3.1 are in order.

- **Verifier Time.** The verifier in the protocol of Theorem 3.1 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 $E_i(u,r)$ values. However, using Equation (4), 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: $E_i(u_i,r)$ and $E_i(v_i,r)$. We explain how both of these updates can be computed in constant time.

  It can be seen from Equation (4) that

  $$\tilde{\delta}_{(u_i,v_i)}(u_i,r) = \frac{\prod_{1 \leq v' \leq n : v' \neq v_i}(r - v')}{\prod_{1 \leq v' \leq n : v' \neq v_i}(v_i - v')} \tag{5}$$

  The right hand side of Equation (5) 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 $q_1(v) := \prod_{1 \leq v' \leq n : v' \neq v_i}(r - v')$ and $q_2(v) = \left(\prod_{1 \leq v' \leq n : v' \neq v_i}(v_i - v')\right)^{-1}$. 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 $q_2(v) = q_2(i-1) \cdot (v-1)^{-1} \cdot (n-v+1)$.

  Finally, the verifier can process the proof itself in time $O(n)$. Indeed, 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., [CMT12]).

- **Prover Time.** The honest prover in the protocol of Theorem 3.1 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). Note that this time is comparable to the cost of a 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 3.1 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. [CCMT14] via a standard reduction to SET-DISJOINTNESS, and answers a question of Cormode [ope].

3.1.1 Comparison to Prior Work

As is typical in the literature on interactive proofs, the verifier in our TRIANGLES protocol evaluates $g_\sigma(r)$ for a random point $r \in \mathbb{F}$, where g is a polynomial derived from the input stream σ. However, our protocol

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

Table 3: Statement of Time Costs For Our TRIANGLES Scheme (Theorem 3.1).

qualitatively departs from prior work in that $g_\sigma(r)$ is not a linear sketch of the input. Here we define a linear sketch as any summary of the form $\mathbf{v} \in \mathbb{F}^w$ for some $w > 0$ which can be computed as $\mathbf{v} = S\mathbf{f}(\sigma)$. Here, $S \in \mathbb{F}^{w \times n}$ is a "sketch matrix" and $\mathbf{f}(\sigma)$ denotes the *frequency-vector* of the stream, i.e., the ith element of $\mathbf{f}(\sigma)$ is the number of times item i appears in σ. To the best of our knowledge, all existing protocols in the interactive proofs literature only require the verifier to compute a linear sketch of the input [GKR08, Sha92, LFKN92, CCMT14, CTY11, CMT12, GR13, AW09]. Typically, this linear sketch consists of one or more evaluations of a *low-degree extension* of the frequency vector $\mathbf{f}$ itself.

In contrast, in our TRIANGLES protocol, we view the quantity $g_{i,\sigma}(r) := \sum_{j \leq i} \Delta_j \cdot \tilde{E}_j(u_j, r) \cdot \tilde{E}_j(v_j, r)$ as the verifier's sketch at time i. While $\tilde{E}_j(u, r)$ is a linear sketch of the input for each j and node $u \in [n]$ (in fact, this is what enables the verifier to compute $\tilde{E}_j(u, r)$ for each $u \in [n]$ in a streaming manner), $g_{i,\sigma}(r) = \sum_{j \leq i} \Delta_j \cdot \tilde{E}_j(u_j, r) \cdot \tilde{E}_j(v_j, r)$ is not. A consequence is that, in our TRIANGLES protocol, the verifier's final state $g_{m,\sigma}(r)$ at the end of the data stream *depends on the order of the stream stream tokens* — it is easy to construct dynamic graph streams σ, σ', such that σ is a permutation of σ', and yet $g_{m,\sigma}(r) \neq g_{m,\sigma'}(r)$ with high probability over the random choice of r.

This contrasts with linear sketches, as the final state of any linear sketch is independent of the order of the data stream. We conjecture that *any* semi-streaming scheme for counting triangles must have a "non-commuting" verifier, as in the protocol of Theorem 3.1. In contrast, recent work has shown that, in the standard streaming model, any "non-commutative" streaming algorithm that works in the turnstile streaming model can be simulated with essentially no increase space usage by a a linear sketching algorithm [LNW14].

3.1.2 Extensions: Counting Structures Other Than Triangles

Let H be a graph on k vertices. It is possible to extend the protocol underlying Theorem 3.1 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 $O(kn \log n)$ respectively. For concreteness, we describe the protocol in detail for the case where H is a 4-cycle.

Protocol for Counting 4-Cycles. As in the proof of Theorem 3.1, 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 four-cycles that e_i completes in E_i is precisely $\Delta_i \cdot \sum_{z_1, z_2 \in [n]} E_i(u_i, z_1) E_i(z_1, z_2) E_i(z_2, v_i)$. (Technically, this equality holds only assuming that there are no self-loops in G. However, the protocol is easily modified to handle graphs G (and sub-graphs H) that both may have self-loops). Thus, at the end of the stream, the total number of 4-cycles in the graph $G = G_m$ is precisely

$$\begin{aligned} &\sum_{i \leq m} \Delta_i \sum_{z_1, z_2 \in [n]} E_i(u_i, z_1) E_i(z_1, z_2) E_i(z_2, v_i) = \\ &\sum_{z_1, z_2 \in [n]} \sum_{i \leq m} \Delta_i E_i(u_i, z_1) E_i(z_1, z_2) E_i(z_2, v_i) = \\ &\sum_{z_1, z_2 \in [n]} \sum_{i \leq m} \Delta_i \tilde{E}_i(u_i, z_1) \tilde{E}_i(z_1, z_2) \tilde{E}_i(z_2, v_i). \end{aligned} \tag{6}$$

In Expression (6), the polynomial $\tilde{E}_i$ is defined exactly as in the proof of Theorem 3.1, except the field $\mathbb{F}$ over which $\tilde{E}_i$ is defined must have size $12(Bn)^4 \leq |\mathbb{F}| 24(Bn)^4$ (the reason for the larger field is simply to ensure

that the field is large enough to represent the maximum possible number of 4-cycles in the graph, without "wrap-around" issues). To compute Expression (6), the prover and verifier run the sum-check protocol of Lund et al. [LFKN92] — specifically, they apply the sum-check protocol to the bivariate polynomial

$$g(Z_1, Z_2) := \sum_{i \leq m} \Delta_i \tilde{E}_i(u_i, Z_1) \tilde{E}_i(Z_1, Z_2) \tilde{E}_i(Z_2, v_i).$$

We direct the interested reader to [AB09, Chapter 8] for a detailed description of the sum-check protocol. For our purposes, the crucial property of the sum-check protocol is the following: in order to run her part of the protocol, the verifier randomly chooses two values r_1, r_2 at random from $\mathbb{F}$, and needs to evaluate the quantity $g(r_1, r_2) = \sum_{i \leq m} \Delta_i \tilde{E}_i(u_i, r_1) \tilde{E}_i(r_1, r_2) \tilde{E}_i(r_2, v_i)$. The verifier can do this in space $O(n \log |\mathbb{F}|)$ with a single streaming pass over the input, using techniques identical to those used in Theorem 3.1.

Summary of Costs. The sum-check protocol applied a v-variate polynomial $g(Z_1, \ldots, Z_v)$ requires v rounds of communication (more precisely, it requires the prover to send a total of v messages to the verifier, and the verifier to send a total of $v-1$ responses to the prover). The verifiers ith message to the prover consists of the single field element $r_i \in \mathbb{F}$, while the provers ith message to the verifier is univariate polynomial $s_i(Z_i)$ whose degree is at most $\deg_i(g)$, the degree of g in the variable Z_i.

In our context, this translates to a 3-message (2-round) protocol for counting 4-cycles, in which the total communication cost is $O(n \log n)$. The soundness parameter δ_s of the protocol is at most $\sum_i \deg_i(g)/|\mathbb{F}| \leq 4n/|\mathbb{F}| \leq B^4/(3n^3)$.

More generally, given any k-node subgraph H, we can count the number of occurrences H in G by using a suitable $(k-2)$-variate polynomial g. Applying the sum-check protocol to g requires $2k-1$ messages (spread over k rounds) between the prover and verifier. The space usage of the verifier to evaluate g at a random point becomes $O(kn \log n)$, and the total communication cost is $O(k^3 n \log n)$. Here, the factor of k in the verifier's space usage is due to the need to work over a field whose size grows exponentially in k. The factor of k^3 in the communication complexity is due to the following three sources: one factor of k is due to the need to work in a field whose size grows exponentially with k, another factor is due to the fact that $\deg_i(g)$ grows linearly with k, for each variable Z_i, and the final factor of k is due to the fact that the number of messages sent by the prover grows linearly with k.

3.2 A Semi-Streaming Scheme for Maximum Matching

We give a semi-streaming scheme for the MAXMATCHING problem in general graphs. Our scheme combines the Tutte-Berge formula with algebraic techniques to allow the prover to prove matching upper and lower bounds on the size of a maximum matching in the input graph.[4]

Theorem 3.2. *[Formal Version of Theorem 1.2] 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 requires total cost $\Omega(n)$, even for $B = 1$.*

Proof. The lower bound follows from an identical lower bound proved for the Bipartite Perfect Matching problem by Chakrabarti et al. [CCMT14], combined with the fact that the MAXMATCHING problem is at least as hard as Bipartite Perfect Matching. We now turn to the upper bound. We begin with a high-level proof sketch that highlights the novel aspects of our scheme, before turning to a detailed scheme description.

Proof Sketch. The prover proves matching upper and lower bounds on the size of a maximum matching. To establish a lower bound, the prover simply sends a set of edges, M, claimed to be a maximum matching,

[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.

and uses techniques from prior work [CCMT14] to establish that $M \subseteq E$. To establish an upper bound, we exploit the Tutte-Berge formula, which states that the size of a maximum matching in $G = (V,E)$ is equal to

$$\frac{1}{2}\min_{U \subseteq V}\left(|U| - odd(G-U) + |V|\right). \tag{7}$$

Here, $G-U$ is the induced graph obtained from G by removing all vertices in U, and $odd(G-U)$ denotes the number of connected components with an odd number of vertices in $G-U$. The prover sends a set $U^* \subseteq V$ claimed to achieve the minimum in Equation (7). The primary novelty in our scheme is a method for computing $odd(G-U^*)$ that (in contrast to the MAXMATCHING scheme of [CMT13]) avoids the need for the prover to replay all of the edges that appeared in the stream.

Detailed Scheme Description. Suppose that the prover claims that the answer is k. Our scheme consists of two parts: in the first part, the prover proves that the size of a maximum matching in G is *at least* k. In the second part, the prover proves that the size of a maximum matching in G is *at most* k. We clarify that the verifier will be able to perform the required processing for both parts of the scheme simultaneously with a single pass over the input stream, and the prover can send the annotation for both parts of the scheme in a single message (i.e., by simply concatenating the annotation for the two parts). If the prover successfully passes all of the verifier's checks in both parts, then the verifier is convinced that the size of a maximum matching in G is exactly k.

Part One. This part of the scheme is similar to the analogous part of the bipartite perfect matching protocol given in [CCMT14]: the prover sends a set M of k edges $M = (e^{(1)}, \ldots, e^{(k)})$ that are claimed to comprise a valid matching in G, and the verifier explicitly stores M (this requires at most $n \log n$ bits of annotation and space). The verifier needs to check that M is indeed a valid matching for G. This requires checking two properties:

- **Property 1:** For all nodes $v \in V$, v is incident to at most one edge $e^{(i)} \in M$.
- **Property 2:** $M \subseteq E$.

Property 1 is trivial to check, because the verifier stores M explicitly. Property 2 can be checked using a scheme for the SUBSET problem described in [CCMT14, Lemma 5.3]. Here, the input to the SUBSET problem consists of two (multi-)sets, S_1 and S_2, over a data universe $\mathcal{U}$, arbitrarily interleaved, and the output is 1 if and only if $S_1 \subseteq S_2$. We apply the SUBSET scheme of Chakrabarti et al. with $S_1 = M$, $S_2 = E$, and $\mathcal{U}$ equal to the set of all $O(n^2)$ possible edges. The scheme requires help cost $O(Bx \log n)$ and space cost $O(By \log n)$ for any $x \cdot y \geq |\mathcal{U}| = O(n^2)$, where B is an a prior upper bound on the multiplicity of any edge in E. In particular, assuming that $B = O(1)$, both the help and space costs of the SUBSET scheme can be set to $O(n \log n)$.

Part Two. We exploit the Tutte-Berge formula. This formula states that the size of a maximum matching in a graph $G = (V,E)$ is equal to

$$\frac{1}{2}\min_{U \subseteq V}\left(|U| - odd(G-U) + |V|\right).$$

Here, $G-U$ is the induced graph obtained from G by removing all vertices in U, and $odd(G-U)$ denotes the number of connected components with an odd number of vertices in $G-U$.

Thus, to establish that the size of the a maximum matching in G is at most k, the prover first identifies a subset $U^* \subseteq V$ such that $k = g(U^*)$, where

$$g(U^*) := \frac{1}{2}\left(|U^*| - odd(G-U^*) + |V|\right).$$

The prover sends U^* to V, which requires $O(n \log n)$ bits of help, and the verifier stores U^* explicitly, which requires $O(n \log n)$ bits of space. Clearly the verifier can easily compute the quantity $g(U^*)$ if she knows the value $odd(G - U^*)$. The remainder of the scheme is therefore devoted to computing $odd(G - U^*)$.

A (Sub-)Scheme for Computing $odd(G - U^*)$. The prover first assigns each connected component in $G - U^*$ a unique label in $[C]$ arbitrarily, where C is the number of connected components in $G - U^*$. The prover then sends a list L to the verifier that contains a pair (v, ℓ_v) for each node $v \in V \setminus U^*$, where ℓ_v is claimed to equal the label of v's connected component in $G - U^*$. The verifier stores the list L explicitly — the help and space costs for the prover to send L and the verifier to store L are both $O(n \log n)$. Since the verifier stores L and U^* explicitly, it is trivial for the verifier to check that every node in $V \setminus U^*$ appears exactly once in L.

If the list L is as claimed, then it is easy for the verifier to compute $odd(G - U^*)$ in $O(n \log n)$ space, as the verifier can simply count of the number of labels that appear in L an odd number of times. The remainder of the scheme is therefore devoted to ensuring that the list L is as claimed.

To this end, the verifier must ensure that the labels ℓ_v in L actually correspond to the connected components of $G - U^*$. This requires checking two properties.

- **Property A:** For each label ℓ, all nodes v with label $\ell_v = \ell$ are connected to each other in $G - U^*$.
- **Property B:** For every pair of nodes u, v in $V \setminus U^*$ with different labels $\ell_u \neq \ell_v$, it holds that $(u, v) \notin E$. That is, there is no edge in $G - U^*$ connecting two nodes that are claimed to be in different connected components.

To prove that Property A holds, the prover sends, for each connected component ℓ in $G - U^*$, a set of edges $T_\ell \subseteq (V \setminus U^*) \times (V \setminus U^*)$ that is claimed to be a spanning tree for all nodes with label ℓ (and the verifier explicitly stores each of the T_ℓ's). Note that sending and storing the T_ℓ's requires $O(n \log n)$ bits of help and space in total. For each ℓ, the verifier must check that T_ℓ spans all nodes with label ℓ in L, and that $\cup_\ell T_\ell \subseteq E$.

Checking that T_ℓ spans all nodes with label ℓ is trivial, as the verifier has explicitly stored the list L of (vertex, label) pairs, as well as the trees T_ℓ. To check that $\cup_\ell T_\ell \subseteq E$, we use the SUBSET scheme of [CCMT14, Lemma 5.3].

A (Sub-)Scheme for Checking that Property B holds. Checking that Property B holds requires more care. Notice that Property B holds if and only if:

$$0 = \sum_{(u,v) \in [n] \times [n]} D(u, v) \cdot E(u, v). \tag{8}$$

Here, $D(u, v) \colon [n] \times [n] \to \{0, 1\}$ is defined to be the function that outputs 1 if $\ell_u \neq \ell_v$, and is 0 otherwise (to clarify, if either u or v is in U^* itself, then $D(u, v)$ is defined to be 0) — we choose the letter D because such edges are *disallowed*, if the component labels provided by the prover are correct. Abusing notation, $E(u, v) \colon [n] \times [n] \to \{0, 1\}$ in Equation (8) denotes the function that on input (u, v), outputs the multiplicity of edge (u, v) in E.

To check that Equation (8) holds, we let $\tilde{D}$ denote the unique bivariate polynomial over a field $\mathbb{F}$ of prime order, $2n^3 \leq |\mathbb{F}| \leq 4n^3$, such that the degree of $\tilde{D}$ is at most n in each variable, and $\tilde{D}(u, v) = D(u, v)$ for all $(u, v) \in [n] \times [n]$. Likewise, we let $\tilde{E}$ denote the unique bivariate polynomial over $\mathbb{F}$ such that the degree of $\tilde{E}$ is at most n in each variable, and $\tilde{E}(u, v) = E(u, v)$ for all $(u, v) \in [n] \times [n]$.

Prover's Computation. After the stream has passed and the prover has sent the list L, the prover is required to send a univariate polynomial $s(Y)$ of degree at most $2n$ claimed to equal

$$g(Y) := \sum_{u \in [n]} \tilde{D}(u, Y) \cdot \tilde{E}(u, Y).$$

Because of the degree bound on $s(Y)$, this can be done with 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 $r \in \mathbb{F}$. We will explain in the next two paragraphs how the verifier can evaluate $g(r)$ in a streaming fashion, with space cost $O(n \log n)$. The verifier then checks that $s(r) = g(r)$. If this check fails, then the verifier halts and rejects. If the check passes, then verifier checks that $\sum_{v=1}^{n} s(i) = 0$. If so, the verifier is convinced that Property B holds, and accepts k as an upper bound on the size of any maximum matching in G.

While observing the data stream, the verifier incrementally computes the n values $\tilde{E}(u,r)$ for each $u \in [n]$. The proof of Theorem 3.1 explained how the verifier can do this in a streaming manner, using $O(\log |\mathbb{F}|) = O(\log n)$ space for each value $u \in [n]$, and thus $O(n \log n)$ space in total. Similarly, after the prover has sent U^* and L, the verifier computes the n values $\tilde{D}(u,r)$ for each $u \in [n]$. Notice that the function $\tilde{D}$ is uniquely determined by U^* and the list L of (vertex, label) pairs. Since the verifier has explicitly stored U^* and L, it is straightforward for the verifier to evaluate each of these n values with total space $O(n \log |\mathbb{F}|) = O(n \log n)$ (in the remarks following the theorem, we also explain in detail how the verifier can perform these n evaluations in $O(n)$ total time).

Once the verifier has computed the values $\tilde{E}(u,r)$ and $\tilde{D}(u,r)$ for each $u \in [n]$, it is straightforward for the verifier to compute $g(r) = \sum_{u \in [n]} \tilde{E}(u,r) \cdot \tilde{D}(u,r)$.

Soundness and Completeness of the (Sub-)Scheme for Property B. The proofs of soundness and completeness are essentially identical to Theorem 3.1. If Equation (8) holds and $s = g$, then the verifier's checks will pass with probability 1, and the verifier will accept k as an upper bound on the size of any maximum matching in G. If Equation (8) is false, then the prover is forced to send a polynomial $s \neq g$, or the verifier's final check that $\sum_{v=1}^{n} s(v) = 0$ will fail. But if $s \neq g$, then $s(r) \neq g(r)$ with probability at least $1 - 2n/|\mathbb{F}| \geq 1 - 1/n$, because any two distinct polynomials of degree at most $2n$ can agree on at most $2n$ inputs. □

Several remarks are in order regarding Theorem 3.2.

- **The Bottleneck in Reducing Space Costs.** In a formal sense, the most difficult part of the MAXMATCHING scheme given in Theorem 3.2 is checking that Property B holds. Indeed, this is the only part of the entire MAXMATCHING scheme for which we are presently unable to reduce the verifier's space usage to $o(n)$ without increasing the help cost to $\Omega(n^2)$. That is, if it were possible for the verifier to *avoid* checking Property B, we would be able to give a scheme with help cost $O(x \log n)$ and space cost $O(y \log n)$ for any pair x, y such that $x \cdot y \geq n^2$ and $x \geq n$ (in contrast to Theorem 3.2, which only gives a scheme for the special values $x = y = n$). We conjecture, however, that the difficulty is inherent, i.e., that there is no way to reduce the space cost to $o(n)$ without increasing the help cost to $\Omega(n^2)$.

- **Prover Time.** In the scheme of Theorem 3.2, the prover has to compute a maximum matching and identify the (possibly non-unique) set U^* whose existence is guaranteed by the Tutte-Berge formula. Fortunately, standard algorithms for computing maximum matchings identify the set U^* as a natural byproduct of their execution.[5] Using Fast Fourier Transform techniques described in [CMT12], the remainder of the prover's computation in the scheme of Theorem 3.2 can be executed in time $O(n^2 \log n)$. Hence, the total runtime of the prover is $O(T + n^2 \log n)$, where T is the time required to find a maximum matching along with the Tutte-Berge decomposition U^*.

[5] See for example the Boost Graph Library, `http://www.boost.org/doc/libs/1_45_0/libs/graph/doc/maximum_matching.html`.

Verfier Pre-Processing Time	Verifier Time Per Stream Update	Verifier Time to Process Proof	Prover Total Time
$O(n \log n)$	$O(1)$	$O(n)$	$O(T + n^2 \log n)$

Table 4: Statement of time costs for our MAXMATCHING scheme (Theorem 3.2). T denotes the time required to find a maximum matching, as well as the Tutte-Berge decomposition of G.

- **Verifier Time.** Recall that we explained in Section 3.1 how to implement a verifier running in constant time per stream update, after a pre-processing stage requiring $O(n)$ time (see the remarks following Theorem 3.1). These techniques are easily modified to enable the verifier in the scheme of Theorem 3.2 to also run in constant time per stream update, after a pre-processing stage requiring $O(n \log n)$ time.

  Finally, we explain that the verifier can process the proof in $O(n)$ time. Indeed, when processing the proof, the verifier's tasks fall into three categories: (1) run two SUBSET protocols involving the sets M and $\cup_\ell T_\ell$ provided by the prover (note both sets are of size at most n), (2) Evaluate $\tilde{D}(u,r)$ for each $u \in [n]$ as part of the (sub-)scheme to check that Property B holds, and (3) perform various other checks on the proof that require $O(n)$ time in total by inspection.

  The verifier's computation for the two SUBSET schemes on the sets M and $\cup_\ell T_\ell$, both of size at most n, can easily be implemented in $O(n)$ total time using the same techniques as above the ensure that the verifier processes each element of M and $\cup_\ell T_\ell$ in constant time — we omit the straightforward details for brevity. Evaluating $\tilde{D}(u,r)$ for each $u \in [n]$ in total time $O(n)$ can be done in a similar fashion, but requires more care. It is straightforward to check that

$$\tilde{D}(u,r) = \sum_{v \in V \setminus U^* : \ell_v \neq \ell_u} \tilde{\delta}_v(r), \tag{9}$$

  where

$$\tilde{\delta}_v(X) = \frac{\prod_{1 \leq v' \leq n, v \neq v'} (r - v')}{\prod_{1 \leq v' \leq n, v \neq v'} (v - v')}.$$

  Using the techniques described in Section 3.1, the verifier can store and compute the values $\tilde{\delta}_v(r)$ for all $v \in [n]$ in pre-processing, in $O(n \log n)$ total time. The verifier also computes and stores the value $H := \sum_{v \in V \setminus U^*} \tilde{\delta}_v(r)$ in $O(n)$ time.

  With these values in hand, the verifier can compute all n values $\tilde{D}(u,r) : u \in [n]$ in $O(n)$ total time. To see this, notice that Equation (9) implies that $\tilde{D}(u,r)$ depends only on the label ℓ_u of the connected component of u in $G - U^*$. That is, $\tilde{D}(u,r) = \tilde{D}(u',r)$ for all u' such that $\ell_u = \ell_{u'}$. Moreover, letting $S(\ell)$ denote the set of all vertices v with $\ell_v = \ell$, $\tilde{D}(u,r)$ can be computed in time $O(|S(\ell_u)|)$, given the precomputed values described above, via the following identity that follows from Equation (9):

$$\tilde{D}(u,r) = H - \sum_{v : \ell_v = \ell_u} \tilde{\delta}_v(r).$$

  Hence, *all* of the $\tilde{D}(u,r)$ values can be computed in total time $\sum_{\text{distinct labels } \ell_u} O(|S(\ell_u)|) = O(n)$.

  The prover and verifier's time costs in the MAXMATCHING scheme of Theorem 3.2 are summarized in Table 4.

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.

4.1 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 the value $\mathbf{Val}^{\mathcal{P}}(x,y)$ to be Merlin's probability of winning this game with optimal play. Given a Boolean function f, we say that $\mathcal{P}$ computes f if, for all (x,y) we have (1) $f(x,y) = 0 \implies \mathbf{Val}^{\mathcal{P}}(x,y) \leq 1/3$, and (2) $f(x,y) = 1 \implies \mathbf{Val}^{\mathcal{P}}(x,y) \geq 2/3$. We refer to the Property (1) as *soundness* and Property (2) as *completeness*. Notice that the completeness requirement is qualitatively weaker than what we require for *schemes*: in a scheme, we required that there exist a prover strategy that can convince the verifier of the value of $f(x)$ for *all* x, while in an MA communication protocol we only require this to hold for $x \in f^{-1}(1)$.

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 communication protocol (OMA protocol for short), neither Merlin nor Bob can talk to Alice. This condition models the one-pass streaming restriction on the verifier in an online scheme. Indeed, 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 $x \circ y$. The help cost of $\mathcal{P}$ is equal to the help cost of the scheme, while the verification cost of $\mathcal{P}$ is equal to the space cost of the scheme. Hence, if we establish lower bounds on the help and verification costs of any OMA protocol for f, we may conclude an equivalent lower bound on the help and space costs of any online scheme for f.

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.

4.2 The Lower Bound

Theorem 4.1. *Consider any OMA protocol $\mathcal{P}$ for* DISCONNECTIVITY^{cc} *or* BIPARTITENESS^{cc}. *Then*

$$(\mathrm{hc}(\mathcal{P}) + n) \cdot \mathrm{vc}(\mathcal{P}) = \Omega(n^2).$$

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 begin with the DISCONNECTIVITY^{cc} problem. Let $\mathcal{P}$ denote any OMA protocol for DISCONNECTIVITY^{cc} that works on graphs with $n+1$ nodes, under the promise 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 [CCMT14] that any OMA protocol $\mathcal{Q}$ for INDEX on $\binom{n}{2}$ inputs requires

$$\mathrm{hc}(\mathcal{Q}) \cdot \mathrm{vc}(\mathcal{Q}) = \Omega(n^2). \tag{10}$$

We show how to use $\mathcal{P}$ to construct a protocol $\mathcal{Q}$ for INDEX on $\binom{n}{2}$ inputs with $\mathrm{hc}(\mathcal{Q}) = n + \mathrm{hc}(\mathcal{P})$ and $\mathrm{vc}(\mathcal{Q}) = \mathrm{vc}(\mathcal{P})$. It will then follow from Equation (10) that $(\mathrm{hc}(\mathcal{P}) + n) \cdot \mathrm{vc}(\mathcal{P}) \geq n^2$ as claimed.

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 Alice connects v^* to every other node v 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^{cc} problem.

Alice, Bob, and Merlin now run the DISCONNECTIVITY^{cc} protocol $\mathcal{P}$ (with Alice's input equal to G_2 and Bob's input equal to L). Bob outputs 1 if and only if 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 $n + \mathrm{hc}(\mathcal{P})$, 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 $\mathrm{vc}(\mathcal{Q})$, 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 the graph 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 complete 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 complete 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^{cc}.

Proof for BIPARTITENESS^{cc}. The proof for the BIPARTITENESS^{cc} problem follows a similar high-level outline. Let $\mathcal{P}'$ be an online MA protocol for BIPARTITENESS^{cc} on graphs with $n+1$ nodes. We show how to use $\mathcal{P}'$ to to construct a protocol $\mathcal{Q}'$ for INDEX on $(n/2) \cdot (n/2)$ inputs with $\mathrm{hc}(\mathcal{Q}') = n + \mathrm{hc}(\mathcal{P}')$ and $\mathrm{vc}(\mathcal{Q}') = \mathrm{vc}(\mathcal{P}')$ (we assume that n is even for simplicity). It will then follow from Equation (10) that $(\mathrm{hc}(\mathcal{P}') + n) \cdot \mathrm{vc}(\mathcal{P}') = \Omega(n^2)$ as claimed.

Description of $\mathcal{Q}'$. In $\mathcal{Q}'$, Alice interprets her input $\mathbf{x} \in \{0,1\}^{n^2/4}$ as an undirected bipartite graph G'_1 with $n/2$ nodes on the left and $n/2$ nodes on the right, as follows. She associates each index $i \in [n^2/4]$ with a unique edge (u_i, v_i) out of the set of all $n^2/4$ possible edges that could appear in the bipartite graph G'_1. Alice also adds to G'_1 a special node v^*, and Alice connects v^* to all $n/2$ nodes on the right side of G'_1. Denote the resulting graph on $n+1$ nodes by G'_2. Notice that G'_2 is always bipartite by design. Indeed, letting S_1 denote the set of all nodes on the left side of G'_1, and S_2 denote the set of all nodes on the right side of G_1, then every edge in G'_2 connects a vertex in $S_1 \cup \{v^*\}$ to one in S_2.

Likewise, Bob interprets his input $i^* \in [n^2/4]$ 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 only requires n bits of "help", since there are only n nodes that u_{i^*} might be adjacent to. Bob treats $L \cup \{(u_{i^*}, v^*)\}$ as his input to the BIPARTITENESS$^{\text{cc}}$ problem.

Alice, Bob, and Merlin now run the BIPARTITENESS$^{\text{cc}}$ protocol $\mathcal{P}'$ (with Alice's input equal to G'_2 and Bob's input equal to $L \cup \{(u_{i^*}, v^*)\}$). Bob outputs 1 if and only if the protocol $\mathcal{P}'$ outputs 1, and L contains the edge (u_{i^*}, v_{i^*}).

Costs of $\mathcal{Q}'$. Analogous to the case of DISCONNECTIVITY$^{\text{cc}}$, 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 $L \cup \{(u_{i^*}, v^*)\}$. Let $I(u_{i^*})$ denote the set of edges incident to u_{i^*} in G'_3. We claim that G'_3 is bipartite 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 $L \cup \{(u_{i^*}, v^*)\}$, every edge incident to node u_{i^*} in G'_2 is deleted from the graph, and edge $\{(u_{i^*}, v^*)\}$ is inserted. It follows that G'_3 is bipartite. Indeed, recall that S_1 denotes the set of all nodes on the left side of G'_1, and S_2 denotes the set of all nodes on the right side of G'_1. Then every edge in G'_3 connects a vertex in $(S_1 \setminus \{u_{i^*}\}) \cup \{v^*\}$ to a node in $S_2 \cup \{u_{i^*}\}$. That is, if L is equal to $I(u_{i^*})$, then the bipartition of G'_3 is identical to that of G'_2, except that u_{i^*} has switched from the left hand side of G'_2 to the right hand side of G'_3.

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 the graph G'_3. Moreover, v is adjacent to node v^* in G'_3, as is u_{i^*}. Hence, there is a triangle in G'_3, so G'_3 cannot be bipartite.

To complete the proof of completeness of $\mathcal{Q}'$, note that if $\mathbf{x}_{i^*} = 1$, then (u_{i^*}, v_{i^*}) is in G'_3. If Merlin sends $L = I(u_{i^*})$, then G'_3 will be bipartite, and by the the completeness of $\mathcal{P}'$, Merlin can convince Bob that G'_3 is indeed bipartite with probability at least $2/3$. In this event, Bob will output 1 in $\mathcal{Q}'$, because (u_{i^*}, v_{i^*}) will be in the list L.

To complete the proof of soundness of $\mathcal{Q}'$, note that if $\mathbf{x}_{i^*} = 0$, then (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 non-bipartite. By the the soundness of $\mathcal{P}'$, Merlin can convince Bob that G'_3 is bipartite with probability at most $1/3$. Hence, Bob will output 1 in $\mathcal{Q}'$ with probability at most $1/3$, completing the proof for BIPARTITENESS$^{\text{cc}}$. □

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

Corollary 4.2 (Formal Version of Theorem 1.3)**.** *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 for these problems is $\Omega(n)$.*

5 Open Questions

A number of questions regarding annotated data streams in general, and semi-streaming annotation schemes in particular, remain open. Here, we highlight five.

- Exhibit any explicit function for which all online schemes require total cost asymptotically larger than the square root of the input size. In particular, it is open to exhibit any graph problem that *cannot* be solved by an online semi-streaming scheme.
- Do there exist schemes of total cost $o(n^2)$ for any of the following graph problems: shortest *s*-*t* path in general graphs, diameter, and computing the value of a maximum flow. Note that a semi-streaming scheme is known for shortest *s*-*t* path in graphs of polylogarithmic diameter [CMT13], but not in general graphs (neither directed nor undirected).
- Do there exist schemes of total cost $o(n)$ for connectivity or bipartiteness in the strict turnstile update model? What about the insert-only update model? We conjecture that the answer is no in all cases.
- Is it possible to give a scheme for TRIANGLES or MAXMATCHING of space cost $o(n)$ and help cost $o(n^2)$? We conjecture that the answer is no for both problems.
- Is it possible to give a semi-streaming scheme for TRIANGLES with a "commutative" verifier? We conjecture that the answer is no. If true, this would contrast with the standard (sans prover) streaming model, where it is known that any streaming algorithm that works in the turnstile update model can be simulated by a linear sketch [LNW14].

Acknowledgments

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

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[AGM12a] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *SODA*, pages 459–467. SIAM, 2012.

[AGM12b] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *PODS*, pages 5–14. ACM, 2012.

[AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *ACM Trans. Comput. Theory*, 1(1):2:1–2:54, February 2009.

[AYZ97] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997.

[Bab85] László Babai. Trading group theory for randomness. In Robert Sedgewick, editor, *STOC*, pages 421–429. ACM, 1985.

[BYKS02] Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In David Eppstein, editor, *SODA*, pages 623–632. ACM/SIAM, 2002.

[CCGT14] Amit Chakrabarti, Graham Cormode, Navin Goyal, and Justin Thaler. Annotations for sparse data streams. In Chandra Chekuri, editor, *SODA*, pages 687–706. SIAM, 2014.

[CCM+13] Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. On interactivity in arthur-merlin communication and stream computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:180, 2013.

[CCMT14] Amit Chakrabarti, Graham Cormode, Andrew Mcgregor, and Justin Thaler. Annotations in data streams. *Preliminary Version in ICALP 2009. Journal Version to Appear in ACM Transactions on Algorithms*, 2014.

[CF14] Graham Cormode and Donatella Firmani. A unifying framework for ; 0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2011.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In Shafi Goldwasser, editor, *ITCS*, pages 90–112. ACM, 2012.

[CMT13] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013.

[CTY11] Graham Cormode, Justin Thaler, and Ke Yi. Verifying computations with streaming interactive proofs. *PVLDB*, 5(1):25–36, 2011.

[GKP12] Ashish Goel, Michael Kapralov, and Ian Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012.

[GKR08] 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.

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *Allerton*, pages 792–799. IEEE, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[GR13] 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.

[KP13] Hartmut Klauck and Ved Prakash. Streaming computations with a loquacious prover. In Robert D. Kleinberg, editor, *ITCS*, pages 305–320. ACM, 2013.

[KP14] Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP (1)*, volume 8572 of *Lecture Notes in Computer Science*, pages 919–930. Springer, 2014.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39:859–868, October 1992.

[LMS13] Troy Lee, Frédéric Magniez, and Miklos Santha. Improved quantum query algorithms for triangle finding and associativity testing. In Sanjeev Khanna, editor, *SODA*, pages 1486–1502. SIAM, 2013.

[LNW14] Yi Li, Huy L. Nguyen, and David Woodruff. Turnstile streaming algorithms might as well be linear sketches. In *STOC*, 2014.

[McG09] Andrew McGregor. Graph mining on streams. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 1271–1275. Springer, 2009.

[McG14] Andrew McGregor. Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20, May 2014.

[Mut05] S. Muthukrishnan. *Data Streams: Algorithms And Applications*. Foundations and Trends in Theoretical Computer Science. Now Publishers Incorporated, 2005.

[ope] List of open problems in sublinear algorithms: Problem 47. `http://sublinear.info/47`.

[PSTY13] Charalampos Papamanthou, Elaine Shi, Roberto Tamassia, and Ke Yi. Streaming authenticated data structures. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2013.

[PTTW13] 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.

[Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39:869–877, October 1992.

[SS12] Dominique Schröder and Heike Schröder. Verifiable data streaming. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 953–964. ACM, 2012.

[SV11] Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *WWW*, pages 607–614. ACM, 2011.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89. Springer, 2013.

[VSBW13] Victor Vu, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *IEEE Symposium on Security and Privacy*, 2013.