Streaming Verification for Graph Problems: Optimal Tradeoffs and Nonlinear Sketches

Amit Chakrabarti
Dartmouth College, Hanover, NH, USA

Prantar Ghosh
Dartmouth College, Hanover, NH, USA

Justin Thaler
Georgetown University, Washington, DC, USA

Abstract

We study graph computations in an enhanced data streaming setting, where a space-bounded client reading the edge stream of a massive graph may delegate some of its work to a cloud service. We seek algorithms that allow the client to *verify* a purported *proof* sent by the cloud service that the work done in the cloud is correct. A line of work starting with Chakrabarti et al. (ICALP 2009) has provided such algorithms, which we call *schemes*, for several statistical and graph-theoretic problems, many of which exhibit a tradeoff between the length of the proof and the space used by the streaming verifier.

This work designs new schemes for a number of basic graph problems – including triangle counting, maximum matching, topological sorting, and single-source shortest paths – where past work had either failed to obtain smooth tradeoffs between these two key complexity measures or only obtained suboptimal tradeoffs. Our key innovation is having the verifier compute certain *nonlinear* sketches of the input stream, leading to either new or improved tradeoffs. In many cases, our schemes in fact provide optimal tradeoffs up to logarithmic factors.

Specifically, for most graph problems that we study, it is known that the product of the verifier's space cost v and the proof length h must be at least $\Omega(n^2)$ for n-vertex graphs. However, matching upper bounds are only known for a handful of settings of h and v on the curve $h \cdot v = \tilde{\Theta}(n^2)$. For example, for counting triangles and maximum matching, schemes with costs lying on this curve are only known for $(h = \tilde{O}(n^2), v = \tilde{O}(1))$, $(h = \tilde{O}(n), v = \tilde{O}(n))$, and the trivial $(h = \tilde{O}(1), v = \tilde{O}(n^2))$. A major message of this work is that by exploiting nonlinear sketches, a significant "portion" of costs on the tradeoff curve $h \cdot v = n^2$ can be achieved.

2012 ACM Subject Classification Theory of computation → Streaming models; Theory of computation → Interactive proof systems; Computer systems organization → Cloud computing

Keywords and phrases data streams, interactive proofs, Arthur-Merlin, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.22

Category RANDOM

Related Version A full version of the paper is available at `https://arxiv.org/pdf/2007.03039.pdf`.

Funding *Amit Chakrabarti*: Work supported in part by NSF under award CCF-1907738.
Prantar Ghosh: Work supported in part by NSF under award CCF-1907738.
Justin Thaler: Work supported by NSF SPX award CCF-1918989, and NSF CAREER award CCF-1845125. Parts of this work were performed while visiting the Simons Institute for the Theory of Computing.

Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020).
Editors: Jarosław Byrka and Raghu Meka; Article No. 22; pp. 22:1–22:23
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

It is far easier to verify a proof than to find one. This intuitively clear fact has been given precise meanings in several settings , leading to such landmark results as the IP = PSPACE [36] and PCP Theorems [3, 4]. There is a growing body of work on results of this flavor for space-efficient computations on large data streams [35]. In this setting, a space-bounded client (henceforth named Verifier) that can only process inputs in the restrictive *data streaming* setting has access to a computationally powerful entity (henceforth named Prover), such as cloud computing service, that has no such space limitations. As past work has shown, many fundamental problems that are intractable in the plain data-streaming model – in the sense that they cannot be solved using sublinear space – do admit nontrivial solutions in this Verifier/Prover model, *without* Verifier having to trust Prover blindly.

An algorithm in this model specifies a protocol to be followed by Verifier and Prover so that the former may compute some function $f(\sigma)$ of the input stream σ. Prover, by performing the specified actions honestly, convinces Verifier to output the correct value $f(\sigma)$. However, if Prover fails to follow the protocol, whether out of malice or error (modeling hardware, software, or network faults in the cloud service), then Verifier is highly likely to detect this and *reject*. Past work has considered a few different instances of this setup, such as (a) *annotated* data streaming algorithms [12, 30] – also called *online schemes* – where the parties read σ together and the protocol consists of Prover streaming a "help message" (a.k.a. proof) to Verifier either during stream processing and/or at the end; (b) *prescient schemes* [11], which are a variant of the above where Prover knows all of σ before Verifier sees it; (c) *streaming interactive proofs* (SIPs) [13, 19], where Verifier and Prover engage in multiple rounds of communication.

This work focuses on the first and arguably best-motivated of these models, namely, online schemes. We simply call them *schemes*. We give new and improved schemes for several graph-theoretic problems, including triangle counting, maximum matching, topological sorting, and shortest paths. In all cases, the input is a huge n-vertex graph G given as a stream σ of edge insertions and/or deletions. While most of our problems have been studied before, we give schemes that (a) have better complexity parameters, in some cases achieving optimality, and (b) use cleverer algebraic encodings of the relevant combinatorial problems, often exploiting the ability of a streaming algorithm to compute *nonlinear* sketches.

1.1 Setup, Terminology, and Motivation

We formalize the setup described above. A scheme for a function f specifies three things: (i) a space-bounded data streaming algorithm used by Verifier to process the input σ and compute a summary $\mathcal{V}_R(\sigma)$, using random coins R; (ii) a help function used by Prover to send a message $\mathcal{H}(\sigma)$ to Verifier as a "proof stream" after the input stream ends;[1] and (iii) an output algorithm $\mathrm{out}_R(\mathcal{V}_R(\sigma), \mathcal{H}(\sigma))$ capturing Verifier's work during and after the proof stream, which produces values in $\mathrm{range}(f) \cup \{\perp\}$, where an output of $\perp$ indicates "reject." If $\mathcal{V}_R$ and out_R run in $O(v)$ bits of space and $\mathcal{H}$ provides $O(h)$ bits of help, then this scheme is called an (h, v)-*scheme*. 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 $h = 0$. A scheme is said to have

- completeness error ε_c if $\forall\sigma\, \exists\mathcal{H} : \Pr_R[\mathrm{out}_R(\mathcal{V}_R(\sigma), \mathcal{H}(\sigma)) = f(\sigma)] \geqslant 1 - \varepsilon_c$;
- soundness error ε_s if $\forall\sigma, \mathcal{H}' : \Pr_R[\mathrm{out}_R(\mathcal{V}_R(\sigma), \mathcal{H}'(\sigma)) \notin \{f(\sigma), \perp\}] \leqslant \varepsilon_s$.

[1] A more general (though seldom used) model allows Prover to send help messages after each data item in σ.

In designing schemes, we will aim for $\varepsilon_s \leqslant 1/3$, which can be reduced further via parallel repetition in standard ways. We will also achieve perfect completeness, i.e., $\varepsilon_c = 0$. For an (h, v)-scheme we refer to h as its hcost (short for "help cost") and v as its vcost ("verification cost"). We use the notation $[h, v]$-scheme as a shorthand for an $(\tilde{O}(h), \tilde{O}(v))$-scheme.[2]

It is intuitive that the parameters h and v are in tension, suggesting that they can be traded off against one another. Most of our algorithms do obtain such tradeoffs. We emphasize that actually obtaining a smooth tradeoff for large ranges of h and v values is not automatic: indeed, an important contribution of this work is to obtain such tradeoffs for problems where past work gave comparable results only for specific settings of h and v.

When studying the results discussed below, it is useful to keep a few cost regimes in mind. We focus on graph problems on n-vertex inputs. An (h, v)-scheme for such a problem is *sublinear* if $h = o(n^2)$ and $v = o(n^2)$; *frugal* if it is sublinear and achieves the stronger guarantee $v = o(n)$; and *laconic* if it is sublinear and achieves the stronger guarantee $h = o(n)$.

Many graph problems are *intractable* in the basic one-pass streaming model, meaning that they provably require $\Omega(n^2)$ space. Past work [12] implies that any (h, v)-scheme for such a problem must have $hv = \Omega(n^2)$. Thus, an $[h, v]$-scheme with $hv = O(n^2)$ for an intractable problem has achieved an optimal tradeoff, up to logarithmic factors. All of the problems we consider in this paper (except for counting connected components) are intractable for dense graphs (i.e., graphs with $\Omega(n^2)$ edges).

Frugal schemes are important when Verifier is so starved for space that it cannot afford to store even a constant fraction of the vertices. They are also very interesting from a theoretical standpoint, since even "easy" graph problems require at least $\Omega(n)$ space in the basic streaming model. On the other hand, laconic schemes are naturally motivated by settings where Verifier does not have streaming access to the proof and has to store it in full. Consider for example a retail client that uploads transactions to the cloud as they occur. It makes sense to have uploaded even terabytes of information *in total* over a long period of time: days, months, or years. However, it might not be reasonable for the cloud to transfer a proof consisting of, say, tens of gigabytes to the client. From a theoretical standpoint, in solving an intractable problem, if Verifier has to store the proof, there is no reason to ever try to reduce vcost to $o(n)$, since hcost will then blow up to $\omega(n)$.

1.2 Problems, Results, and Comparisons with Related Work

Throughout, the input graph G will be on the fixed vertex set $V = [n] := \{1, \ldots, n\}$ and will have m edges. Many results will be stated in terms of tunable parameters $t, s \in \mathbb{Z}^+$ that must satisfy $ts \geqslant n$. Since bounds are asymptotic, this condition can be read as $ts = n$.

Triangle Counting. Our starting point is the triangle counting problem (henceforth, TriangleCount), studied heavily in past work on graph streaming [7, 8, 10, 24, 25, 27, 34, 38]. Given a multigraph G as a dynamic stream (i.e., insertions and deletions), the goal is to compute T, the number of triangles in G. The exact counting version studied in this paper is an intractable problem in the sense of Section 1.1: it requires $\Omega(n^2)$ space in basic streaming.

As noted in Table 1, we give several new algorithms for TriangleCount. Our $[nt^2, s]$-scheme improves upon the best known frugal scheme for the problem: for a fixed hcost $\geqslant n$, it improves the vcost from $v^{4/3}$ to v, and for a fixed vcost $v \leqslant n$, it improves the hcost from $n^3/v^{3/2}$ to n^3/v^2. Our $[t, ns]$-scheme is not only the first laconic scheme for the problem but

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

Table 1 Summary of results on the problems considered in this paper. A scheme is deemed *optimal* if it has help cost at most h and space cost at most v for at least one pair h, v such that $h \cdot v \leqslant \tilde{O}(L)$, whereas it is known that *any* (h, v) scheme that applies to all graphs requires $h \cdot v \geqslant \Omega(L)$. A blank space in the *Tradeoff* column indicates that it remains open whether the scheme can be strictly improved. Here, α' is the size of a maximum matching in the input graph, K is the length of a shortest v_s–v_t path, D is the maximum distance from the source to the any other reachable vertex, and W is the maximum weight of an edge.

Problem	Scheme	Tradeoff	Reference
TriangleCount	$[h, v]$; $hv = n^3$	Suboptimal	[12]
	$[n^2, 1]$	Optimal	[12]
	$[n, n]$	Optimal	[38]
	$[t^3, s^2]$; $ts = n$	Suboptimal	[14]
	$[nt^2, s]$; $ts = n$		Theorem 4
	$[t, ns]$; $ts = n$	Optimal	Theorem 5
	$[mn/\sqrt{v},\ v]$	Suboptimal	[11]
	$[m + h, v]$; $hv = n^2$		Theorem 10
TriangleCount-Adj	$[h, v]$; $hv = n^2$		Theorem 11
MaxMatching	$[m, 1]$	Optimal	[18]
	$[n, n]$	Optimal	[38]
	$[t^3, s^2]$; $ts = n$	Suboptimal	[14]
	$[nt, s]$; $ts = n$	Optimal	Theorem 8
	$[\alpha' + h, v]$; $hv = n^2$		Theorem 9
MIS	$[nt, s]$; $ts = n$	Optimal	Theorem 12
Acyclicity/TopoSort	$[m, 1]$	Optimal	[18]
	$[nt, s]$; $ts = n$	Optimal	Theorem 13; Corollary 14
st-ShortestPath	$[Dnt, s]$; $ts = n$		[18]
	$[Kn, n]$		[14]
	$[Knt, s]$; $ts = n$		Corollary 18
Unweighted SSSP	$[Dnt, s]$; $ts = n$		Theorem 17
Weighted SSSP	$[m + n, 1]$	Optimal	[18]
	$[DWn, n]$		Theorem 19
	$[Dn, Wn]$		Theorem 20

also achieves smooth optimal tradeoff in its parameter range; thus, it settles the complexity of the problem in the laconic regime. The $[m+h, v]$-scheme whenever $hv = n^2$ generalizes the $[n^2, 1]$-scheme from prior work for any m-edge graph (for the setting $h = m$ and $v = n^2/m$) and is interesting in the frugal regime for sparse graphs.

The problem has also been studied in the adjacency-list model (call it TriangleCount-Adj) [10, 26, 31, 34], where the stream presents the full neighbor list for each vertex contiguously. We give an $[h, v]$-scheme for any $hv = n^2$ for TriangleCount-Adj (again, exact counting). In basic streaming, there is no nontrivial algorithm for computing T exactly, or even approximately when T is small; in fact, under a long-standing conjecture in communication complexity, these problems require $\Omega(m)$ space [26].

Maximum Matching. There is a recent and ongoing flurry of activity on streaming algorithms for MAXMATCHING, the problem of computing the cardinality $\alpha'(G)$ of a maximum-sized matching[3] in G [6, 16, 21, 22, 28, 33, 20, 29]. The exact version of the problem (which is what we study here) is intractable. For the special case of detecting whether a bipartite graph has a perfect matching, there is a frugal $[nt, s]$-scheme [12], which achieves optimal tradeoff. See Table 1 for previous results for the general problem.

In this work, we give (i) the first optimal frugal $[nt, s]$-scheme for the general MAXMATCHING problem, settling its complexity in the frugal regime, and (ii) an $[\alpha' + h, v]$-scheme whenever $hv = n^2$, which yields a laconic scheme provided $\alpha'(G) = o(n)$. Obtaining a fully general laconic scheme remains an interesting open problem and we suspect that it will require a breakthrough in exploiting the problem's combinatorial structure.

Further Graph Problems and a Common Framework. We obtain new schemes for the MIS problem, which asks for an inclusion-wise maximal independent set of vertices; the ACYCLICITY problem, which asks whether the input digraph is acyclic; and the TOPOSORT problem, which asks for a vertex ordering of the input DAG that orients all edges "forwards." In each case, we give an $[nt, s]$-scheme. Recent results show that MIS [5, 17] and TOPOSORT [15] are intractable in basic streaming, so our schemes are optimal in the frugal regime. Importantly, these schemes, the frugal MAXMATCHING scheme, and two of the TRIANGLECOUNT schemes all fit a common framework: they boil the problem down to counting the number of edges in one or more induced subgraphs of the input graph. Our scheme for this INDUCEDEDGECOUNT problem could be a useful technical result for future work.

Shortest Paths. The single-source shortest path (SSSP) problem is perhaps the most basic problem in classic graph algorithms. In the streaming setting, even the special case of undirected v_s–v_t connectivity in constant-diameter graphs is intractable [21]. As Table 1 shows, our $[Dnt, s]$-scheme for unweighted SSSP (where D is the maximum distance from the source vertex v_s to any vertex reachable from it) generalizes the result of Cormode et al. [18] from ST-SHORTESTPATH to SSSP. Again, as a corollary, we obtain a $[Knt, s]$-scheme for ST-SHORTESTPATH, where K is the length of a shortest v_s–v_t path. This result generalizes the $[Kn, n]$-scheme of Chakrabarti and Ghosh [14] and improves upon the $[Dnt, s]$-scheme of Cormode et al. [18], since K can be arbitrarily smaller than D. The schemes for the weighted version are interesting for small D and W, where W is the maximum weight of any edge.

1.3 Other Related Works

Abdullah et al. [1] studied the TRIANGLECOUNT and MAXMATCHING problems in the stronger SIP model that allows rounds of interaction between Prover and Verifier. For TRIANGLECOUNT, they gave a $(\log^2 n, \log^2 n)$-SIP using $\log n$ rounds of interaction. They also designed an $(n^{1/\gamma} \log n, \log n)$-SIP with $\gamma = O(1)$ rounds. For the weighted MAXMATCHING problem, they gave a $(\rho + n^{1/\gamma'} \log n, \log n)$-SIP using γ rounds of interaction, where γ' is a linear function of γ, and ρ is the weight of an optimal matching.

Early works on the concept of annotated streams include Tucker et al. [39] and Yi et al. [41], who studied *stream punctuations* and *stream outsourcing* respectively. Motivated by these works, Chakrabarti et al. [12] then formalised the model theoretically as the *annotated streaming model* and gave schemes for statistical streaming problems including

[3] The notation $\alpha'(G)$ is by analogy with $\alpha(G)$, which denotes the cardinality of a maximum independent set of *vertices*. It can be found, e.g., in the textbook by West [40].

frequency moments and heavy hitters, along with some basic results for graph problems. This non-interactive model was subsequently studied by multiple works including Klauck and Prakash [30], Cormode et al. [18], and Chakrabarti et al. [11]. Subsequent works considered generalized versions of the model, allowing rounds of interaction. These include *Arthur-Merlin streaming protocols* of Gur and Raz [23] and the *streaming interactive proofs* (SIP) of Cormode et al. [19]. Chakrabarti et al. [13] and Abdullah et al. [1] further studied this generalized setting. We refer to the expository article of Thaler [37] for a more detailed survey of this area.

1.4 Our Techniques

Sum-Check and Polynomial Encodings. As with much prior work in this area (and probabilistic proof systems more generally), our schemes are variants of the famous *sum-check protocol* of Lund et al. [32]. Specialized to our (non-interactive) schemes, this protocol allows Verifier to make Prover honestly compute $\sum_{x \in \mathcal{X}} g(x)$ for some low-degree polynomial $g(X)$ derived from the input data and some designated set $\mathcal{X}$. Verifier has no space to compute g explicitly, nor all values $\langle g(x) : x \in \mathcal{X} \rangle$, but he can afford to evaluate $g(r)$ at a *random* point r. The Prover steps in by explicitly providing $\hat{g}(X)$, a polynomial claimed to equal $g(X)$: this is cheap since g has low degree. Verifier can be convinced of this claim by checking that $\hat{g}(r) = g(r)$.

Hence, the main challenge in applying the sum-check technique is to find a way to encode the data stream problem's output as the sum of the evaluations of a low-degree polynomial g so that Verifier can, in small space, evaluate g at a random point r.

Sketches: Linearity and Beyond. A streaming Verifier evaluates $g(r)$ by suitably summarizing the input in a *sketch*. Viewing the input as updates to a data vector $\mathbf{f} = (f_1, \ldots, f_N)$, such a sketch $\mathbf{v}$ is *linear* if $\mathbf{v} = S\mathbf{f}$ for some matrix $S \in \mathbb{F}^{v \times N}$, for some field $\mathbb{F}$.[4] Typically, S is implicit in the sketching algorithm and enables stream processing in $\tilde{O}(v)$ space by translating a stream update $f_i \leftarrow f_i + \Delta$ into the sketch update $\mathbf{v} \leftarrow \mathbf{v} + \Delta S \mathbf{e}_i$, where $\mathbf{e}_i$ is the ith standard basis vector. In essentially all prior works on stream verification, the polynomial g was such that $g(r)$ could be derived from such a linear sketch $\mathbf{v}$.

There is one exception: Thaler [38] introduced an optimal $[n, n]$-scheme for TRIANGLE-COUNT in which Verifier computes a *nonlinear* sketch.[5] Roughly speaking, the verifier in Thaler's protocol maintains two n-dimensional linear sketches $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$, plus a value C that is *not* a linear function of the input stream but instead depends quadratically on $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$. Moreover, the jth increment to C uses information that is available while processing the jth stream update, but not after the stream is gone. This is in contrast to linear sketches themselves, where the jth sketch update depends only on the jth stream update and no others.

The Shaping Technique. Another ubiquitous idea in streaming verification is the *shaping technique*, which transforms a data vector into a multidimensional array. This trick realizes $g(X)$ as a summation of an even simpler multivariate polynomial: the latter can be evaluated directly by Verifier at several points, which forms the basis for his sketching. When applied to graph problems, this technique was historically used to reshape the $\binom{n}{2}$-dimensional vector

[4] This field is finite in the streaming verification literature, whereas traditional data streaming uses $\mathbb{R}$.
[5] Simliar nonlinearity was used recently in the more powerful model of 2-*pass schemes* [14].

of edge multiplicities. Recently, Chakrabarti and Ghosh [14] introduced the idea of reshaping the graph's *vertex space*, rather than just the edge space, thereby transforming the adjacency matrix into a 4-dimensional array. This trick was crucial to obtaining the first frugal schemes for TriangleCount and MaxMatching.

Our Contributions. The new schemes in this work make the following contributions.

- We design new polynomial encodings for the graph-theoretic problems we study.
- We prominently employ nonlinear sketches, in the above sense, for almost all of our scheme designs.
- We use the shaping technique on the vertex space, often combining it with nonlinear sketching, thus expanding the applications of this very recent innovation.

Our solutions for TriangleCount are particularly good illustrations of all of these ideas. Where Thaler's nonlinear-sketch protocol treated each vertex as monolithic, our view of each vertex as an object in $[t] \times [s]$ (for some pair t, s with $t \cdot s = n$) let us do two things. In the laconic regime, we get to use Verifier's increased space allowance in a way that Thaler's protocol cannot, thereby extending his $[n, n]$-scheme to get an optimal tradeoff. In the frugal regime, it is significantly harder to exploit vertex-space shaping because Verifier cannot even afford to devote one entry per vertex in his linear sketches. We overcome this by finding a way for many vertices to "share" each entry of each linear sketch (see the string of equations culminating in Equation (7)), thus extending Thaler's protocol to smoothly trade off communication for space.

We also extend the applicability of nonlinear sketching by identifying many further graph problems for which it yields significant improvements. Specifically, in Section 3, we describe two technical problems called InducedEdgeCountand CrossEdgeCount, which are later used as primitives to optimally solve several important graph problems, including MaxMatching. We show how to apply sum-check with a nonlinear Verifier (see, e.g, Equation (11)) to optimally solve InducedEdgeCount and CrossEdgeCount.

Finally, our schemes for SSSP feature a different kind of innovation on top of vertex-space shaping and new, clever encodings of shortest-path problems in a manner amenable to sum-check. They overcome the frugal Verifier's space limitation by exploiting the Prover's room to generate a proof stream that mimics an iterative algorithm. For the Verifier to play along with such an iterative algorithm while lacking even one bit of space per vertex, a careful layering of fingerprint-based checks is needed on top of the sum-checks. We hope that our work here opens up possibilities for other instances of porting iterative algorithms to a streaming setting with the help of a prover.

1.5 Preliminaries

In this work, the input graph, multigraph, or digraph is denoted G and defined on a fixed vertex set $V = [n]$. In the *vanilla* streaming model, G is given as a stream of (u, v) tokens, where $u, v \in V$: the token is interpreted as an insertion of edge $\{u, v\}$ or directed edge (u, v). If G is edge-weighted, the tokens are of the form (u, v, w), where $w \in \mathbb{Z}^+$ is a weight. In the *turnstile* streaming model, tokens are of the form (u, v, Δ), denoting that the quantity $\Delta \in \mathbb{Z}$ (which can be negative) is added either to the multiplicity or the weight of the edge $\{u, v\}$.

An important primitive in all our schemes is sketching a data vector by evaluating its low-degree extension at a random point. Let us explain what this means. Suppose our data vector, which has dimensionality N, is shaped into a k-dimensional array f with dimensions $(s_1, \ldots, s_k)$, where $s_1 s_2 \cdots s_k \geqslant N$. Equivalently, we have a function f on domain $[s_1] \times \cdots \times [s_k]$. We work over a suitable finite field[6] $\mathbb{F}$. By Lagrange interpolation, there is a 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 $\deg_{X_i} \tilde{f} \leqslant s_i - 1$.

We call $\tilde{f}$ the low-degree $\mathbb{F}$-*extension* of f. Since $f \mapsto \tilde{f}$ is a linear map, we can write $\tilde{f}$ as a linear combination of "unit impulse" functions (also known as Lagrange basis polynomials):

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

To be precise, $\tilde{f}(X_1, \ldots, X_k) = \sum_{(u_1,\ldots,u_k) \in [s_1] \times \cdots \times [s_k]} f(u_1, \ldots, u_k)\, \delta_{u_1,\ldots,u_k}(X_1, \ldots, X_k)$. In particular, if f is built up from a stream of pointwise updates, where the jth update adds Δ_j to entry $(u_1, \ldots, u_k)_j$ of the array, then

$$\tilde{f}(X_1, \ldots, X_k) = \sum_j \Delta_j\, \delta_{(u_1,\ldots,u_k)_j}(X_1, \ldots, X_k)\,. \tag{2}$$

▶ **Fact 1.** *Given $\mathbf{p} = (p_1, \ldots, p_k) \in \mathbb{F}^k$ and a stream of pointwise updates to an initially-zero array with dimensions $(s_1, \ldots, s_k)$, we can maintain the evaluation $\tilde{f}(\mathbf{p})$ using $O(\log |\mathbb{F}|)$ space, performing $O(k)$ field arithmetic operations per update. In applications, we usually take $\mathbf{p} \in_R \mathbb{F}^k$.*[7] *For details and implementation considerations, see Cormode et al. [19].* ◀

Another useful primitive is *fingerprinting*, used prominently in our SSSP scheme and subtly in subroutines within other schemes. Its goal is to check equality between two vectors $\mathbf{a} = (a_1, \ldots, a_N)$ and $\mathbf{b} = (b_1, \ldots, b_N)$ that are provided via turnstile streams in some possibly intermixed order. This is achieved by checking that $\varphi_{\mathbf{a}}(r) = \varphi_{\mathbf{b}}(r)$ for $r \in_R \mathbb{F}$, where $\varphi_{\mathbf{a}}(X) = \sum_{j=1}^{N} a_j X^j$ is the *fingerprint polynomial* of $\mathbf{a}$ and has degree at most N. Both fingerprinting and the eventual uses of Fact 1 in sum-check protocols depend upon the following basic but powerful result.

▶ **Fact 2** (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, $\Pr_{(r_1,\ldots,r_n) \in_R \mathbb{F}^n}[P(r_1, \ldots, r_n) = 0] \leqslant d/|\mathbb{F}|$.* ◀

At various points, we shall use a couple of schemes from Chakrabarti et al. [11, 12].

▶ **Fact 3** (SUBSET and INTERSECTION schemes; Prop. 4.1 of [12] and Thm. 5.3 of [11])**.** *Given a stream of elements of sets $S, T \subseteq [N]$ interleaved arbitrarily, for any h, v with $hv \geqslant N$, there are $[h, v]$-schemes to compute $|S \cap T|$ and to determine whether $S \subseteq T$.* ◀

2 The Triangle Counting Problem

A triangle in a (multi)graph is a set of three edges of the form $\{\{u, v\}, \{v, w\}, \{u, w\}\}$. The TRIANGLECOUNT problem asks for the number of such triangles in the input graph. We solve this problem for multigraphs given by a turnstile stream, establishing the following two theorems. The first gives improved (but possibly still not tight) tradeoffs between hcost h and vcost v in the parameter regime where $h \geqslant n$ and $v \leqslant n$. The second gives *optimal*

[6] The characteristic of $\mathbb{F}$ must be large enough to avoid "wrap around" problems under arithmetic in $\mathbb{F}$.

[7] The notation $r \in_R A$ means that r is drawn uniformly at random from the finite set A.

tradeoffs (up to logarithmic factors) in the regime where $h \leqslant n$ and $v \geqslant n$, based on the known lower bound that hv must be $\Omega(n^2)$. Both results were previously only known when $h = \Theta(n)$.

We remind the reader that parameters $t, s \in \mathbb{Z}^+$ are tunable, subject to $ts = n$.

▶ **Theorem 4** (Improved frugal schemes). *There is an $[nt^2, s]$-scheme for* TriangleCount*.*

▶ **Theorem 5** (Optimal tradeoff for laconic schemes). *There is a $[t, ns]$-scheme for* TriangleCount*. This is optimal up to logarithmic factors.*

Overview of Our Methods. Consider an adjacency matrix A of a graph on vertex set V. The addition of a new edge $\{u, v\}$ creates $\sum_{z \in V} A(u, z)A(v, z)$ new triangles.

Suppose that the input stream consists of L edge updates, the jth being $(v_{1j}, v_{2j}, \Delta_j)$; recall that its effect is to add Δ_j to the multiplicity of edge $\{v_{1j}, v_{2j}\}$. Suppose that the cumulative effect of the first j updates is to produce a multigraph G_j whose adjacency matrix is A_j and which has T_j triangles (counting multiplicity). As in Thaler's protocol [38], we can then account for the number of triangles added by the jth update:

$$T_j - T_{j-1} = \sum_{v_3 \in V} \Delta_j \, A_{j-1}(v_{1j}, v_3) \, A_{j-1}(v_{2j}, v_3) \, .$$

As a result, the number of triangles T in the final graph $G = G_L$ is

$$T = \sum_{j \in [L]} \sum_{v_3 \in V} \Delta_j \, A_{j-1}(v_{1j}, v_3) \, A_{j-1}(v_{2j}, v_3) \, . \tag{3}$$

Our two new families of schemes for TriangleCount apply the shaping technique to the above equation in two distinct ways, resulting in markedly different complexity behaviors.

The Laconic Schemes Regime (Proof of Theorem 5). Let $t, s \in \mathbb{N}$ be parameters with $ts = n$. We first consider rewriting the variable v_3 in Equation (3) as a pair of integers $(x_3, y_3) \in [t] \times [s]$ using some canonical bijection. This shapes each matrix A_{j-1} into a 3-dimensional array a_{j-1}, i.e., a function with domain $[n] \times [t] \times [s]$. Let $\tilde{a}$ be the $\mathbb{F}$-extension of a for a sufficiently large finite field $\mathbb{F}$ to be chosen later. Then Equation (3) becomes

$$T = \sum_{j \in [L]} \sum_{x_3 \in [t]} \sum_{y_3 \in [s]} \Delta_j \, \tilde{a}_{j-1}(v_{1j}, x_3, y_3) \, \tilde{a}_{j-1}(v_{2j}, x_3, y_3) = \sum_{x_3 \in [t]} p(x_3) \, , \quad \text{where} \tag{4}$$

$$p(X_3) = \sum_{j \in [L]} \sum_{y_3 \in [s]} \Delta_j \, \tilde{a}_{j-1}(v_{1j}, X_3, y_3) \, \tilde{a}_{j-1}(v_{2j}, X_3, y_3) \, . \tag{5}$$

By the properties of $\mathbb{F}$-extensions observed above, we have the bound $\deg p \leqslant 2(t-1)$. We now design our scheme as follows.

Stream processing. Verifier starts by picking $r_3 \in_R \mathbb{F}$. As the stream arrives, he maintains a 2-dimensional array of values $\tilde{a}_{j-1}(v, r_3, y)$, for all $(v, y) \in [n] \times [s]$, using Fact 1. He also maintains an accumulator that starts at zero and, after the jth update, is incremented by $\Delta_j \sum_{y_3 \in [s]} \tilde{a}_{j-1}(v_{1j}, r_3, y_3) \, \tilde{a}_{j-1}(v_{2j}, r_3, y_3)$. By Equation (5), the final value of this accumulator is $p(r_3)$.

Help message. Prover sends Verifier a polynomial $\hat{p}(X_3)$ of degree $\leqslant 2(t-1)$ that she claims equals $p(X_3)$.

Verification and output. Using Prover's message, Verifier computes the check value $C := \hat{p}(r_3)$ and the result value $\hat{T} := \sum_{x_3 \in [t]} \hat{p}(x_3)$. If he finds that $C \neq p(r_3)$, he outputs $\perp$. Otherwise, he believes that $\hat{p} \equiv p$ and accordingly, based on Equation (4), outputs $\hat{T}$ as the answer.

The analysis of this scheme proceeds along standard lines long established in the literature.

Error probability. An honest Prover ($\hat{p} \equiv p$) clearly ensures perfect completeness. The soundness error is the probability that Verifier's check passes despite $\hat{p} \not\equiv p$, i.e., that the random point $r_3 \in \mathbb{F}$ is a root of the nonzero degree-$(2t-2)$ polynomial $\hat{p} - p$. By the Schwartz–Zippel Lemma (Fact 2), this probability is at most $(2t-2)/|\mathbb{F}| < 1/n$, by choosing $|\mathbb{F}|$ large enough.

Help and Verification costs. Prover describes $\hat{p}$ by listing its $O(t)$ many coefficients, spending $O(t \log n)$ bits, since each is an element of $\mathbb{F}$ and $|\mathbb{F}| = n^{O(1)}$ suffices above. Verifier maintains an $n \times s$ array whose entries are in $\mathbb{F}$, for a vcost of $O(ns \log n)$. Overall, we get a $[t, ns]$-scheme, as required.

The Frugal Schemes Regime (Proof of Theorem 4). Designing frugal schemes on the basis of Equation (3) is more intricate. This time we rewrite the variables v_{1j} and v_{2j} as pairs (x_{1j}, y_{1j}) and (x_{2j}, y_{2j}), each in $[t] \times [s]$ for parameters t, s with $ts = n$. The matrices A_{j-1} are now shaped into 3-dimensional arrays b_{j-1} that can be seen as functions on the domain $[t] \times [s] \times [n]$. As before, let $\tilde{b}$ be an appropriate $\mathbb{F}$-extension. Working from Equation (3) and cleverly using the "unit impulse" function δ seen in Equation (1),

$$\begin{aligned} T &= \sum_{v_3 \in V} \sum_{j \in [L]} \Delta_j \, \tilde{b}_{j-1}(x_{1j}, y_{1j}, v_3) \, \tilde{b}_{j-1}(x_{2j}, y_{2j}, v_3) \\ &= \sum_{v_3 \in V} \sum_{w_1, w_2 \in [t]} \sum_{j \in [L]} \Delta_j \, \tilde{b}_{j-1}(w_1, y_{1j}, v_3) \, \tilde{b}_{j-1}(w_2, y_{2j}, v_3) \, \delta_{x_{1j}}(w_1) \, \delta_{x_{2j}}(w_2) \\ &= \sum_{v_3 \in V} \sum_{w_1, w_2 \in [t]} q(w_1, w_2, v_3), \quad \text{where} \end{aligned} \tag{6}$$

$$q(W_1, W_2, V_3) = \sum_{j \in [L]} \Delta_j \, \tilde{b}_{j-1}(W_1, y_{1j}, V_3) \, \tilde{b}_{j-1}(W_2, y_{2j}, V_3) \, \delta_{x_{1j}}(W_1) \, \delta_{x_{2j}}(W_2) \,. \tag{7}$$

In contrast to the laconic case, we have a *multivariate* polynomial $q(W_1, W_2, V_3)$. We have the bounds $\deg_{W_1} q \leqslant 2(t-1)$, $\deg_{W_2} q \leqslant 2(t-1)$, and $\deg_{V_3} q \leqslant 2(n-1)$, for a total degree of $O(t+n) = O(n)$. Importantly, the number of monomials in q is at most $(2t-1)^2(2n-1) = O(nt^2)$. We now present the corresponding scheme and its analysis.

Stream processing. Verifier picks $r_1, r_2, r_3 \in_R \mathbb{F}$. As the stream arrives, he maintains two 1-dimensional arrays: $\tilde{b}_{j-1}(r_1, y, r_3)$ and $\tilde{b}_{j-1}(r_2, y, r_3)$, for all $y \in [s]$ (using Fact 1). He also maintains an accumulator that starts at zero and, after the jth update $(x_{1j}, y_{1j}, x_{2j}, y_{2j})$, is incremented by $\Delta_j \tilde{b}_{j-1}(r_1, y_{1j}, r_3) \tilde{b}_{j-1}(r_2, y_{2j}, r_3) \delta_{x_{1j}}(r_1) \delta_{x_{2j}}(r_2)$. By Equation (7), the final value of this accumulator is $q(r_1, r_2, r_3)$.

Notice that the accumulator is a nonlinear sketch of the input.

Help message. Prover sends Verifier a polynomial $\hat{q}(W_1, W_2, V_3)$ that she claims equals $q(W_1, W_2, V_3)$. It should satisfy the degree bounds noted above. He lacks the space to store $\hat{q}$, so she streams the coefficients of $\hat{q}$ in some canonical order.

Verification and output. As $\hat{q}$ is streamed in, Verifier computes the check value $C := \hat{q}(r_1, r_2, r_3)$ and the result value $\hat{T} := \sum_{v_3 \in [n]} \sum_{w_1, w_2 \in [t]} \hat{q}(w_1, w_2, v_3)$. If he finds that $C \neq q(r_1, r_2, r_3)$, he outputs $\perp$. Otherwise, he believes that $\hat{q} \equiv q$ and accordingly, based on Equation (6), outputs $\hat{T}$ as the answer.

Error probability. As before, we have perfect completeness and by the Schwartz–Zippel Lemma (Fact 2, this time using its full multivariate strength), this soundness error is at most $\deg q / |\mathbb{F}| = O(n)/|\mathbb{F}| < 1/n$, by choosing $|\mathbb{F}|$ large enough.

Help and Verification costs. Prover can describe $\hat{q}$ by listing its $O(nt^2)$ coefficients. Verifier maintains two s-length arrays. Overall, we get an $[nt^2, s]$-scheme, as required.

3 A Technical Result: Counting Edges in Induced Subgraphs

We introduce two somewhat technical, though still natural, graph problems: InducedEdgeCount and CrossEdgeCount. We design schemes for these problems giving optimal tradeoffs (as usual, up to logarithmic factors). These schemes are key subroutines in our schemes for more standard, well-studied graph problems – such as MaxMatching– considered in Section 4.

The InducedEdgeCount problem is defined as follows. The input is a stream of edges of a graph $G = (V, E)$ followed by a stream of vertex subsets $\langle U_1, \dots U_\ell\rangle$ for some $\ell \in \mathbb{N}$, where $U_i \subseteq V$ for $i \in [\ell]$. To be precise, the latter portion of the stream consists of the vertices of U_1 in arbitrary order, followed by a delimiter, followed by the vertices of U_2 in arbitrary order, and so on. The desired output is $\sum_{i=1}^{\ell} |E(G[U_i])|$, the sum of the numbers of edges in the induced subgraphs $G[U_1], \dots, G[U_\ell]$. Note that $U_1, \dots, U_\ell$ need not be pairwise disjoint, so the sum may count some edges more than once.

The CrossEdgeCount problem is an analog of the above for induced bipartite subgraphs. The input is a stream of edges followed by ℓ *pairs* of vertex subsets $\langle (U_1, W_1), \dots, (U_\ell, W_\ell)\rangle$, where $U_i \cap W_i = \varnothing$ for $i \in [\ell]$. The desired output is $\sum_{i=1}^{\ell} |E(G[U_i, W_i])|$, the sum of the number of cross-edges in the induced bipartite subgraphs $G[U_1, W_1], \dots, G[U_\ell, W_\ell]$. Note that the U_is (or W_is) need not be disjoint among themselves.

Importantly, in both of these problems, the edges *precede* the vertex subsets in the stream. This makes the problems intractable in the basic data streaming model. We shall prove the following results.

▶ **Lemma 6.** *For any h, v with $hv = n^2$, there is an $[h, v]$-protocol for* InducedEdgeCount.

▶ **Lemma 7.** *For any h, v with $hv = n^2$, there is an $[h, v]$-protocol for* CrossEdgeCount.

Scheme for InducedEdgeCount (Proof of Lemma 6). For the given instance, let M denote the desired output and let A be the adjacency matrix of G. For each $i \in \ell$, let $B_i \in \{0, 1\}^V$ be the indicator vector of the set U_i, i.e., $B_i(v) = 1 \iff v \in U_i$. Then,

$$M = \frac{1}{2}\sum_{i=1}^{\ell} \sum_{v_1, v_2 \in V} B_i(v_1)\, B_i(v_2)\, A(v_1, v_2)\,. \tag{8}$$

Let t, s be integer parameters such that $ts = n$. We apply the shaping technique to Equation (8) by rewriting the variables v_j as pairs of integers $(x_j, y_j) \in [t] \times [s]$, for $j \in \{1, 2\}$. This transforms the matrix A into a 4-dimensional array a and each B_i into a 2-dimensional array b_i. Let $\tilde{a}$ and $\tilde{b}_i$ be the respective $\mathbb{F}$-extensions. Equation (8) now gives

$$2M = \sum_{i=1}^{\ell} \sum_{x_1, x_2 \in [t]} \sum_{y_1, y_2 \in [s]} \tilde{b}_i(x_1, y_1)\, \tilde{b}_i(x_2, y_2)\, \tilde{a}(x_1, y_1, x_2, y_2) = \sum_{x_1, x_2 \in [t]} p(x_1, x_2)\,, \tag{9}$$

$$\text{where} \quad p(X_1, X_2) = \sum_{i=1}^{\ell} \sum_{y_1, y_2 \in [s]} \tilde{b}_i(X_1, y_1)\, \tilde{b}_i(X_2, y_2)\, \tilde{a}(X_1, y_1, X_2, y_2)\,. \tag{10}$$

Our scheme exploits this expression in the same general manner as the analogous expressions for the TriangleCount schemes from Section 2 (e.g., Equation (4)). Prover sends a bivariate polynomial $\hat{p}(X_1, X_2)$, which is claimed to be p, by streaming its coefficients. Since $\deg_{X_j} p \leqslant 2(t-1)$ for $j \in \{1, 2\}$, Prover need only send $O(t^2)$ coefficients, for a help cost of $\tilde{O}(t^2)$. Verifier computes his output using Equation (9), giving perfect completeness. On the soundness side, Verifier checks the condition $\hat{p}(r_1, r_2) = p(r_1, r_2)$ for randomly chosen $r_1, r_2 \in_R \mathbb{F}$. By the Schwartz-Zippel Lemma (Fact 2), the probability that he is fooled is at most $\deg p/|\mathbb{F}| = O(t)/|\mathbb{F}| < 1/n$, for the right choice of $\mathbb{F}$. It remains to describe how exactly Verifier evaluates $p(r_1, r_2)$, which we now address.

Processing the stream of edges. This is straightforward: Verifier maintains the 2-dimensional array of values $\tilde{a}(r_1, w, r_2, z)$, for all $w, z \in [s]$, using Fact 1.

Processing the stream of vertex subsets. Verifier initializes an accumulator to zero and allocates workspace for two arrays of length s with entries in $\mathbb{F}$. For each $i \in [\ell]$, as the vertices of U_i arrive, he maintains $\tilde{b}_i(r_1, z)$ and $\tilde{b}_i(r_2, z)$ for each $z \in [s]$, using that workspace. Upon seeing the delimiter marking the end of U_i, he computes

$$\sum_{y_1, y_2 \in [s]} \tilde{b}_i(r_1, y_1)\, \tilde{b}_i(r_2, y_2)\, \tilde{a}(r_1, y_1, r_2, y_2) \tag{11}$$

and adds this quantity to the accumulator. Note that the workspace is reused when the stream moves on from U_i to U_{i+1}. By Equation (10), after the last set U_ℓ is streamed, the accumulator holds $p(r_1, r_2)$.

Help and verification costs. We argued above that the hcost is $\tilde{O}(t^2)$. Meanwhile, Verifier's storage is dominated by the $s \times s$ array he maintains, leading to a vcost of $\tilde{O}(s^2)$.

Therefore, we obtain a $[t^2, s^2]$-scheme for any parameters t, s with $ts = n$. In other words, we get an $[h, v]$-scheme for any h, v with $hv = n^2$.

Scheme for CrossEdgeCount (Proof of Lemma 7). Our solution for InducedEdgeCount can easily be modified to obtain a protocol for CrossEdgeCount with the same costs. If B_i and C_i are the indicator vectors of the sets U_i and W_i, respectively, then the desired output is

$$M = \sum_{i=1}^{\ell} \sum_{v_1, v_2 \in V} B_i(v_1)\, C_i(v_2)\, A(v_1, v_2)\,, \tag{12}$$

where we used the fact that each $U_i \cap W_i = \varnothing$. Since Equation (12) has essentially the same form as Equation (8), a scheme very similar to the previous one solves CrossEdgeCount: Verifier simply keeps track of arrays corresponding to C_i alongside ones corresponding to B_i.

4 Maximum Matching and Other Applications of Edge Counting

In this section, we show how InducedEdgeCount and CrossEdgeCount can be used as subroutines to solve multiple problems that have been widely studied in the basic and annotated data streaming models. These problems include Maximum Matching, Triangle-Counting, Maximal Independent Set, Acyclicity Testing, Topological Sorting, and Graph Connectivity. For the frugal regime where vcost $= o(n)$, our schemes are often optimal. We specifically discuss the application to MaxMatching in Section 4.1, state the results on the other applications in Section 4.2, and give a detailed account on them in Appendix B.

4.1 The Maximum Matching Problem

We give the first optimal frugal scheme for computing the cardinality $\alpha'(G)$ of a maximum matching. As noted in prior works [14, 38], checking whether $\alpha'(G) \geqslant k$ for some k is not hard, given $\tilde{\Omega}(k)$ bits of help: Prover can simply send a matching of size k and prove its validity. The interesting part is to verify that $\alpha'(G) \leqslant k$. For this, as in prior works, we exploit the Tutte–Berge formula [9]:

$$\alpha'(G) = \frac{1}{2} \min_{U \subseteq V} \Big(|U| + |V| - \operatorname{odd}(G \setminus U) \Big) , \tag{13}$$

where $\operatorname{odd}(G \setminus U)$ denotes the number of connected components in $G \setminus U$ with an odd number of vertices. Thus, to show that $\alpha'(G) \leqslant k$, Prover needs to exhibit $U^* \subseteq V$ such that $k = \frac{1}{2}(|U^*| + |V| - \operatorname{odd}(G \setminus U^*))$. Set $H := G \setminus U^*$. To verify the value of $\operatorname{odd}(H)$, the most important sub-check that Verifier must do is to check that all purported connected components of H (sent by Prover) are actually disconnected from each other. Thaler [38] gave an $[n, n]$-scheme for this subproblem (thus obtaining the first $[n, n]$-scheme for MaxMatching), while Chakrabarti and Ghosh [14] gave a $[t^3, s^2]$-scheme for any $ts = n$ (thus designing the first frugal scheme for MaxMatching, though suboptimal). The latter work notes that all other sub-checks for MaxMatching can be done by *optimal* frugal schemes (see [14], Section 4).

Optimal Frugal Scheme. To optimally check that the purported connected components of H are indeed disconnected from each other, we use the InducedEdgeCount scheme as a subroutine. Prover streams the vertices in H by listing its connected components in some order $\langle U_1, \ldots, U_\ell \rangle$. Verifier uses Lemma 6 to count $m_1 := |E(H)|$ (invoking that lemma with a single subset $V(H)$). In parallel, using the same scheme, Verifier computes the sum $m_2 = \sum_{i=1}^{\ell} |E(G[U_i])|$. The subsets U_i are pairwise disconnected iff $m_2 = m_1$, which Verifier checks. The sub-checks of whether U_is are indeed pairwise disjoint (as sets) and whether $U^* \sqcup V(H) = V(G)$ can be done via fingerprinting (as in Section 1.5).

Help and verification costs. Prover streams U^* and the vertices in H in a certain order, which adds $O(n \log n)$ bits to the hcost of the InducedEdgeCount protocol. The vcost stays the same, asymptotically, giving us an $[n + h, v]$-scheme for MaxMatching for any h, v with $hv = n^2$. Overall, we have established the following theorem.

▶ **Theorem 8.** *There is an $[nt, s]$-scheme for MaxMatching. This is optimal up to logarithmic factors, since any (h, v)-scheme is known to require $hv = \Omega(n^2)$ [12].*

Protocol for Space Larger Than n. There is no laconic scheme known for the general MaxMatching problem. The barrier seems to be that a natural witness for the problem is an actual maximum matching of the graph, which can be of size $\Theta(n)$. We show that large maximum matching size $\alpha'(G)$ is indeed the sole barrier to obtaining a laconic scheme. In particular, for any graph G, we give a scheme for MaxMatching with hcost $\alpha'(G)$. This yields a laconic scheme for the case when $\alpha'(G) = o(n)$. We defer the proof to Appendix A.

▶ **Theorem 9.** *For any h, v with $hv \geqslant n^2$ and $v \geqslant n$, there is an $[\alpha' + h, v]$-scheme for MaxMatching, where α' is the size of the maximum matching of the input graph. In particular, there is an $[\alpha', n^2/\alpha']$-scheme.*

4.2 Applications to Other Graph Problems

Here, we state the results we obtain for several graph problems by applying InducedEdgeCount and CrossEdgeCount. The details and proofs appear in Appendix B.

We apply the edge-counting protocols to solve the triangle-counting problem in both the standard edge arrival and the vertex arrival (adjacently list) streaming models and obtain the following theorems.

▶ **Theorem 10.** *For any h, v with $hv \geqslant n^2$, there is an $[m + h, v]$-scheme for TriangleCount. In particular, there is an $[m, n^2/m]$-scheme.*

▶ **Theorem 11.** *For any h, v with $hv \geqslant n^2$, there is an $[h, v]$-scheme for TriangleCountAdj.*

We obtain optimal frugal schemes for the fundamental graph problems of maximal independent set (MIS) and for topological sorting and acyclicity testing in directed graph streams.

▶ **Theorem 12.** *For any t, s with $ts = n$, there is an $[nt, s]$-scheme for MIS. This is optimal up to logarithmic factors, since any (h, v)-scheme is known to require $hv = \Omega(n^2)$.*

▶ **Theorem 13.** *For any t, s with $ts = n$, there is an $[nt, s]$-scheme for TopoSort. This is optimal up to logarithmic factors, since any (h, v)-scheme is known to require $hv = \Omega(n^2)$.*

▶ **Corollary 14.** *For any t, s with $ts = n$, there is an $[nt, s]$-scheme for Acyclicity. This is optimal up to logarithmic factors, since any (h, v)-scheme is known to require $hv = \Omega(n^2)$.*

Finally, we show that we can apply edge-counting schemes to count the number of connected components of a graph.

▶ **Theorem 15.** *For any t, s with $ts = n$, there is an $[nt, s]$-scheme for counting the number of connected components of an input graph.*

5 The Single-Source Shortest Path Problem

In the single-source shortest path (SSSP) problem, the goal is to find the distances from a source vertex v_s to every other vertex reachable from it. In Section 5.1, we give a $[Dnt, s]$-scheme for the unweighted version, whenever $ts = n$. If $s = o(n)$, Verifier does not have enough space to store the output; therefore, we aim for a protocol where Prover streams the output, and Verifier checks that it is correct using $o(n)$ space, thus achieving a frugal scheme.

In Section 5.2, we state our results for weighted SSSP for the two different weight update models (vanilla and turnstile) described in Section 1.5. The proofs follow in Appendix C.

5.1 Unweighted SSSP

We shall design a scheme that works even if the same edge appears multiple times in the stream (unlike prior work [18] that assumes that an edge appears at most once).

Prover sends distance labels $\widehat{\text{dist}}[v]$ for all $v \in V$, claiming that $\widehat{\text{dist}}[v] = \text{dist}(v_s, v)$, the actual distance from the source vertex v_s to v. Let the radius-d ball around v_s be $B_d := \{v \in V : \text{dist}(v_s, v) \leqslant d\}$ and let $\mathcal{B} := \{B_d : d \in [D]\}$ be the family of such balls. Let $\widehat{B}_d$ be the corresponding balls implied by Prover's $\widehat{\text{dist}}$ labels, and $\widehat{\mathcal{B}} := \{\widehat{B}_d : d \in [D]\}$.

To check correctness, Verifier uses fingerprinting (Section 1.5) modified as follows. Letting $B, \widehat{B}$ also denote the respective characteristic vectors, define fingerprint polynomials

$$\varphi_{\mathcal{B}}(X,Y) := \sum_{i\in[n]} \sum_{d\in[D]} B_d(i) X^i Y^d \,, \quad \varphi_{\widehat{\mathcal{B}}}(X,Y) := \sum_{i\in[n]} \sum_{d\in[D]} \widehat{B}_d(i) X^i Y^d \,,$$

As the $\widehat{\text{dist}}$ labels are streamed, Verifier constructs the fingerprint $\varphi_{\widehat{\mathcal{B}}}(\beta_1, \beta_2)$ for some $\beta_1, \beta_2 \in_R \mathbb{F}$.

Over the course of the protocol, using further help from Prover, Verifier will construct the sets B_d inductively and, in turn, the "actual" fingerprint $\varphi_{\mathcal{B}}(\beta_1, \beta_2)$. The next claim shows that comparing this with $\varphi_{\widehat{\mathcal{B}}}(\beta_1, \beta_2)$ validates Prover's $\widehat{\text{dist}}$ labels.

▷ Claim 16. If $\widehat{B}_d = B_d$ for all d, then $\widehat{\text{dist}}[v] = \text{dist}(v_s, v)$ for all vertices v.

Proof. Suppose not. Let d^* be the smallest d such that $\exists\, u \in B_{d^*}$ with $\widehat{\text{dist}}[u] \neq \text{dist}(v_s, u)$. Therefore, $\text{dist}(v_s, u) = d^*$. Now, d^* cannot be 0 since v_s is the only vertex in B_0 and Verifier would reject immediately if $\widehat{\text{dist}}(v_s) \neq 0$. Since $B_{d^*} = \widehat{B}_{d^*}$, we have $u \in \widehat{B}_{d^*}$. This means $\widehat{\text{dist}}(u) \leqslant d^*$. Since $\widehat{\text{dist}}(u) \neq d^*$, we have $\widehat{\text{dist}}(u) \leqslant d^* - 1$. Thus, $u \in \widehat{B}_{d^*-1}$, i.e., $u \in B_{d^*-1}$, which is a contradiction to the minimality of d^*. ◁

As before, A denotes the adjacency matrix of the graph. Putting

$$q_d(u) := \sum_{v\in V} B_d(v)\, A(v,u) \,, \text{ for each } u \in V \,, \tag{14}$$

$$\text{we have} \quad B_{d+1} = \{u \in V : q_d(u) \neq 0\} \,. \tag{15}$$

To apply the shaping technique to (14), rewrite v as $(x,y) \in [t] \times [s]$. This reshapes A into a $t \times s \times n$ array $a(x,y,u)$ and B_d into a $t \times s$ array $b_d(x,y)$. As usual, let $\tilde{a}$ and $\tilde{b}_d$ be the respective $\mathbb{F}$-extensions for a suitable finite field $\mathbb{F}$. Then, Equation (14) gives

$$q_d(u) = \sum_{x\in[t]} p_d(x,u) \,, \quad \text{where} \tag{16}$$

$$p_d(X,U) := \sum_{y\in[s]} \tilde{b}_d(X,y)\, \tilde{a}(X,y,U) \,. \tag{17}$$

Stream processing. Verifier picks $r_1, r_2 \in_R \mathbb{F}$ and maintains $\tilde{a}(r_1, y, r_2)$. When he sees vertices in B_1, i.e., v_s and its neighbors, he maintains $b_1(r_1, y)$ for all $y \in [s]$ and also updates the fingerprint $\varphi_{\mathcal{B}}(\beta_1, \beta_2)$ accordingly.

Verifier wants to construct the values $b_d(r_1, y)$ inductively for $d \in [D]$. For constructing b_{d+1} values for some d, he wants all u such that $q_d(u) \neq 0$ (Equation (15)) in streaming order since he doesn't have enough space to either store the entire polynomial of degree $n-1$ that agrees with q_d (so as to go over all evaluations), or to parallelly evaluate it at n values while its coefficients are streamed. Hence, he asks for the following help message.

Help message processing. Prover continues her proof stream by sending $\langle \hat{p}_1, Q_1, \ldots, \hat{p}_D, Q_D \rangle$, where $Q_d := \langle \hat{q}_d(u) : u \in V \rangle$, claiming that $\hat{p}_d \equiv p_d$ and $\hat{q}_d(u) = q_d(u)$ for each $d \in [D]$ and $u \in [n]$.

While $\hat{p}_d$ is streamed, Verifier computes the following in parallel:

- $\hat{p}_d(r_1, r_2)$;
- $p_d(r_1, r_2)$, using Equation (17);
- the fingerprint $g_d := \sum_{u\in[n]} \sum_{x\in[t]} \hat{p}_d(x,u)\beta^u$ (for some $\beta \in_R \mathbb{F}$).

After reading $\hat{p}_d$, he checks whether $\hat{p}_d(r_1, r_2) = p_d(r_1, r_2)$. If so, he believes that $\hat{p}_d \equiv p_d$ and, in turn, that $g_d = \sum_{u \in [n]} q_d(u)\beta^u$ (by Equation (16)). Next, as Q_d is streamed,

- Verifier computes the fingerprint $g'_d := \sum_{u \in [n]} \hat{q}_d(u)\beta^u$.
- For each u with $\hat{q}_d(u) \neq 0$, due to Equation (15) (and assuming for now that the $\hat{q}_d$ values are correct), he treats u as a stream update for B_{d+1}, and (i) maintains $b_{d+1}(r_1, y)$ for all $y \in [s]$, and (ii) accordingly updates the fingerprint $\varphi_{\mathcal{B}}(\beta_1, \beta_2)$.

After reading Q_d, he checks if the fingerprints g_d and g'_d match. If they do, he believes that all $\hat{q}_d$ values in Q_d were correct and hence, the b_{d+1} values he constructed are correct as well. He moves on to the next iteration, i.e., starts reading $\hat{p}_{d+1}$.

Final Verification. After the Dth iteration, Verifier checks if the two fingerprints $\varphi_{\mathcal{B}}(\beta_1, \beta_2)$ and $\varphi_{\widehat{\mathcal{B}}}(\beta_1, \beta_2)$ match. If the check passes, then he believes that the $\widehat{\text{dist}}$ labels were correct, at least upto distance D (by Claim 16). Finally, he checks if fingerprints for B_D and B_{D+1} match to verify that vertices in $V \setminus B_D$ are indeed unreachable.

Error probability. Verifier does $O(D)$ fingerprint-checks and $O(D)$ sum-checks, using degree-$O(n)$ polynomials. Using $|\mathbb{F}| > n^3$ (and a union bound), the soundness error is $< 1/n$.

Help and verification costs. The set of $\widehat{\text{dist}}$ labels sent by the Prover has size $\tilde{O}(n)$. Each polynomial $\hat{p}_d$ has nt monomials and each Q_d has $O(n)$ field elements, and hence, size $\tilde{O}(n)$. Therefore, the total hcost is $\tilde{O}(Dnt)$. Initially, the $\tilde{A}$ and $\tilde{b}_1$ values are stored using $\tilde{O}(s)$ space. Next, the $\tilde{b}_d$ and g_d values are maintained reusing space of b_{d-1} and g_{d-1} values respectively. We also use $O(1)$ many other fingerprints that take $O(\log n)$ space each. Hence, the total vcost is $\tilde{O}(s)$.

▶ **Theorem 17.** *There is a $[Dnt, s]$-scheme for unweighted SSSP, where $D = \max_{v \in V} \text{dist}(v_s, v)$.*

▶ **Corollary 18.** *There is a $[Knt, s]$-scheme for* st-ShortestPath*, where $K = \text{dist}(v_s, v_t)$.*

Proof. The protocol for SSSP incurs a factor of D in the hcost since it constructs B_d for each $d \in [D]$. For the simpler st-ShortestPath problem, we can inductively construct balls and stop as soon as we find the destination vertex v_t in some B_d (i.e., get $\hat{q}_{d-1}(v_t) \neq 0$). We must find it in B_K where K is the length of a shortest v_s–v_t path. Thus, we will only incur a factor of K in the hcost, which implies a $[Knt, s]$-scheme for st-ShortestPath. ◀

Thus, we generalize the $[Dnt, s]$-scheme of Cormode et al. [18] from st-ShortestPath to SSSP. Our result for st-ShortestPath generalizes the $[Kn, n]$-scheme of Chakrabarti and Ghosh [14] by giving a smooth tradeoff and also improves upon the $[Dnt, s]$-scheme of Cormode et al. [18], since K can be arbitrarily smaller than D.

5.2 Weighted SSSP

We consider the more general weighted version of SSSP in the turnstile and the vanilla weight update models (see Section 1.5 for the definitions) and obtain the following results. The details of the schemes along with the proofs of correctness appear in Appendix C.

▶ **Theorem 19.** *There is a $[DWn, n]$-scheme for weighted SSSP in the turnstile weight update model.*

▶ **Theorem 20.** *There is a $[Dn, Wn]$-scheme for weighted SSSP in the vanilla streaming model.*

References

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

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 459–467, 2012.

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

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

5 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear algorithms for $(\Delta+1)$ vertex coloring. In *Proc. 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 767–786, 2019.

6 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1723–1742, 2017.

7 Ziv Bar-Yossef, Ravi 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.

8 Suman K. Bera and Amit 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.

9 J.A. Bondy and U.S.R Murty. *Graph Theory*. Springer Publishing Company, Incorporated, 1st edition, 2008.

10 Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proc. 25th ACM Symposium on Principles of Database Systems*, pages 253–262, 2006.

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

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

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

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

15 Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 1786–1802, 2020.

16 Amit Chakrabarti and Sagar 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.

17 Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. In *Proc. 46th International Colloquium on Automata, Languages and Programming*, pages 45:1–45:14, 2019.

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

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

20 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate maximum matching in random streams. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1773–1785, 2020.

21 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian 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.

22 Ashish Goel, Michael Kapralov, and Sanjeev 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.

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

24 Madhav Jha, C. Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proc. 19th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 589–597, 2013.

25 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics*, pages 710–716. Springer Berlin Heidelberg, 2005.

26 John Kallaugher, Andrew McGregor, Eric Price, and Sofya Vorotnikova. The complexity of counting cycles in the adjacency list streaming model. In *Proc. 38th ACM Symposium on Principles of Database Systems*, pages 119–133, 2019.

27 Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. Counting arbitrary subgraphs in data streams. In *Automata, Languages, and Programming*, pages 598–609. Springer Berlin Heidelberg, 2012.

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

29 Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. Space efficient approximation to maximum matching size from uniform edge samples. In *Proc. 31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1753–1772, 2020.

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

31 Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012.

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

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

34 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better algorithms for counting triangles in data streams. In *Proc. 35th ACM Symposium on Principles of Database Systems*, pages 401–411, 2016.

35 Michael Mitzenmacher and Justin Thaler. Technical perspective: Catching lies (and mistakes) in offloaded computation. *Commun. ACM*, 59(2):102, 2016.

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

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

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

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

40 Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2001.

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

A Maximum Matching Redux (Proof of Theorem 9)

In Section 4.1, we gave a frugal scheme for MaxMatching, i.e., one that has hcost $\geqslant n$ and vcost $\leqslant n$. Here, we show that we can get a laconic scheme, i.e., one with hcost $= o(n)$ and vcost $= \omega(n)$ as long as the maximum matching size is $o(n)$.

Let $H = G \setminus U^*$, where U^* is as in the MaxMatching protocol in Section 4.1, and let $U_1, \ldots, U_\ell$ be the connected components of H. By the Tutte-Berge formula (Equation (13)), we have $2k = |U^*| + (n - \text{odd}(H))$. This leads to the following observations.

▶ **Observation 21.** $|U^*| = O(k)$.

▶ **Observation 22.** *The number of edges in a spanning forest of H is $|V(H)| - \ell \leqslant n - \mathit{odd}(H) = O(k)$.*

We now describe our protocol, which is along the lines of the protocol in Section 4.1, but this time we crucially use the fact that we are allowing Verifier a space usage of $v \geqslant n$.

To show that $\alpha'(G) \geqslant k$, Prover sends a matching M of size k. Verifier stores M explicitly and checks that it is indeed a matching. Then, he verifies that $M \subseteq E$ using the Subset Scheme (Fact 3). Therefore, this part of the scheme uses hcost $\tilde{O}(k + h)$ and vcost $\tilde{O}(v)$ for any h, v with $hv = n^2$ and $v \geqslant n$.

Recall from Section 4.1 that to show that $\alpha'(G) \leqslant k$, it suffices to compute odd(H). Prover sends the set U^*. By Observation 21, this takes $\tilde{O}(k)$ hcost. Verifier has $\Omega(n)$ space, and hence, he can store $V \setminus U^* = V(H)$. Next, Prover sends a spanning forest F of H. By Observation 22, this again incurs hcost $\tilde{O}(k)$. Verifier stores F and verifies that $F \subseteq E$ using the Subset Scheme (Fact 3). From F, Verifier explicitly knows the purported connected components $U_1, \ldots, U_\ell$ of H. He finally verifies that U_i's are disconnected from each other by checking that all edges in H are contained in these components. He can do this by checking whether $|E \cap (V(H) \times V(H))| = |E \cap (\cup_{i=1}^{\ell} U_i \times U_i)|$ using the Intersection Scheme (Fact 3). If the check passes he goes over the U_is to compute odd(H) and thus, this part can also be solved using a $[k + h, v]$ scheme for any h, v with $hv = n^2$ and $v \geqslant n$. Hence, we obtain the following theorem.

▶ **Theorem 9.** *For any h, v with $hv \geqslant n^2$ and $v \geqslant n$, there is an $[\alpha' + h, v]$-scheme for MaxMatching, where α' is the size of the maximum matching of the input graph. In particular, there is an $[\alpha', n^2/\alpha']$-scheme.*

B Applications of Edge-Counting to Other Graph Problems

In Section 4.1, we used a scheme for InducedEdgeCount to obtain an optimal frugal scheme for MaxMatching. Below, we give applications of edge-counting schemes to several other well-studied graph problems.

Triangle-Counting. A scheme for TRIANGLECOUNT follows immediately from INDUCEDEDGECOUNT. For $v \in [n]$, set the subsets $U_v = N(v)$, the neighborhood of vertex v. Then, observe that INDUCEDEDGECOUNT returns three times the total number of triangles in the graph. The sets U_v, however, need to be sent in some order by Prover, and so the additional hcost to INDUCEDEDGECOUNT is $\tilde{O}\left(\sum_v |N(v)|\right) = \tilde{O}(m)$. As Prover basically repeats the edge stream in a different order, we can check if it's consistent with the input stream by fingerprinting (see Section 1.5). Hence, we get an $[m+h, v]$-scheme for any h, v with $hv = n^2$.

▶ **Theorem 10.** *For any h, v with $hv \geqslant n^2$, there is an $[m+h, v]$-scheme for TRIANGLECOUNT. In particular, there is an $[m, n^2/m]$-scheme.*

The only other scheme for TRIANGLECOUNT achieving $hv = n^2$ tradeoff with vcost $= o(n)$ was an $[n^2, 1]$-scheme by Chakrabarti et al. [12]. Our result generalizes it for any graph with m edges, thus achieving a better hcost and a smooth tradeoff for sparse graphs.

We note that in the above scheme, Prover needs to send the sets $U_v = N(v)$ because the INDUCEDEDGECOUNT protocol needs the neighborhood of each vertex to arrive contiguously in the stream. This is essentially the input stream order in the *adjacency-list* or the *vertex-arrival* streaming model. Thus, for the problem TRIANGLECOUNT-ADJ, Verifier gets the U_vs in the desired order as part of the input; so Prover need not repeat them, saving the huge $\tilde{O}(m)$ hcost. However, there is another issue in directly applying the INDUCEDEDGECOUNT subroutine in this case. In the definition of INDUCEDEDGECOUNT, we assume that all the edges in the graph arrive before the vertex subsets U_i. Here, the U_vs and the edges arrive in interleaved manner (although each U_v arrives contiguously). But we show that we can still apply the scheme for INDUCEDEDGECOUNT to get the desired output. Let the order in which the U_vs appear be $\langle U_1, \ldots U_n\rangle$, and let G_v denote the graph consisting of edges seen till the arrival of $U_v = N(v)$. Then, applying INDUCEDEDGECOUNT, what we count is

$$\sum_{v \in [n]} |E(G_v[N(v)])| = \sum_{v \in [n]} \#\{\text{triangles incident on } v \text{ in } G_v\} = 2T\,.$$

The last equality follows since every triangle whose vertices appear in the order $\langle v_1, v_2, v_3\rangle$ will be counted twice: once when v_2 arrives and once when v_3 arrives. We therefore obtain the following theorem.

▶ **Theorem 11.** *For any h, v with $hv \geqslant n^2$, there is an $[h, v]$-scheme for TRIANGLECOUNT-ADJ.*

Maximal Independent Set (MIS). Recent works [5, 17] have studied the problem of finding a maximal independent set in the basic data streaming model. They show a lower bound of $\Omega(n^2)$ for a one-pass streaming algorithm. This implies a lower bound of $hv \geqslant n^2$ for any $[h, v]$-scheme for MIS. Hence, we aim for $hv = n^2$ and describe a frugal scheme using INDUCEDEDGECOUNT. Since the output size of the problem can be $\Theta(n)$, it would only make sense in the frugal regime if the Prover sends the output as a stream and the Verifier checks that it is valid using $o(n)$ space.

Let U be an MIS in the graph G. Prover sends U and Verifier uses INDUCEDEDGECOUNT to count the number of edges in $G[U]$ and verifies that it equals 0. If the check passes, U is indeed an independent set. It remains to check the maximality of U. If U is maximal, then, for each vertex v in $G \setminus U$, there must be a vertex u in U, such that (v, u) is an edge. Prover points out such a vertex $u \in U$ for each $v \in G \setminus U$. Let F denote this set of $|G \setminus U|$

purported edges. Now, we use Subset Scheme (Fact 3) to verify that $F \subseteq E$, i.e., all these edges are actually present in G. We can use fingerprinting (as in Section 1.5) to check that F contains an edge for each vertex in $G \setminus U$ and the Intersection Scheme to verify that the set of their partners is disjoint from $G \setminus U$, i.e., belong to U. Thus, the additional hcost to InducedEdgeCount, Subset, and Intersection Schemes is $\tilde{O}(n)$, the number of bits required to send U and F. Therefore, by Lemma 6, we get an $[n+h, v]$-scheme for MIS for any h, v with $hv = n^2$. Thus, our scheme is optimal for the frugal regime.

▶ **Theorem 12.** *For any t, s with $ts = n$, there is an $[nt, s]$-scheme for MIS. This is optimal up to logarithmic factors, since any (h, v)-scheme is known to require $hv = \Omega(n^2)$.*

Acyclicity Testing and Topological Sorting. We now turn to the Acyclicity problem in directed graphs. It is easy to prove that a graph is *not* acyclic by showing the existence of a cycle C. Verifier checks that $C \subseteq E$ using Subset Scheme (Fact 3). Hence, this can be done using an $[h, v]$-scheme for any $h \geqslant |C|$.

The more interesting case is when the graph is indeed acyclic. Note that a directed graph is acyclic if and only if it has a topological ordering. Thus, it suffices to show a valid topological ordering of the vertices. TopoSort is a fundamental graph algorithmic problem of independent interest. Acyclicity has a one-pass lower bound of $\Omega(n^2)$ in the basic data streaming model. Recently, Chakrabarti et al. [15] showed that TopoSort also requires $\Omega(n^2)$ space in one pass. These translate to a lower bound of $hv \geqslant n^2$ for any $[h, v]$-scheme for these problems. Hence, we aim for a scheme with $hv = n^2$ and design a protocol for TopoSort in the frugal regime. Since this problem has output size $\tilde{\Theta}(n)$, we aim for a protocol where Prover sends a topological ordering of the graph and Verifier checks its validity using $o(n)$ space. Moreover, this protocol can be used for the YES case of Acyclicity.

Verifier uses CrossEdgeCount to solve this. As Prover sends the topological order $\langle v_1, \ldots, v_n \rangle$, for each $i \in [n-1]$, Verifier sets $U_i = \{v_1, \ldots, v_i\}$ and $W_i = \{v_{i+1}\}$ for CrossEdgeCount. Thus, the protocol counts precisely the number of forward edges induced by the ordering. If it equals m, then the ordering is indeed a valid topological order. Note that since $U_{i+1} = U_i \cup \{v_{i+1}\}$, Prover doesn't need to send U_{i+1} afresh; just v_{i+1} is enough for Verifier to update his sketch. Verifier can use fingerprinting (see Section 1.5) to make sure that precisely the set V was sent in some order. Hence, the additional hcost to CrossEdgeCount is the number of bits required to express the topological order, i.e., $\tilde{O}(n)$. Therefore, by Lemma 7, we get a $[n+h, v]$-scheme for any $hv = n^2$.

▶ **Theorem 13.** *For any t, s with $ts = n$, there is an $[nt, s]$-scheme for TopoSort. This is optimal up to logarithmic factors, since any (h, v)-scheme is known to require $hv = \Omega(n^2)$.*

▶ **Corollary 14.** *For any t, s with $ts = n$, there is an $[nt, s]$-scheme for Acyclicity. This is optimal up to logarithmic factors, since any (h, v)-scheme is known to require $hv = \Omega(n^2)$.*

For dense graphs, our result generalizes the $[m, 1]$-scheme of Cormode et al. [18] for Acyclicity by achieving a smooth tradeoff.

Graph Connectivity. The graph connectivity problem has garnered considerable attention in the basic and annotated streaming settings [2, 12, 38]. For any t, s with $ts = n$, Chakrabarti et al. [12] gave an $[nt, s]$-scheme that determines whether an input graph is connected or not. Their scheme cannot, however, solve the more general problem of returning the number of connected components. The $[t^3, s^2]$-scheme (for any $ts = n$) of Chakrabarti and Ghosh [14] does solve this problem, but has a worse tradeoff. As noted in Section 4.1, we can use InducedEdgeCount to check that all purported connected components are indeed disconnected from each other. On the other hand, the scheme of Chakrabarti et al. [12] can check whether each component is actually connected. Hence, we can verify the number of connected components claimed by Prover by running these schemes parallelly. Thus, we generalize the result of Chakrabarti et al. [12] by obtaining an $[nt, s]$-scheme for counting the number of connected components of a graph.

▶ **Theorem 15.** *For any t, s with $ts = n$, there is an $[nt, s]$-scheme for counting the number of connected components of an input graph.*

C Details of Weighted SSSP Schemes

Here, we give full details of the schemes for weighted SSSP in our two streaming models, thereby proving Theorems 19 and 20.

Turnstile weight update (Proof of Theorem 19). Assume that the edge weights are positive integers. Each stream update increments/decrements the weight of an edge. The *distance* from vertex u to vertex v refers to the weight of the shortest path from u to v. Let D be the longest distance from the source s to any other vertex reachable from it, and W be the maximum weight of an edge.

Define

$$\delta_w(X) := \prod_{\substack{w' \in [W] \\ w' \neq w}} (X - w') \Bigg/ \prod_{\substack{w' \in [W] \\ w' \neq w}} (w - w') \,.$$

Let A denote the adjacency matrix of the weighted graph G, i.e., $A(u, v)$ is the weight of the edge (u, v). Let B_d (resp. N_d) denote the set of vertices at a distance of at most (resp. exactly) d from the source vertex v_s. Then,

$$N_{d+1} = \{u \in V \setminus B_d : p_d(u) \neq 0\} \,, \tag{18}$$

$$\text{where} \quad p_d(U) = \sum_{v \in B_d} \delta_{w(v)}(\tilde{A}(v, U)) \quad \text{and} \quad w(v) = d + 1 - \text{dist}[v] \,. \tag{19}$$

Stream processing. Verifier chooses $r \in_R \mathbb{F}$ and maintains $\tilde{A}(v, r)$ for all v. He stores B_1 with dist$[v]$ labelled as 1 for each $v \in B_1$.

Help message processing and verification. Prover sends polynomials $\hat{p}_d$ and claims that $\hat{p}_d \equiv p_d$ for each $d \in [D]$. Verifier computes B_d inductively for $d \in [D]$ as follows.

Assume that, for some $d \in [D - 1]$, he has the set B_d with dist$[v]$ labeled on each vertex $v \in B_d$; this holds initially as he has stored B_1. He computes $p_d(r)$ using Equation (19) and checks whether $\hat{p}_d(r) = p_d(r)$. If the check passes, he believes that $\hat{p}_d \equiv p_d$ and evaluates $\hat{p}_d(u)$ for each $u \in V \setminus B_d$ and constructs N_{d+1} using Equation (18). Then, B_{d+1} is given by $N_{d+1} \uplus B_d$.

After B_D is obtained, we get all vertices reachable from s along with their distances from s. Finally, Verifier checks if the other vertices are indeed unreachable from s by verifying that there is no cross-edge between B_D and $V \setminus B_D$, i.e., if $E \cap (B_D \times (V \setminus B_D)) = \varnothing$. (Intersection scheme, see Fact 3)

Error probability. Verifier uses the same element r for $O(D)$ invocations of the sum-check protocol, where each application of the sum-check protocol is to a univariate polynomial of degree $O(Wn)$. Choosing $|\mathbb{F}| > DWn^2$, the soundness error for each invocation of the sum-check protocol is at most $1/(Dn)$. Taking a union bound over all $O(D)$ invocations, we get that the total error probability of the protocol is at most $O(1/n)$.

Help and verification costs We have $\deg p_d = O(Wn)$ for each $d \in [D]$ and hence, hcost is $\tilde{O}(DWn)$. Verifier needs to store all vertices and $\tilde{A}(v, r)$ for each $v \in [n]$, and hence, vcost is $\tilde{O}(n)$. The final disjointness can be checked by an $[n, n]$ intersection scheme.

▶ **Theorem 19.** *There is a $[DWn, n]$-scheme for weighted SSSP in the turnstile weight update model.*

Vanilla Stream (Proof of Theorem 20). We now describe a protocol for SSSP in the model where the edges arrive with their weights, without any further update on them. This is the "vanilla" streaming model.

At the end of the stream, Prover sends the distances $\text{dist}[v]$ and $\text{prev}[v]$ – the parent of v in the shortest path tree rooted at s – for all $v \in V$. Verifier checks whether the edges and their weights implied by this proof are correct, using a $[Wn, n]$ subset scheme. Thus, if Prover is honest, we get the distance as well as shortest path from s to each vertex. But we also need to check that there is no path to any vertex shorter than the ones claimed by Prover. We describe a protocol for this.

For $u, v \in V$ and $w \in [W]$, define the indicator function f as $f(u, v, w) = 1$ iff $A(u, v) = w$. Let $\tilde{f}$ be the $\mathbb{F}$-extension of f, for some large finite field $\mathbb{F}$.

Retain the definitions of B_d and N_d from last section with the definition of the polynomial p_d changed to

$$p_d(U) = \sum_{v \in B_d} \tilde{f}(v, U, d + 1 - \text{dist}_s[v]) \tag{20}$$

Hence, it still holds that

$$N_{d+1} = \{u \in V \setminus B_d : p_d(u) \neq 0\} \; . \tag{21}$$

Stream processing. The stream updates are of the form (u, v, w) denoting that $A(u, v) = w$. Verifier picks $r \in_R \mathbb{F}$ and maintains $\tilde{f}(v, r, w)$ for each $v \in V$ and $w \in [W]$. He also stores the set B_1 with dist_s labels set to 1 for each vertex in the set.

Help message processing and verification. This part is similar to the turnstile weight update protocol. Of course, this time, the Verifier computes $p_d(r)$ using Equation (20).

Error probability. Each polynomial p_d has degree $O(n)$. Verifier does sum-checks for $O(D)$ such polynomials. Choosing $|\mathbb{F}| \gg Dn$, we can make the error probability small by union bound.

Help and Verification costs. Since the degree of each p_d is at most n, the total hcost is $\tilde{O}(Dn)$. Verifier stores $\tilde{f}(v, r, w)$ for each $v \in V$ and $w \in [W]$, which requires $\tilde{O}(Wn)$ space. We also need to store all vertices as we go on assigning the distance labels. Hence, the total vcost of this protocol is $\tilde{O}(Wn)$.

▶ **Theorem 20.** *There is a $[Dn, Wn]$-scheme for weighted SSSP in the vanilla streaming model.*