Streaming Verification of Graph Computations via Graph Structure

Amit Chakrabarti* Prantar Ghosh*

Abstract

We give new algorithms in the annotated data streaming setting—also known as verifiable data stream computation—for certain graph problems. This setting is meant to model outsourced computation, where a space-bounded verifier limited to sequential data access seeks to overcome its computational limitations by engaging a powerful prover, without needing to trust the prover. As is well established, several problems that admit no sublinear-space algorithms under traditional streaming do allow protocols using a sublinear amount of prover/verifier communication and sublinear-space verification. We give algorithms for many well-studied graph problems including triangle counting, its generalization to subgraph counting, maximum matching, problems about the existence (or not) of short paths, finding the shortest path between two vertices, and testing for an independent set. While some of these problems have been studied before, our results achieve new tradeoffs between space and communication costs that were hitherto unknown. In particular, two of our results disprove explicit conjectures of Thaler (ICALP, 2016) by giving triangle counting and maximum matching algorithms for n-vertex graphs, using $o(n)$ space and $o(n^2)$ communication.

1 Introduction

A major philosophical message of theoretical computer science is that a computationally bounded entity can greatly expand its space of tractable problems with access to a more powerful entity, *without* having to trust the latter. The celebrated IP = PSPACE [28] and PCP Theorems [3, 4] are perhaps the best known such results. In the realm of space-efficient computations on large data streams, there is a growing trend towards results of this flavor [26]. In this case, the powerful entity (henceforth named Prover) is often thought of as a cloud computing service that is free of the space limitations that the computationally bounded data streaming process (henceforth named Verifier) is subject to. This work designs new algorithms for graph computations on data streams in such Verifier/Prover models and proves some related complexity-theoretic results.

Early works on such "prover-enhanced data streaming algorithms" considered the *annotated streams* model [10, 22], where Prover reads the input data stream together with Verifier and, during stream processing and/or at the end, supplies Verifier with a *proof* (streamed to him) that convinces him of the correct answer to what he wants to compute on the stream. Subsequent works [11, 14] considered a more general model of *streaming interactive proofs* (SIPs), where the communication between Verifier and Prover is more general, rather than one way. Several recent works in the annotated stream and the SIP models have focused on basic algorithmic problems on graphs [2, 13, 29], often giving sublinear-space algorithms for problems that provably do not admit sublinear solutions in the basic (sans prover) streaming setting.

In this work, we give new algorithms in the annotated streaming setting for certain graph problems, including triangle counting, its generalization to subgraph counting, maximum matching, problems about the existence (or not) of short paths, finding the shortest path between two vertices, and testing for an independent set. Two of our results provide "unexpected" new upper bounds, disproving published conjectures [29] asserting that such bounds would be unattainable.

*Department of Computer Science, Dartmouth College. Email: {ac,prantarg}@cs.dartmouth.edu. Work supported in part by NSF under award CCF-1907738.

1.1 Our Results and Techniques

Background and Motivation. Suppose that we wish to compute a function $f(\sigma)$ on an input stream σ consisting of *tokens* from some universe. For instance, for a graph computation, σ could be a stream of vertex pairs (u, v) specifying the input graph's edges, or it could be a stream of edge insertions and/or deletions to an evolving (multi)graph. Following established terminology [10], an *online scheme* is a protocol between Prover and Verifier wherein they observe σ together and, after each token appears, Prover provides zero or more bits of "help" to Verifier (as specified by the protocol). After σ is fully consumed, if Prover has followed the protocol faithfully, Verifier is very likely to output $f(\sigma)$; otherwise, he is very likely to "reject." If Verifier does all his work using at most $O(v)$ bits of working memory and Prover sends at most $O(h)$ bits of help, we call this an (h, v)-*scheme*.[1] A scheme is interesting if we can use $h > 0$ to achieve a value of v asymptotically smaller than what is feasible or known for a basic streaming algorithm, where $h = 0$.

All interesting schemes from previous work in fact use the prover in a more restricted way: Verifier processes all of σ on his own and *then* interacts with Prover. This work continues the tradition. There is practical motivation for building this restriction into the model of computation. Think of a cloud computing service where compute cycles are available only at certain times of day, or need to be booked in advance, whereas the client needs to access and process the input earlier, when it is made available to him. In such a setting, a scheme is most useful if the client can do its own processing first and wait for its time slot with the cloud service to finalize its computation.

Further, we focus only on *schemes*, which feature a single streamed message from Prover to Verifier, rather than the more general setting of SIPs, which allow rounds of interaction. This too is practically motivated: the cloud service need not dedicate a chunk of time to interact with the client, but need only promise that it will perform its portion of the computation *by* an agreed-upon deadline, at which time the client will download the "proof" it has constructed. In view of this latter style of computation, we also consider *multi-pass schemes*, where Verifier may use a "few" passes over its input σ and later receive a single streamed message from Prover, after which he produces his output. Most of our schemes will be single-pass (and we shall call them simply "schemes"), but in a few cases, we will give multi-pass schemes when they can achieve provably better costs than single-pass schemes.

Setup and Terminology. All problems studied in this paper involve an input graph, multigraph, or digraph $G = (V, E)$ on the vertex set $[n] := \{1, 2, \ldots, n\}$. We shall reserve the basic term "graph" for simple, undirected graphs. The input is described either as a stream of edges (the default case) or as a stream of edge insertions and deletions: the latter type of stream is called a *dynamic* or *turnstile* graph stream. For an (h, v)-scheme to be interesting we at least require $v = o(n^2)$. If we also have $h = o(n^2)$, we call it a *sublinear* scheme. If we have $v = o(n)$ while $h = o(n^2)$, we call it a *frugal* scheme. This is an especially interesting setting of parameters, because most interesting graph problems provably require $\Omega(n)$ space in the basic streaming setting [15]. A frugal scheme shows that one can beat this space bound with the aid of only a sublinear-length proof. Recall that while $h \gg v$ is allowed, the proof must be processed using only $O(v)$ space.

For an (h, v)-scheme we refer to h as its hcost (short for "help cost") and v as its vcost (short for "verification cost"). We use the notation $[h, v]$-scheme as a shorthand for an $(\tilde{O}(h), \tilde{O}(v))$-scheme.[2] An $[n, n]$-scheme is called a *semi-streaming* scheme.

Subgraph Counting. The literature on graph streaming contains many works on the central problem of triangle counting (henceforth, TriangleCount): given a multigraph G as a dynamic stream, compute T, the number of triangles in G [6, 7, 18, 25, 29]. In Section 3, we study this and the more general problem of subgraph-counting (SubgraphCount$_k$) [7, 19, 20, 29], where the goal is to compute T_H, the number of copies

[1] We will drop the qualifier "online" and simply call our protocols "schemes" because we will not be considering the more powerful setting of "prescient schemes" [10] in this paper.

[2] The notation $\tilde{O}(\cdot)$ hides factors polynomial in $\log n$.

of a fixed, k-sized graph H, where k is a constant. In the basic streaming model, computing T or T_H exactly is impossible in sublinear space and it becomes necessary to approximate. In contrast, we design a family of $(o(n^2), o(n))$-schemes for TriangleCount that give exact answers. Such a frugal scheme had been conjectured not to exist [29]. We extend our ideas to give sublinear $(o(n^2), o(n^2))$-schemes for SubgraphCount$_k$.

Maximum Matching. Determining $\alpha'(G)$, the cardinality of a maximum-sized matching in G, is a central problem in graph algorithms and has received a lot of attention in the recent literature on streaming algorithms [5, 12, 15, 16, 21, 24]. In Section 4, we consider this problem (henceforth, MaxMatching) for multigraphs given by dynamic streams. As with TriangleCount, we give a frugal scheme for MaxMatching, which had been conjectured to be impossible [29]. In the process, we present a frugal scheme for the subproblem of verifying that the purported connected components of a graph are indeed disconnected from each other, which might be of independent interest for future work on connectivity-related problems.

Independent Sets and Length-Three Paths. In Section 5, we study the independent set testing problem (IndSetTest), where we are given a multigraph G and a set $U \subseteq V$ (also streamed and interleaved with the edge stream arbitrarily) and we must determine whether or not U is independent. We also study the st-3Path problem, where G (which might be a digraph) has two designated vertices v_s and v_t and we must determine whether G has a path of length at most 3 from v_s to v_t. By results from prior work, any (h, v)-scheme for these problems must have total cost $h + v = \Omega(n)$. We therefore design two-pass schemes for these problems, achieving $h + v = \tilde{O}(n^{2/3})$. In fact, we obtain a more general tradeoff, giving a two-pass $[t^2, s]$-scheme for any parameters t, s with $ts = n$. Our schemes instantiate a protocol for the abstract problem CrossEdgeCount, which asks for a count of the number of edges in G from $U \subseteq V$ to $W \subseteq V$, where these sets U and W are also streamed.

In each case, we *can* design ordinary (one-pass) schemes with the same complexity parameters under a natural assumption on the way the stream is ordered, and these schemes still beat the space bound achievable by basic (sans prover) streaming algorithms.

Short Paths and Shortest Path. Finally, in Section 6, we consider shortest path problems, perhaps the most basic problem in classic graph algorithms. We study the st-kPath problem, which is to detect whether or not G has a path of length at most k from v_s to v_t, where k, v_s, and v_t are prespecified. We first present a $[kn, n]$-scheme for st-kPath. This gives a semi-streaming scheme for detecting short (of length polylogarithmic in n) paths, which is optimal in terms of total cost. It also implies a $[kn, n]$-scheme for st-ShortestPath problem—where k is the length of the shortest path from v_s to v_t—which is to find the shortest path between vertices v_s and v_t, and output No if none exists. For directed graphs of small (polylogarithmic in n) diameter, it implies a semi-streaming scheme for checking v_s–v_t connectivity. Note that these problems require $\Omega(n^2)$ space in the basic data streaming model, even for constant k or constant-diameter graphs [15].

Targeting a different cost regime, we generalize our result for st-3Path from Section 5 to obtain multi-pass (h, v)-schemes for st-kPath with total cost $h + v = o(n)$, for constant k. To be precise, we present a $\lceil k/2 \rceil$-pass $[n^{1-1/k}, n^{1-1/k}]$-scheme for st-kPath.

Our Techniques. Similar to past work in the area of streaming verification—indeed, hearkening back to classic interactive proof protocols [23, 28]—our schemes make heavy use of "arithmetization," i.e., they recast the underlying problem in terms of evaluating certain polynomials and exploit the encoding properties of polynomials (as captured in the Schwartz-Zippel Lemma) to protect the verifier from a cheating prover. Also as in past work, we use what we call the *shaping technique*, where we conceptually shape a data vector into an array with two or more dimensions. This seemingly innocuous trick allows us to consider input data as a table of values of a *multivariate* polynomial and we can use the "separation" afforded by these multiple variables to divide up work between Verifier and Prover.

The novelty in our algorithms comes from a twist on the shaping technique that was hitherto unexploited. At a high level, almost all earlier annotation schemes or SIPs for graph problems viewed the edge stream as

a flat vector (a characteristic vector in the case of graphs or a frequency vector in the case of multigraphs). We crucially exploit the fact that the index set of this vector has additional structure: it consists of pairs of *vertices* and these vertices are very meaningful entities in the context of graph problems. Simply put, we exploit *graph structure* more fully in our use of the shaping technique.

Another novel feature of our schemes is that they involve Prover sending *multivariate* polynomials; their correctness analysis then involves the full multivariate strength of the Schwartz-Zippel Lemma. In all past work on interactive proofs and schemes, Prover only sent univariate polynomials (and the corresponding analyses used the more basic statement that a nonzero, degree-d, univariate polynomial has at most d roots). Thus, our scheme designs can be seen as exploiting the power of arithmetization more fully.

1.2 Related Work

The annotated data streaming model of computation was motivated in part by the need to develop a theory to capture ideas such as the stream punctuations of Tucker et al. [30] and the stream outsourcing framework of Yi et al. [31]. Chakrabarti et al. [10] formulated the model and provided the first theoretical results, focusing largely on the traditional statistical problems of frequency moments and heavy hitters, but also giving a handful of basic results for graph problems. Other early works in the same model include Klauck and Prakash [22] and Chakrabarti et al. [9]. Cormode et al. [14] generalized the model to SIPs, which allow a few interactive rounds of communication between Verifier and Prover; this generalized setting was studied further in Chakrabarti et al. [11] and Abdullah et al. [2]. We refer the reader to the expository article of Mitzenmacher and Thaler [26] for a more detailed survey of this area.

We turn to graph computations and the specific problems studied in this work. For simplicity, we state complexities in terms of n alone, rather than using both m and n (m being the number of edges of the input graph). Cormode et al. [13] gave annotated data stream algorithms (schemes, in our terminology) for many canonical graph problems, often exploiting linear programming formulations of the problems. In particular, they gave an $[n^2, 1]$-scheme For MaxMatching. For a weighted version of st-ShortestPath, on simple graphs (not multigraphs) they gave $[h, v]$-schemes with $hv \geqslant dn^2$ and $h \geqslant dn$, where d is the maximum distance to *any* node reachable from v_s. Contrast this with our $[kn, n]$-scheme for unweighted multigraphs, where k is the length of the shortest v_s–v_t path.

Thaler [29] studied the problems TriangleCount, MaxMatching, and SubgraphCount$_k$. He gave semi-streaming schemes for the first two. In the same paper, he explicitly conjectured that these two problems would not admit frugal schemes: he imagined that achieving vcost $= o(n)$ would bump up the hcost to $\Omega(n^2)$. Our results here disprove these conjectures. For SubgraphCount$_k$, Thaler gave a $[k^3 n, kn]$-SIP with $k - 2$ rounds of interaction. We achieve sublinear cost with just a single Prover-to-Verifier message. Sublinear schemes for SubgraphCount$_k$ were hitherto unknown for any $k > 3$.

For the TriangleCount problem, Chakrabarti et al. [10] gave an $[h, v]$-scheme for any h, v with $hv = n^3$, and also an $[n^2, 1]$-scheme. For the same problem, Abdullah et al. [2] gave a $(\log^2 n, \log^2 n)$-SIP that uses $\log n$ rounds of interaction, and a $(n^{1/\gamma} \log n, \log n)$-SIP with $\gamma = O(1)$ rounds of interaction. The latter paper also studied MaxMatching, giving a $(\rho + n^{1/\gamma'} \log n, \log n)$-SIP with γ rounds of interaction, where γ' is a linear function of γ, and ρ is the weight of an optimal matching (weighted or unweighted).

Guruswami and Onak [17] show a space lower bound of $\Omega(n^{1+\Omega(1/k)}/k^{O(1)})$ for st-kPath (where k is even) in $k/2 - 1$ passes in the basic streaming model. In contrast, our results show that, for any k, with the help of a prover, one can get a total cost of $\tilde{O}(n^{1-1/k})$ in $\lceil k/2 \rceil$ passes.

2 Preliminaries

For a positive integer n, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$ and the set $\{-n, -n+1, \ldots, n-1, n\}$ by $[\![n]\!]$. The ring of polynomials in variables $X_1, \ldots, X_k$ with coefficients in the ring R is denoted by $R[X_1, \ldots, X_k]$. If S is a finite set, we write $r \in_R S$ to say that r is a random element drawn uniformly from S. In an undirected graph $G = (V, E)$, the *ith neighborhood* of a vertex v is the set of vertices u such that there is a walk of length i from v to u. We denote this set by $N_i(v)$. We put $N(v) := N_1(v)$ and $N[v] := N_1(v) \cup \{v\}$.

Following Chakrabarti et al. [10], an annotated data streaming algorithm, a.k.a. *scheme*, is a pair $\mathcal{A} = (\mathfrak{h}, \mathcal{B})$, where $\mathfrak{h}$ is a help function and $\mathcal{B}$ is a data stream algorithm that computes a function f of an input $\mathbf{x} \in \mathcal{U}^m$, where $\mathcal{U}$ is some universe. We see $\mathfrak{h}$ as an m-tuple $(\mathfrak{h}_1, \ldots, \mathfrak{h}_m)$, where $\mathfrak{h}_i : \mathcal{U}^i \to \{0,1\}^*$ is the *annotation* provided to $\mathcal{B}$ after the ith stream update x_i, depending on the elements seen so far, i.e. $x_1, \ldots, x_i$. Thus, $\mathcal{B}$ sees the *annotated* stream $\mathbf{x}^{\mathfrak{h}} := (x_1, \mathfrak{h}_1(x_1), x_2, \mathfrak{h}_2(x_1, x_2), \ldots, x_m, \mathfrak{h}_m(x_1, \ldots, x_m))$. Using a random string R, it processes this annotated stream, giving an output $\mathrm{out}(\mathcal{B}; \mathbf{x}^{\mathfrak{h}}, R)$. We say that $\mathcal{A}$ is a δ-*error* scheme if

- (completeness) for all $\mathbf{x} \in \mathcal{U}^m$: $\Pr_R[\mathrm{out}(\mathcal{B}; \mathbf{x}^{\mathfrak{h}}, R) \neq f(\mathbf{x})] \leqslant \delta$; and
- (soundness) for all $\mathbf{x} \in \mathcal{U}^m$, $\mathfrak{h}' = (\mathfrak{h}'_1, \mathfrak{h}'_2, \ldots, \mathfrak{h}'_m) \in (\{0,1\}^*)^m$: $\Pr_R[\mathrm{out}(\mathcal{B}; \mathbf{x}^{\mathfrak{h}'}, R) \notin \{f(\mathbf{x}), \perp\}] \leqslant \delta$,

where "$\perp$" is a special symbol indicating that $\mathcal{B}$ rejects the annotation (proof) provided, having detected cheating. When δ is left unspecified, we assume a default value of $1/3$. The hcost (help cost) of $\mathcal{A}$ is $\max_{\mathbf{x} \in \mathcal{U}^m} \sum_i |\mathfrak{h}_i(\mathbf{x})|$, and the vcost (verification cost) is the space usage of $\mathcal{B}$.

The scheme $\mathcal{A}$ is said to be an (h, v)-scheme (resp. $[h, v]$-scheme) if its hcost is $O(h)$ (resp. $\tilde{O}(h)$) and its vcost is $O(v)$ (resp. $\tilde{O}(v)$). The sum hcost + vcost is called the *total cost* of $\mathcal{A}$. In the context of problems on n-vertex graphs, an $(o(n^2), o(n^2))$-scheme is called a *sublinear* scheme, an $[n, n]$-scheme is called a *semi-streaming* scheme and an $(o(n^2), o(n))$-scheme is called a *frugal* scheme.

A *multi-pass scheme*—more precisely, a p-pass scheme with $p \geqslant 2$—is a scheme $\mathcal{A} = (\mathfrak{h}, \mathcal{B})$ where $\mathcal{B}$ makes $p-1$ passes over the input $\mathbf{x}$ followed by a final pass over the annotated stream $\mathbf{x}^{\mathfrak{h}}$. As discussed in Section 1.1, all schemes and multi-pass schemes we design in this work have the feature that the entire annotation $\mathfrak{h}(\mathbf{x})$ arrives only after $\mathcal{B}$ is done processing the plain stream $\mathbf{x}$. That said, the negative results in this work do not require the scheme to be restricted in this way.

Let f be a k-dimensional array with dimensions $(s_1, \ldots, s_k)$ each of whose entries is an integer in $[\![M]\!]$. Equivalently, we have a function $f\colon [s_1] \times \cdots \times [s_k] \to [\![M]\!]$. For a finite field $\mathbb{F}$ of sufficiently large characteristic,[3] we define the $\mathbb{F}$-extension of f to be the unique polynomial $\tilde{f}(X_1, \ldots, X_k) \in \mathbb{F}[X_1, \ldots, X_k]$ such that

- for all $(x_1, \ldots, x_k) \in [s_1] \times \cdots [s_k]$, we have $\tilde{f}(x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$, and
- for all $i \in [k]$, we have $\deg_{X_i} \tilde{f} \leqslant s_i - 1$.

Note that $\tilde{f}$ can be described explicitly using Lagrange interpolation:

$$\tilde{f}(X_1, \ldots, X_k) = \sum_{(u_1, \ldots, u_k) \in [s_1] \times \cdots \times [s_k]} f(u_1, \ldots, u_k)\, \delta_{u_1, \ldots, u_k}(X_1, \ldots, X_k)\,, \quad \text{where} \tag{1}$$

$$\delta_{u_1, \ldots, u_k}(X_1, \ldots, X_k) = \prod_{i=1}^{k} \prod_{x_i \in [s_i] \setminus \{u_i\}} (u_i - x_i)^{-1} (X_i - x_i)\,. \tag{2}$$

In particular, if f is built up from a stream of pointwise updates, where the jth update adds Δ_j to entry

[3] We need the characteristic to be at least $\max\{s_1, \ldots, s_k, 2M+1\}$ to avoid "wrap around problems," i.e., to ensure that all integers in each $[s_i]$ as well as all integers in $[\![M]\!]$ have distinct images under the ring homomorphism from $\mathbb{Z}$ to $\mathbb{F}$.

$(u_1, \ldots, u_k)_j$ of the array, then

$$\tilde{f}(X_1, \ldots, X_k) = \sum_j \Delta_j \, \delta_{(u_1,\ldots,u_k)_j}(X_1, \ldots, X_k) \,. \tag{3}$$

This leads to the following fact that we use in all our protocols. For details and a thorough discussion, including implementation considerations, see Cormode et al. [14].

Fact 2.1. *Given a point $(p_1, \ldots, p_k) \in \mathbb{F}^k$ and a stream of pointwise updates to an array with dimensions $(s_1, \ldots, s_k)$ that is initially all-zero, we can keep track of the value $\tilde{f}(p_1, \ldots, p_k)$ using $O(\log |\mathbb{F}|)$ space, performing $O(k)$ field arithmetic operations after each update.* □

We record results proved in Chakrabarti et al. [9, 10] that can be seen as generalizing the Aaronson-Wigderson protocol for Merlin-Arthur communication complexity of set disjointness [1].

Fact 2.2 (SUBSET and INTERSECTION schemes; Prop. 4.1 of [10] and Thm. 5.3 of [9])**.** *Consider a stream consisting of elements of two sets $S, T \subseteq [N]$ interleaved arbitrarily. Then, for any h, v with $hv \geqslant N$, there are $[h, v]$-schemes to compute $|S \cap T|$ and to determine whether $S \subseteq T$. For the latter problem, there is a $[\ell h, v]$-scheme handling the more general setting where S and T are multisets updated dynamically by the stream and the multiplicity of each element is at most ℓ.* □

Fact 2.3 (Schwartz-Zippel Lemma)**.** *For a nonzero polynomial $P(X_1, \ldots, X_n) \in \mathbb{F}[X_1, \ldots, X_n]$ of total degree d, where $\mathbb{F}$ is a finite field, $\Pr_{(r_1,\ldots,r_n)\in_R \mathbb{F}^n}[P(r_1, \ldots, r_n) = 0] \leqslant d/|\mathbb{F}|$.* □

3 Subgraph Counting

We begin by describing a frugal scheme for TRIANGLECOUNT and then extend our ideas to obtain a sublinear scheme for the more general problem SUBGRAPHCOUNT. Throughout, we assume that the input is an n-vertex multigraph $G = (V, E)$ with adjacency matrix A, built up through a stream of edge insertions and deletions.

3.1 Triangle Counting

Let $T = T(G)$ be the number of triangles in G taking edge multiplicities into account, i.e., two triangles are considered distinct iff their corresponding sets of *edges* are distinct. Then,

$$6T = \sum_{v_1, v_2, v_3 \in V} A_{v_1 v_2} A_{v_2 v_3} A_{v_3 v_1} \,. \tag{4}$$

Let t and s be integer-valued parameters such that $ts = n$. Using a canonical bijection, we represent each vertex $v \in V$ by a pair of integers $(x, y) \in [t] \times [s]$. This transforms the matrix A into a 4-dimensional array a, given by $a(x_1, y_1, x_2, y_2) = A_{v_1 v_2}$. Let $\tilde{a}$ be the $\mathbb{F}$-extension of a for a sufficiently large finite field $\mathbb{F}$ to be chosen later. Equation (4) now gives

$$6T = \sum_{x_1, x_2, x_3 \in [t]} p(x_1, x_2, x_3) \,, \quad \text{where} \tag{5}$$

$$p(X_1, X_2, X_3) = \sum_{y_1, y_2, y_3 \in [s]} \tilde{a}(X_1, y_1, X_2, y_2) \, \tilde{a}(X_2, y_2, X_3, y_3) \, \tilde{a}(X_3, y_3, X_1, y_1) \,. \tag{6}$$

Note that, for each $i \in \{1, 2, 3\}$, we have $\deg_{X_i} p \leqslant 2t - 2$. Thus, the number of monomials in p is at most $(2t - 1)^3 \leqslant 8t^3$ and the total degree $\deg p \leqslant 6t - 6 \leqslant 6t$.

Our scheme for triangle counting operates as follows.

Stream processing. Verifier starts by picking $r_1, r_2, r_3 \in_R \mathbb{F}$. As the edge stream arrives, he maintains the three 2-dimensional arrays $\tilde{a}(r_1, w, r_2, z)$, $\tilde{a}(r_2, w, r_3, z)$, and $\tilde{a}(r_3, w, r_1, z)$, for all $(w, z) \in [s] \times [s]$ (using Fact 2.1). At the end of the stream, he uses these arrays to compute $p(r_1, r_2, r_3)$, using eq. (6).

Help message. Prover sends Verifier a polynomial $\hat{p}(X_1, X_2, X_3)$ that she claims equals $p(X_1, X_2, X_3)$; in particular, for each $i \in \{1, 2, 3\}$, $\deg_{X_i} \hat{p} \leqslant 2t - 2$. She streams the coefficients of $\hat{p}$ one at a time, according to some canonical ordering of the possible monomials.

Verification and output. As $\hat{p}$ is streamed in, Verifier computes the check value $C := \hat{p}(r_1, r_2, r_3)$ and the result value $\hat{T} := \frac{1}{6} \sum_{x_1, x_2, x_3 \in [t]} \hat{p}(x_1, x_2, x_3)$. If he finds that $C \neq p(r_1, r_2, r_3)$, he outputs $\perp$. Otherwise, he believes that $\hat{p} \equiv p$ and accordingly, based on eq. (5), outputs $\hat{T}$ as the answer.

The analysis of this scheme is along now-standard lines.

Error probability. Clearly, if Prover is honest (i.e., $\hat{p} \equiv p$), then the output is always correct. So the scheme errs only when $\hat{p} \not\equiv p$ but Verifier's check passes. This means that the random point $(r_1, r_2, r_3) \in \mathbb{F}^3$ is a root of the nonzero polynomial $\hat{p} - p$, which has total degree at most $6t$. By the Schwartz-Zippel Lemma (Fact 2.3), the probability of this event is at most $6t/|\mathbb{F}| < 1/n$, by choosing $|\mathbb{F}|$ large enough.

Help and Verification costs. The number of bits used to describe the polynomial $\hat{p}$ is the hcost. As noted, the polynomial $\hat{p}$ has $O(t^3)$ many coefficients, each of which is an element of $\mathbb{F}$, and hence has size $O(\log n)$. So the hcost is $\tilde{O}(t^3)$. The Verifier maintains three $s \times s$ arrays, where each entry is an element of $\mathbb{F}$. Hence, the vcost is $\tilde{O}(s^2)$. Therefore, we get a $[t^3, s^2]$-scheme for triangle counting, for parameters t, s with $ts = n$. Setting $t = n^\alpha$ for $\alpha \in (1/2, 2/3)$, we get a $(o(n^2), o(n))$-scheme, which is frugal.

The result in this section is captured in the theorem below.

Theorem 3.1. *For any parameters t, s with $ts = n$, there is a $[t^3, s^2]$-scheme for* TriangleCount. *In particular, there is an $(o(n^2), o(n))$-scheme for* TriangleCount. □

This disproves Thaler's conjecture [29], which stated that TriangleCount has no frugal scheme.

3.2 Generalization to Counting Copies of an Arbitrary Subgraph

Now we consider the SubgraphCount$_k$ problem. Let H be a fixed k-vertex graph. The goal is to determine $T_H = T_H(G)$, the number of copies of H in the n-vertex multigraph G given by an input stream: n is growing whereas $k = O(1)$. As before, we take edge multiplicities into account.

Fix a numbering of the vertices of H as $1, 2, \ldots, k$. Write $i \sim j$ to denote $\{i, j\} \in E(H) \wedge i < j$. To generalize eq. (4), note that the expression $\prod_{i \sim j} A_{v_i v_j}$ counts the number of copies of H occurring amongst vertices $v_1, \ldots, v_k$ in G where $i \in V(H)$ is mapped to $v_i \in V$, *provided that* $v_1, \ldots, v_k$ *are distinct*. This subtlety of explicitly requiring the v_is to be distinct did not arise for TriangleCount because $A_{v_1 v_2} A_{v_2 v_3} A_{v_3 v_1}$ is zero unless v_1, v_2, v_3 are distinct. To enforce the distinctness condition in our more general setting, define an $n \times n$ Boolean matrix B by $B_{uv} = 1$ iff $u \neq v$. Then, defining α_H to be the number of automorphisms of H,

$$\alpha_H T_H = \sum_{v_1, \ldots v_k \in V} \left(\prod_{i \sim j} A_{v_i v_j} \right) \left(\prod_{i \neq j \in [k]} B_{v_i v_j} \right). \tag{7}$$

As before, we shape V into $[t] \times [s]$ for parameters t and s with $ts = n$. This turns the 2-dimensional matrices A, B into 4-dimensional arrays a, b, which in turn have $\mathbb{F}$-extensions $\tilde{a}, \tilde{b}$. Equation (7) gives

$$\alpha_H T_H = \sum_{x_1,\ldots,x_k \in [t]} p(x_1, \ldots, x_k), \quad \text{where} \tag{8}$$

$$p(X_1, \ldots, X_k) = \sum_{y_1,\ldots,y_k \in [s]} \left(\prod_{i \sim j} \tilde{a}(X_i, y_i, X_j, y_j) \right) \left(\prod_{i \neq j \in [k]} \tilde{b}(X_i, y_i, X_j, y_j) \right). \tag{9}$$

For each $i \in [k]$, $\deg_{X_i} p \leqslant 2(k-1)(t-1) = O(t)$. So the total degree $\deg p = O(t)$ and p has at most $O(t^k)$ monomials. This leads to a scheme for subgraph counting that naturally generalizes our earlier scheme for triangle counting. We sketch the salient features and the analysis.

Stream processing. Verifier picks $r_1, \ldots, r_k \in_R \mathbb{F}$ and maintains (using Fact 2.1) $O(k^2) = O(1)$ many $s \times s$ arrays: $\tilde{a}(r_i, w, r_j, z)$ for each $i \sim j \in [k]$ and $\tilde{b}(r_i, w, r_j, z)$ for each $i \neq j \in [k]$, where $(w, z) \in [s] \times [s]$. The $\tilde{b}$ arrays do not depend on the input stream and can be computed once and for all. At the end of the stream, he computes $p(r_1, \ldots, r_k)$ with the help of these values, using eq. (9).

Help message. Prover sends a polynomial $\hat{p}(X_1, \ldots, X_k)$ that she claims to be $p(X_1, \ldots, X_k)$. She streams the $O(t^k)$ coefficients of $\hat{p}$, using some canonical ordering of the monomials.

Verification and output. Verifier computes the check value $C := \hat{p}(r_1, \ldots, r_k)$ and the result value $\hat{T}_H := \alpha_H^{-1} \sum_{x_1,\ldots,x_k \in [t]} \hat{p}(x_1, \ldots, x_k)$. He outputs $\perp$ if $C \neq p(r_1, \ldots, r_k)$. Else, believing $\hat{p} \equiv p$, he outputs $\hat{T}_H$ as the answer, in view of eq. (8).

Error probability. By a Schwartz-Zippel Lemma (Fact 2.3) argument as before, the error probability is at most $\deg p / |\mathbb{F}| = O(t)/|\mathbb{F}| < 1/n$, by choosing $|\mathbb{F}|$ large enough.

Help and Verification costs. The hcost is $\tilde{O}(t^k)$, by the bound on the number of monomials in $\hat{p}$. Verifier stores $O(1)$ many $s \times s$ arrays, leading to a vcost of $\tilde{O}(s^2)$.

In summary, we obtain a $[t^k, s^2]$-scheme for counting copies of a fixed k-vertex subgraph H, for all choices of parameters t, s with $ts = n$. Setting $t = n^{2/(k+2)}$ and $s = n^{k/(k+2)}$ gives a scheme where both these costs are $\tilde{O}(n^{2k/(k+2)})$, which is $o(n^2)$ for constant k. Thus, we get the following theorem.

Theorem 3.2. *For any parameters t, s such that $ts = n$, there is a $[t^k, s^2]$-scheme for* SubgraphCount$_k$*, where k is a constant. In particular, there is a sublinear scheme for* SubgraphCount$_k$ *with total cost $\tilde{O}(n^{2k/(k+2)})$.* □

4 Maximum Matching

We now turn to the MaxMatching problem, again giving a frugal scheme. Our input is an edge stream of an n-vertex graph $G = (V, E)$ and we would like to determine $\alpha'(G)$, the cardinality of a maximum matching in G. We follow the broad outline of the semi-streaming scheme for MaxMatching by Thaler [29]. That scheme has two parts. In the first part, Prover convinces Verifier that $\alpha'(G) \geqslant k$, for some integer k. In the second part, she convinces him that $\alpha'(G) \leqslant k$. For the former, Prover simply provides a suitable matching M and convinces Verifier that $M \subseteq E$ using the subset scheme from Fact 2.2. For the latter, Prover uses the Tutte-Berge formula [8], which states that

$$\alpha'(G) = \frac{1}{2} \min_{U \subseteq V} \Big(|U| + |V| - \text{odd}(G \setminus U) \Big), \tag{10}$$

where odd($G \setminus U$) denotes the number of connected components in $G \setminus U$ with an odd number of vertices. The most challenging part of the scheme is evaluating odd($G \setminus U$), which involves the sub-problem of verifying whether all the connected components of a graph (as claimed by the Prover) are disconnected from each other. Thaler comments that this is the part that acts as a barrier in reducing the vcost to $o(n)$ without increasing the hcost to $\Omega(n^2)$. We present a novel frugal scheme for this sub-problem. The rest of the protocol solves the same sub-problems as the aforementioned paper. Most of their sub-schemes for these sub-problems, however, were trivial for $\tilde{O}(n)$ space. We need schemes for the same problems that use only $o(n)$ space and hence require more work. We describe our protocol below.

To convince the Verifier that the size of a maximum matching in G is k, Prover proves that it is (a) at least k, and (b) at most k. For (a), she simply sends (as a stream) a set M of k edges that constitutes a matching of G. Verifier can easily check using $O(\log n)$ space that the set has size k. Next, he needs to check that $M \subseteq E$, and that M is indeed a matching. For the former, we can use the SUBSET scheme (Fact 2.2) and get an $[h, v]$-scheme, where v is the $o(n)$ value we are aiming for and $h = n^2/v$. To verify that M is a matching, we check whether every vertex in M appears exactly once in this stream. Treating M as a stream of vertices, we can do this as follows: First, compute F_2, the second frequency moment of the stream, using an $[h, v]$-scheme where v is the $o(n)$ vcost we want, and $h = n/v$ ([10], Theorem 4.1). Next, verify that it equals $2k$ (this happens iff all $2k$ elements are distinct).

For (b), we apply eq. (10). Prover sends $U^* \subseteq V$ and claims that $k = \frac{1}{2}(|U^*| + |V| - \text{odd}(G \setminus U^*))$. To check this, Verifier just needs to compute odd($G \setminus U^*$). We do this in the following way.

Let $[C]$ be the set of C connected components of $G \setminus U^*$. For $c \in [C]$ and $u \in G \setminus U^*$, Prover sends an array L of pairs (c, u) such that $u \in c$. The array L is sorted in non-decreasing order of c, i.e., she first sends the vertices in connected component 1, followed by those in component 2, and so on. If L is indeed as Prover claims, then odd($G \setminus U^*$) is equal to the number of components c that arrive with an odd number of vertices in L. Since L is sorted with respect to c, Verifier can count this number easily using $O(\log n)$ space. He can verify that the vertices in the tuples of L constitute $G \setminus U^*$, and that no vertex u is repeated in different tuples of L, using frugal schemes implied by the standard protocols mentioned above.

Thus, it only remains to verify that L is as claimed. For this, we need to check whether the following two properties hold:

(i) For each $c \in [C]$, the vertices in $G \setminus U^*$ that are claimed to be in component c are all connected in $G \setminus U^*$.

(ii) For every pair (u, v) of vertices in $G \setminus U^*$ that are claimed to be in different components, we have $(u, v) \notin E$.

For Property (i), Prover sends a spanning tree for each connected component c and Verifier can check if all of them are valid using an $[n^{1+\alpha}, n^{1-\alpha}]$-scheme, for any $\alpha \in [0, 1]$ ([10], Theorem 7.7) so as to get the desired $o(n)$ vcost.

Checking Property (ii) is the most challenging part. We give a novel protocol for this part that uses $o(n)$ vcost and $o(n^2)$ hcost. Slightly abusing notation, consider the array L in the form of a $C \times |G \setminus U^*|$ matrix, such that $L_{cu} = 1$ if $u \in c$, and $L_{cu} = 0$ otherwise. Denote the ones' complement of this matrix by $\overline{L}$. Let A be the adjacency matrix of $G \setminus U^*$. Finally, let γ denote the total number of cross edges that go between two connected components in $G \setminus U^*$. Then, we have

$$2\gamma = \sum_{\substack{c \in [C] \\ u,v \in G \setminus U^*}} L_{cu} \overline{L}_{cv} A_{uv} \,. \tag{11}$$

Property (ii) is satisfied iff $\gamma = 0$. Recalling that $C = O(n)$ and $|G \setminus U^*| = O(n)$, we note that eq. (11) has a similar form as that of eq. (4). Thus, it can be exploited in essentially the same way as the $[t^3, s^2]$-scheme

for TriangleCount, for parameters t, s with $ts = n$. Once again, setting $t = n^\alpha$ for $\alpha \in (1/2, 2/3)$, we get a frugal scheme.

The next theorem summarizes the result in this section.

Theorem 4.1. *For any parameters t, s with $ts = n$, there is a $[t^3, s^2]$-scheme for* MaxMatching. *In particular, there is an $(o(n^2), o(n))$-scheme for* MaxMatching. □

This disproves yet another conjecture of Thaler [29], which stated that MaxMatching has no frugal scheme.

5 Counting Cross-edges and its Applications to Other Problems

Consider the problems IndSetTest and st-3Path defined in Section 1.1. The key task underlying these problems is counting the number of edges crossing between two subsets U and W of V that arrive in some adversarial streaming order along with the edges: for IndSetTest, U and W are the same set; for st-3Path, they are (closed) neighborhoods of the designated vertices v_s and v_t. This is precisely the abstract problem of CrossEdgeCount. Clearly, a scheme for this problem can be used as a subroutine to solve IndSetTest and st-3Path.

Any one-pass (h, v)-scheme for CrossEdgeCount, IndSetTest, or st-3Path must have $hv \geqslant n^2$ and hence, total cost $h + v = \Omega(n)$. We therefore consider *two-pass* schemes for these problems. In particular, we design such a scheme for CrossEdgeCount with total cost $\tilde{O}(n^{2/3})$ and apply it to obtain similar bounds for other graph problems. We also note that our schemes can be implemented in one pass each, under natural assumptions on the way the stream is ordered; this is addressed in Section 5.3.

5.1 One-Pass Lower Bounds

We quickly review some relevant material from communication complexity. In the index_N problem, there are two players: Alice, who holds a vector $\mathbf{x} \in \{0,1\}^N$, and Bob, who holds an index $k \in [N]$. Their goal is to output the bit $\mathbf{x}_k$. To prove lower bounds for one-pass *schemes*, we consider the *Online Merlin–Arthur* (OMA) communication model.[4] Here, in addition to Alice and Bob, there is a super-player, Merlin, who knows both their inputs, but is not to be blindly trusted. Merlin sends a message to Bob; then Alice sends a randomized message to Bob; finally, Bob either outputs either a bit or $\perp$. If Merlin is honest, Bob should output $\mathbf{x}_k$ with probability at least $2/3$; if he is dishonest, Bob should output $\perp$ with probability at least $2/3$.

The cost of an OMA protocol is the total number of bits communicated to Bob. The OMA complexity of a communication game is the minimum cost of a correct OMA protocol for it. Chakrabarti et al. [10, Theorem 3.1] showed that the OMA Complexity of index_N is $\Omega(\sqrt{N})$. Our lower bounds follow from this result, using simple reductions from index_N to the various graph problems.

Using a canonical bijection from $[n]^2$ to $[N]$, Alice rewrites her input vector $\mathbf{x} \in \{0,1\}^N$ as a matrix $(\mathbf{x}_{ij})_{i,j\in[n]}$, while Bob looks at his input index $k \in [N]$ as $(y, z) \in [n]^2$. Our reduction creates a graph $G = (V, E)$ on $2n$ vertices: the vertex set V is $L \uplus R$ (here, $\uplus$ denotes disjoint union), where $|L| = |R| = n$. We denote the ith vertex of L (resp. R) by ℓ_i (resp. r_i). The edge set E is given by $\{(\ell_i, r_j) : \mathbf{x}_{ij} = 1\}$. Now, by checking if (ℓ_y, r_z) is an independent set in G, or whether there's a cross-edge between the sets $\{\ell_y\}$ and $\{r_z\}$, or solving st-3Path in the graph $G' = (V \cup \{v_s, v_t\}, E \cup \{(v_s, \ell_y), (r_z, v_t)\})$, Bob can solve the index_N problem. Thus, a one-pass scheme that solves any of these problems must have a total cost of $\Omega(n)$. We remark that Fact 2.2 implies matching semi-streaming upper bounds for each of them.

[4] Note that our semantics are slightly different from the usual definition of Merlin–Arthur where Bob is supposed "accept" each 1-input and reject each 0-input with probability at least $2/3$.

5.2 Two-pass Scheme for CrossEdgeCount with Applications

We now design a two-pass scheme for CrossEdgeCount, aiming for total cost $o(n)$.

Let $\gamma = \gamma(U, W, G)$ denote the number of Cross-edges between U and W in a (directed or undirected) graph G. Formally, it is the number of ordered pairs $(u, w) \in U \times W$ such that $(u, w) \in E$. Note that, in an undirected graph, γ counts an edge (u, w) with multiplicity 2 whenever $u, w \in U \cap W$. For some applications (e.g., counting number of 3-walks in an undirected graph), we *do* need to count them with multiplicity. We discuss later how we can remove this multiplicity if needed.

We describe a scheme that works even on turnstile graph streams, i.e., a stream of the vertices in U and W intermixed with *updates* to edge multiplicities. Let L and F denote the characteristic vectors of the sets U and W respectively and let A be the (weighted) adjacency matrix of G. Then,

$$\gamma = \sum_{u \in U, w \in W} L_u A_{u,w} F_w \,. \tag{12}$$

Let t and s be integer parameters such that $ts = n$. As usual, using a canonical bijection, we represent each vertex $v \in V$ by a pair of integers $(x, y) \in [t] \times [s]$. As a result, the vectors L, F transform into 2-dimensional arrays ℓ, f given by $\ell(x, y) = L_v$ and $f(x, y) = F_v$. As before, the adjacency matrix A turns into a 4-dimensional array a, such that $a(x_1, y_1, x_2, y_2) = A_{v_1 v_2}$. Let $\tilde{\ell}$, $\tilde{f}$ and $\tilde{a}$ be $\mathbb{F}$-extensions of ℓ, f and a respectively, for a sufficiently large finite field $\mathbb{F}$. Now, eq. (12) yields

$$\gamma = \sum_{x_1, x_2 \in [t]} p(x_1, x_2) \,, \quad \text{where} \tag{13}$$

$$p(X_1, X_2) = \sum_{y_1, y_2 \in [s]} \tilde{\ell}(X_1, y_1)\, \tilde{a}(X_1, y_1, X_2, y_2)\, \tilde{f}(X_2, y_2) \,. \tag{14}$$

For $i \in \{1, 2\}$, $\deg_{X_i} p = 2t - 2$. Thus, it follows that the number of monomials in p is at most $O(t^2)$, and the total degree of p is $O(t)$.

We are now ready to design a two-pass scheme for CrossEdgeCount.

Stream processing. Verifier first chooses $r_1, r_2 \in_R \mathbb{F}$. For $y \in [s]$, define

$$g(y) := \sum_{y' \in [s]} \tilde{a}(r_1, y, r_2, y') \tilde{f}(r_2, y') \tag{15}$$

Thus,

$$p(r_1, r_2) = \sum_{y \in [s]} \tilde{\ell}(r_1, y) g(y) \,. \tag{16}$$

Pass 1. Only process the vertices in L and F in the stream. Maintain (using Fact 2.1) two s-dimensional vectors: $\tilde{\ell}(r_1, y)$ and $\tilde{f}(r_2, y)$, where $y \in [s]$.

Pass 2. Only process the edges in the stream. We want to maintain the s-dimensional vector $g(y)$ so that we can compute $p(r_1, r_2)$ using eq. (16). Suppose that the jth edge update $(x_1, y_1, x_2, y_2)_j$ adds Δ_j to that edge's multiplicity. This results in updates to several entries of $\tilde{a}$, but we want to use only $O(s)$ space, so we cannot afford to maintain $\tilde{a}$ directly. Instead, for each $j \in [m]$, let g_j and $\tilde{a}_j$ denote the values of g and $\tilde{a}$ (respectively) after the jth stream update. Then

$$\begin{aligned} g_j(y) &= \sum_{y' \in [s]} \tilde{f}(r_2, y')\, \tilde{a}_j(r_1, y, r_2, y') \\ &= \sum_{y' \in [s]} \tilde{f}(r_2, y') \left(\tilde{a}_{j-1}(r_1, y, r_2, y') + \Delta_j\, \delta_{(x_1, y_1, x_2, y_2)_j}(r_1, y, r_2, y') \right) \\ &= g_{j-1}(y) + h_j(y) \,, \end{aligned} \tag{17}$$

where eq. (17) follows from eq. (3) and

$$h_j(y) := \sum_{y' \in [s]} \tilde{f}(r_2, y')\, \Delta_j\, \delta_{(x_1,y_1,x_2,y_2)_j}(r_1, y, r_2, y')\,. \tag{18}$$

Hence, after the jth update, the Verifier can compute $h_j(y)$ and maintain the vector $g(y)$.

Help message. After the second pass, Prover sends a polynomial $\hat{p}(X_1, X_2)$ (as a stream of coefficients) that she claims equals $p(X_1, X_2)$.

Verification and output. At the end of the second pass, Verifier gets $g(y)_m = g(y)$ for each y. Now, he uses eq. (16) to compute the check value $p(r_1, r_2)$ and the result value $\hat{\gamma} := \sum_{x_1,x_2 \in [t]} \hat{p}(x_1, x_2)$. If he finds that $p(r_1, r_2) \neq \hat{p}(r_1, r_2)$, he outputs $\perp$. Otherwise, he believes that $\hat{p} \equiv p$ and exploiting eq. (13), outputs $\hat{\gamma}$ as the answer.

Now, we analyze the correctness and complexity parameters of the scheme.

Error probability. The protocol errs only when $\hat{p} \not\equiv p$, but Verifier's check passes. Then, $(r_1, r_2) \in \mathbb{F}^2$ must be a root of the nonzero polynomial $\hat{p} - p$. We noted that its total degree is $O(t)$. Thus, the Schwartz-Zippel Lemma bounds the error probability by at most $O(t)/|\mathbb{F}| < 1/n$, for large enough choice of $|\mathbb{F}|$.

Help and Verification costs. The polynomial $\hat{p}$ has $O(t^2)$ monomials, and so, the hcost is $\tilde{O}(t^2)$. Verifier stores constant many vectors of size s at a time and incurs a vcost of $\tilde{O}(s)$.

Thus, we obtain a two-pass $[t^2, s]$-scheme for CrossEdgeCount, for parameters t, s with $ts = n$. Setting $t = n^{1/3}$ and $s = n^{2/3}$, we get a scheme with total cost $\tilde{O}(n^{2/3})$.

Finally, we discuss how one can count cross-edges between U and W when they are defined as unordered pairs. Define this problem as CrossEdgeCount-Uniq. Let γ' be the number of edges that γ counts with multiplicity 2, i.e., the number of undirected edges $(u, w) \in U \times W$ such that $u, w \in U \cap W$. Then,

$$\gamma' = \sum_{u \in U, w \in W} L_u F_u A_{u,w} L_w F_w\,. \tag{19}$$

Hence, we modify the definitions of $p(X_1, X_2)$ and $g(y)$ as

$$p(X_1, X_2) := \sum_{y_1,y_2 \in [s]} \tilde{\ell}(X_1, y_1)\tilde{f}(X_1, y_1)\, \tilde{a}(X_1, y_1, X_2, y_2)\, \tilde{\ell}(X_2, y_2)\tilde{f}(X_2, y_2)\,. \tag{20}$$

$$g(y) := \sum_{y' \in [s]} \tilde{a}(r_1, y, r_2, y')\tilde{\ell}(r_2, y')\tilde{f}(r_2, y'). \tag{21}$$

Then, proceeding as in CrossEdgeCount, we compute γ'. Thus, we can compute γ and γ' in parallel and finally output $\gamma - \gamma'$ as the answer to CrossEdgeCount-Uniq.

Theorem 5.1. *For parameters t, s with $ts = n$, there are two-pass $[t^2, s]$-schemes for* CrossEdgeCount *and* CrossEdgeCount-Uniq. *In particular, there are two-pass schemes with total cost $\tilde{O}(n^{2/3})$.* □

Applications. Our scheme for CrossEdgeCount can be used as a black box for solving a number of other problems. These include standard problems like IndSetTest and st-3Path, as well as their generalizations or variations such as the following problems.

- InducedEdgeCount: Given a graph $G = (V, E)$ and a subset U of V, find the number of edges in G that are induced by U.

- RootedTriangleCount: Given a (directed or undirected) graph $G = (V, E)$ and a vertex $v_r \in V$, find the number of triangles in G that are rooted at v_r.

Corollary 5.2. *Let t and s be parameters such that $ts = n$. Then each of the problems* InducedEdgeCount, IndSetTest, st-3Path, *and* RootedTriangleCount *admits a two-pass $[t^2, s]$-scheme; in particular, each of them admits a two-pass scheme with total cost $\tilde{O}(n^{2/3})$.*

Proof. For InducedEdgeCount, if the input graph is undirected, then considering U and W as the same set, solve CrossEdgeCount-Uniq. (Alternatively, solve CrossEdgeCount and divide the answer by two.) If the graph is directed, then solve CrossEdgeCount.

For IndSetTest, solve InducedEdgeCount on U and check whether the answer equals zero.

For st-3Path, use a scheme for CrossEdgeCount to find the number of cross-edges between the closed neighborhoods $N[v_s]$ and $N[v_t]$ of vertices v_s and v_t. This actually solves the more general problem of counting the number of walks of length at most 3 from v_s to v_t. Checking whether this number is non-zero decides st-3Path.

Finally, for RootedTriangleCount, if the input graph is undirected, solve InducedEdgeCount on $N(v_r)$. Otherwise, solve CrossEdgeCount on the out-neighborhood $N^+(v_r)$ and in-neighborhood $N^-(v_r)$ of v_r. □

5.3 One-Pass Schemes for Certain Stream Orderings

Our two-pass solution to the CrossEdgeCount problem, as well as its corollaries, allowed the vertices and edge updates to be arbitrarily intermixed in the input stream. That said, it is interesting to focus on a natural restriction of these problems where the vertices are streamed first, followed by the edge updates. For the st-3Path problem, the corresponding restriction is that the edges incident to v_s and v_t appear before any other edges in the stream; for RootedTriangleCount, it is that the edges incident to v_r appear first.

Under such a restriction on the stream ordering, our two-pass solutions naturally become one-pass, as we now note.

Proposition 5.3. *The schemes for* CrossEdgeCount *and* CrossEdgeCount-Uniq *in Theorem 5.1 and for* InducedEdgeCount, IndSetTest, st-3Path, *and* RootedTriangleCount *in Corollary 5.2 can each be implemented in one pass under a restricted stream ordering as noted above.*

Proof. Consider the protocol described in Section 5.2. Note that the first pass processes only vertices and the second pass processes only edges. This implies the claimed results for CrossEdgeCount, CrossEdgeCount-Uniq, InducedEdgeCount, and IndSetTest. For st-3Path, note that requiring edges incident to v_s and v_t to arrive first is equivalent to the vertex sets $N(v_s)$ and $N(v_t)$ arriving first. A similar consideration applies to RootedTriangleCount. □

It is important to note that despite the restriction on the stream ordering, the schemes in Proposition 5.3 are nontrivial. Without Prover's help, the problems remain hard, even with multiple passes. We give the simple proof for the basic problem CrossEdgeCount.

Proposition 5.4. *Any p-pass streaming algorithm for* CrossEdgeCount, *with vertices streamed before edges, requires $\Omega(n/p)$ space, even for insertion-only streams.*

Proof. We reduce from DISJ_n, the set-disjointness communication problem on the universe $[n]$. Recall that, in DISJ_n, Alice holds a set $x \subseteq [n]$ and Bob holds a set $y \subseteq [n]$. Their goal is to determine whether or not $x \cap y = \emptyset$. This problem has randomized communication complexity $R(\text{DISJ}_n) = \Omega(n)$ [27].

Consider an $(n+1)$-vertex graph G where $V(G) = \{0, \ldots, n\}$ and $E(G) = \{\{0, i\} : i \in y\}$. Let $U = \{0\}$ and $W = x$. Then the number of cross edges in G from U to W is non-zero iff $x \cap y \neq \emptyset$. The result now follows along standard lines. □

6 Path Problems

In this section, we focus on path-related problems. Specifically, we study st-kPath for $k \geqslant 3$ and the fundamental st-ShortestPath problem (defined in Section 1.1). Simple reductions from the INDEX_N problem, for $N = n^2$, show that a one-pass algorithm for either of these problems would require $\Omega(n^2)$ space in the basic (sans prover) streaming model. They also show that a one-pass scheme would require a total cost of $\Omega(n)$. We present a scheme for st-kPath for general k that can also be used to solve st-ShortestPath. It is a semi-streaming scheme when k is polylogarithmic in n, and hence matches the lower bound (up to polylogarithmic factors). Next, we explore if we can break the $\Omega(n)$ barrier for schemes for st-kPath at the cost of allowing a few more passes over the input. We achieve this for constant k by generalizing the protocol for st-3Path. We present all our schemes for undirected graphs, but they can easily be modified to work for directed graphs as well.

6.1 A Single-Pass Semi-Streaming Scheme for Detecting Short Paths

For st-3Path, it is easy to obtain a semi-streaming scheme by checking (using Fact 2.2) whether the set $N[v_s] \times N[v_t]$ and the edge set E are disjoint. For $k > 3$, things are not that direct and we require more work. We describe the protocol below for a multigraph G.

Let A denote the adjacency matrix of the multigraph G and let $\tilde{A}$ be the $\mathbb{F}$-extension of A, for some large finite field $\mathbb{F}$. For $u \in N_{i+1}(v_s)$, let $d_{u,i}$ be the number of (in-)neighbors of u in $N_i(v_s)$. It follows that

$$d_{u,i} = \sum_{v \in N_i(v_s)} A(v, u)\,. \tag{22}$$

We are now ready to describe the protocol.

Stream processing. Verifier picks $r \in_R \mathbb{F}$ and stores $\tilde{A}(v, r)$ for each $v \in [n]$, maintaining them dynamically as the stream arrives (using Fact 2.1). He also stores the set $N_1(v_s)$.

Help message. At the end of the stream, Prover sends Verifier $k-1$ polynomials $\hat{p}_1, \ldots, \hat{p}_{k-1}$, and she claims $\hat{p}_i \equiv p_i$ for each $i \in [k]$, where

$$p_i(U) = \sum_{v \in N_i(v_s)} \tilde{A}(v, U)\,. \tag{23}$$

Verifier's computation. Verifier iteratively constructs $N_i(v_s)$ for $i \in [k]$. Each time, after computing $N_i(v_s)$ for a distance parameter i, he checks whether $v_t \in N_i(v_s)$. If so, he stops and outputs Yes. Otherwise, he proceeds to compute $N_{i+1}(v_s)$. If he finds that $\forall i \in [k] : v_t \notin N_i(v_s)$, then he outputs No. The inductive neighborhood computation is done as follows.

Assume that Verifier has the set $N_i(v_s)$ for some $i \in [k-1]$; this holds initially, since he has stored $N_1(v_s)$. He computes $p_i(r)$ using Equation (23) and checks whether $\hat{p}_i(r) = p_i(r)$. If the check passes, he believes that $\hat{p}_i \equiv p_i$ and evaluates $\hat{p}_i(u)$ for each $u \in V$. By eq. (22), $p_i(u)$ equals $d_{u,i}$, which is non-zero iff $u \in N_{i+1}(v_s)$. Hence, he sets $N_{i+1}(v_s) = \{u : \hat{p}_i(u) \neq 0\}$.

Error probability. The protocol errs when we have $\hat{p}_i \not\equiv p_i$ for some i, but Verifier's check passes. This implies that r is a root of the non-zero polynomial $\hat{p}_i - p_i$. For a given i, the total degree of this polynomial is at most $2n$. Then, probability that r is a root is at most $2n/|\mathbb{F}| < 1/n^2$, for large enough choice of $|\mathbb{F}|$. Taking a union bound over all $i \in [k]$, we get that the probability that r is a root of $\hat{p}_i - p_i$ for some i is at most $1/n$.

Help and Verification costs. Since the degree of each p_i is $\leqslant 2n$, the total hcost is $\tilde{O}(kn)$. Verifier stores $\tilde{A}(v, r)$ for each $v \in [n]$, which requires $\tilde{O}(n)$ space. Additionally, to compute $N_{i+1}(v_s)$ for some $i \in [k]$, he needs only the set $N_i(v_s)$. Thus, we can store the $N_i(v_s)$ sets by reusing space repeatedly, and this requires $O(n)$ space. Hence, the total vcost of this protocol is $\tilde{O}(n)$. Therefore, we get a $[kn, n]$-scheme for checking for the existence of a path of length at most k from v_s to v_t.

Theorem 6.1. *Given an n-vertex (directed or undirected) multigraph $G(V, E)$ and specified vertices $v_s, v_t \in V$, for any $k \leqslant n - 1$, there is a $[kn, n]$-scheme for* st-kPath. *In particular, there is a semi-streaming scheme for* st-kPath *when k is polylogarithmic in n.* □

Applications. Based on the scheme in Theorem 6.1, we have the following straightforward corollaries. Contrast these results with Theorem 7 of Cormode et al. [13]. They give an $[h, v]$-scheme for a *weighted* version of st-ShortestPath for any h, v such that $hv \geqslant Dn^2$ and $h \geqslant Dn$, where D is the maximum distance from v_s to any other vertex reachable from it. A similar result holds for v_s–v_t connectivity in directed graphs with diameter D. Their schemes work only for simple graphs, whereas ours naturally work for multigraphs; on the other hand, we only solve the unweighted version of the problem. Notably, there is a significant difference in the underlying techniques: their schemes are based on linear programming duality, while we have a more directly algebraic approach.

Corollary 6.2. *Given a (directed or undirected) multigraph $G(V, E)$, with edge multiplicities polylogarithmic in n, and specified vertices $v_s, v_t \in V$, there is a $[kn, n]$-scheme for* st-ShortestPath*, where k is length of the shortest v_s–v_t path.*

Proof. If there is no v_s–v_t path, Prover sends the connected component C that v_s is in. Verifier first checks that C is indeed connected ([10], Theorem 7.7). Next, he verifies that there is no edge going out from C by checking whether the set $C \times (V \setminus C)$ and the edge set E are disjoint (Fact 2.2). Both of these are $[n, n]$-schemes.

If there *is* a v_s–v_t path, and the shortest such path H has length k, then Prover sends it to Verifier, who checks whether H is indeed a v_s–v_t path and whether $H \subseteq E$ using an $[n, n]$-scheme, using the polylogarithmic bound on the edge multiplicities (Fact 2.2). In parallel, he uses a $[kn, n]$-scheme to verify that there is no v_s–v_t path of length at most $k - 1$ (Theorem 6.1). □

Corollary 6.3. *Given a directed n-vertex multigraph G, with edge multiplicities polylogarithmic in n, there is a $[Dn, n]$-scheme for checking v_s–v_t connectivity, where D is the maximum distance from v_s to any other vertex reachable from it. In particular, there is a semi-streaming scheme for checking v_s–v_t connectivity in a directed multigraph with diameter polylogarithmic in n.*

Proof. If there is a v_s–v_t path H, then Prover sends it to the Verifier, and he can check whether $H \subseteq E$ using an $[n, n]$-scheme, as edge multiplicity is polylogarithmic in n (Fact 2.2). If not, then we verify that there is no v_s–v_t path of length at most D using a $[Dn, n]$-scheme (Theorem 6.1). □

6.2 A Multi-Pass Scheme for Detecting Short Paths

In Section 5, we obtained a scheme for st-3Path of total cost $o(n)$ using two passes over the input. We investigate if the same is true for st-kPath (for $k > 3$) if we allow "a few" more passes. For constant k, we answer this in the affirmative as we generalize the scheme for st-3Path and obtain such a scheme for st-kPath with $\lceil k/2 \rceil$ passes.

As usual, A denotes the adjacency matrix of the multigraph G. Let L and F be the characteristic vectors of $N[v_s]$ and $N(v_t)$ respectively. Let $\kappa = \kappa(G)$ denote the number of walks of length at most k from v_s to v_t in

G. Then,

$$\kappa = \sum_{u_1,\ldots,u_{k-1}\in V} L_{u_1}\left(\prod_{i=1}^{k-2} A_{u_i,u_{i+1}}\right) F_{u_{k-1}}\,. \tag{24}$$

Note that there is a path of length at most k from v_s to v_t iff $\kappa > 0$. Therefore, computing κ suffices.

Let h and v be integer parameters with $hv = n$. Again, using a canonical bijection, we represent each vertex $u \in V$ by a pair of integers $(x, y) \in [h] \times [v]$. The vectors L and F become 2-dimensional arrays ℓ and f, given by $\ell(x, y) = L_u$ and $f(x, y) = F_u$. Again, the adjacency matrix A turns into a 4-dimensional array a, such that $a(x, y, x', y') = A_{uu'}$. Let $\tilde{\ell}$, $\tilde{f}$, and $\tilde{a}$ be $\mathbb{F}$-extensions of ℓ, f, and a respectively, where $\mathbb{F}$ is a finite field of cardinality q and q is a prime chosen *uniformly at random* from $[n^3, n^4]$. Then eq. (24) gives

$$\kappa = \sum_{x_1,\ldots,x_{k-1}\in[h]} p(x_1, \ldots, x_{k-1})\,, \quad \text{where} \tag{25}$$

$$p(X_1, \ldots, X_{k-1}) = \sum_{y_1,y_2\in[v]} \tilde{\ell}(X_1, y_1)\left(\prod_{i=1}^{k-2} \tilde{a}(X_i, y_i, X_{i+1}, y_{i+1})\right) \tilde{f}(X_{k-1}, y_{k-1})\,. \tag{26}$$

For $i \in [k-1]$, $\deg_{X_i} p = 2h - 2$. Therefore, the number of monomials in p is at most $O(h^{k-1})$ and the total degree is $O(kh)$.

We present a $\lceil k/2 \rceil$-pass protocol for st-kPath.

Stream processing. Verifier chooses $r_1, \ldots, r_{k-1} \in_R \mathbb{F}$.

Pass 1. Process only the vertices in $N_1[v_s]$ and $N_1(v_t)$ in the stream. We maintain, for each $y \in [v]$, two vectors of size v: $\tilde{\ell}(r_1, y)$ and $\tilde{f}(r_{k-1}, y)$, where $y \in [s]$.

Pass i, for $2 \leqslant i \leqslant \lceil k/2 \rceil$. Define $g_0(y) := \tilde{\ell}(r_1, y)$ and $g_k(y) = \tilde{f}(r_{k-1}, y)$. For each $y \in [v]$, compute $g_{i-1}(y) := \sum_{y'\in[v]} \tilde{a}(r_{i-1}, y, r_i, y')g_{i-2}(y')$ as well as $g_{k-i+1}(y) := \sum_{y'\in[v]} \tilde{a}(r_{k-i}, y, r_{k-i+1}, y')g_{k-i+2}(y')$. The $g_j(y)$ values are updated dynamically with the stream updates in a similar way as in the protocol for CrossEdgeCount in Section 5.2.

Help message. At the end of the final pass, Prover sends a polynomial $\hat{p}(X_1, \ldots, X_{k-1})$—as a stream of coefficients—that she claims equals $p(X_1, \ldots, X_{k-1})$.

Verification and output. After the final pass, Verifier computes $\sum_{y\in[v]} g_{\lceil k/2 \rceil}(y)g_{\lceil k/2 \rceil+1}(y)$, which, by Equation (26), equals $p(r_1, \ldots, r_{k-1})$. If he finds that it doesn't equal $\hat{p}(r_1, \ldots, r_{k-1})$, he outputs $\perp$. Otherwise, he believes that $\hat{p} \equiv p$ and, following eq. (25), computes $\hat{\kappa} := \sum_{x_1,\ldots,x_{k-1}\in[h]} \hat{p}(x_1, \ldots, x_{k-1})$. He outputs Yes if $\hat{\kappa} > 0$ and No otherwise.

Error probability. We err when $\hat{p} \not\equiv p$, but Verifier's check passes. In this case, $(r_1, \ldots, r_{k-1}) \in \mathbb{F}^{k-1}$ is a root of the nonzero polynomial $\hat{p} - p$. We noted that its total degree is at most $O(kh)$. By the Schwartz-Zippel Lemma (Fact 2.3), the probability of this event is at most $O(kh)/|\mathbb{F}| < 1/n$. We err also when $\hat{\kappa}$ is non-zero, but the prime q divides $\hat{\kappa}$, making $\hat{\kappa} \bmod q = 0$. But $\hat{\kappa}$ can have value at most $2^n n!$, and so has at most $O(n \log n)$ distinct prime factors. Since we chose q uniformly at random within $[n^3, n^4]$, by the Prime Number Theorem, the probability that q equals one of the prime factors of $\hat{\kappa}$ is at most $1/n^2$. Hence, the total error is at most $1/n + 1/n^2$.

Help and Verification costs. The number of monomials of $\hat{p}$ is $O(h^{k-1})$, giving an hcost of $\tilde{O}(h^{k-1})$. Verifier reuses space and, during each pass, stores $O(1)$ many v-dimensional vectors, each entry of which is $O(\log n)$ bits long. Thus, the vcost is $\tilde{O}(v)$.

This gives a $\lceil k/2 \rceil$-pass $[h^{k-1}, v]$-scheme for st-kPath, for parameters h, v with $hv = n$. Setting $h = n^{1/k}$ and $v = n^{1-1/k}$, we get a scheme with total cost $\tilde{O}(n^{1-1/k})$.

Theorem 6.4. *There is a $\lceil k/2 \rceil$-pass $[n^{1-1/k}, n^{1-1/k}]$-scheme for* st-kPathCount *in a (directed or undirected) multigraph. In particular, for constant k, there is constant-pass scheme with total cost $o(n)$.* □

We note the contrast between this result and that of Guruswami and Onak [17]. They showed a lower bound of $\Omega(n^{1+\Omega(1/k)}/k^{O(1)})$ for st-kPath in $k/2 - 1$ passes in the basic (sans prover) streaming model (for even k). Our results show that using $\lceil k/2 \rceil$ passes, we can obtain a scheme for the same problem with total cost of $\tilde{O}(n^{1-1/k})$.

References

[1] S. Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. In *Proc. 40th Annual ACM Symposium on the Theory of Computing*, pages 731–740, 2008.

[2] A. Abdullah, S. Daruki, C. D. Roy, and S. Venkatasubramanian. Streaming verification of graph properties. In *Proc. 27th International Symposium on Algorithms and Computation*, pages 3:1–3:14, 2016.

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

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

[5] S. Assadi, S. Khanna, and Y. Li. On estimating maximum matching size in graph streams. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1723–1742, 2017.

[6] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 623–632, 2002.

[7] S. K. Bera and A. Chakrabarti. Towards Tighter Space Bounds for Counting Triangles and Other Substructures in Graph Streams. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, pages 11:1–11:14, 2017.

[8] J. Bondy and U. Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[9] A. Chakrabarti, G. Cormode, N. Goyal, and J. Thaler. Annotations for sparse data streams. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 687–706, 2014.

[10] A. Chakrabarti, G. Cormode, A. McGregor, and J. Thaler. Annotations in data streams. *ACM Trans. Alg.*, 11(1):Article 7, 2014.

[11] A. Chakrabarti, G. Cormode, A. McGregor, J. Thaler, and S. Venkatasubramanian. Verifiable stream computation and Arthur-Merlin communication. In *Proc. 30th Annual IEEE Conference on Computational Complexity*, pages 217–243, 2015.

[12] A. Chakrabarti and S. Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1–2):225–247, 2015. Preliminary version in *Proc. 17th Conference on Integer Programming and Combinatorial Optimization*, pages 210–221, 2014.

[13] G. Cormode, M. Mitzenmacher, and J. Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013.

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

[15] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(6):1709–1727, 2008. Preliminary version in *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 745–754, 2005.

[16] A. Goel, M. Kapralov, and S. Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 468–485, 2012.

[17] V. Guruswami and K. Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, Nov. 2016.

[18] M. Jha, C. Seshadhri, and A. Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013.

[19] J. Kallaugher, A. McGregor, E. Price, and S. Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 119–133, 2019.

[20] D. M. Kane, K. Mehlhorn, T. Sauerwald, and H. Sun. Counting arbitrary subgraphs in data streams. In *Automata, Languages, and Programming - 39th International Colloquium, Proceedings, Part II*, pages 598–609, 2012.

[21] M. Kapralov. Better bounds for matchings in the streaming model. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1679–1697, 2013.

[22] H. Klauck and V. Prakash. Streaming computations with a loquacious prover. In *Proc. 4th Conference on Innovations in Theoretical Computer Science*, pages 305–320, 2013.

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

[24] A. McGregor. Finding graph matchings in data streams. In *Proc. 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 170–181, 2005.

[25] A. McGregor, S. Vorotnikova, and H. T. Vu. Better algorithms for counting triangles in data streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 401–411, 2016.

[26] M. Mitzenmacher and J. Thaler. Technical perspective: Catching lies (and mistakes) in offloaded computation. *Commun. ACM*, 59(2):102, 2016.

[27] A. Razborov. On the distributional complexity of disjointness. *Theoretical Comput. Sci.*, 106(2):385–390, 1992. Preliminary version in *Proc. 17th International Colloquium on Automata, Languages and Programming*, pages 249–253, 1990.

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

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

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

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