Streaming Graph Computations with a Helpful Advisor

Graham Cormode[1], Michael Mitzenmacher[2⋆], and Justin Thaler[2⋆⋆]

[1] AT & T Labs – Research,
graham@research.att.com
[2] Harvard University, School of Engineering and Applied Sciences
{michaelm,jthaler}@seas.harvard.edu

Abstract. Motivated by the trend to outsource work to commercial cloud computing services, we consider a variation of the streaming paradigm where a streaming algorithm can be assisted by a powerful helper that can provide annotations to the data stream. We extend previous work on such *annotation models* by considering a number of graph streaming problems. Without annotations, streaming algorithms for graph problems generally require significant memory; we show that for many standard problems, including all graph problems that can be expressed with totally unimodular integer programming formulations, only constant memory is needed for single-pass algorithms given linear-sized annotations. We also obtain a protocol achieving *optimal* tradeoffs between annotation length and memory usage for matrix-vector multiplication; this result contributes to a trend of recent research on numerical linear algebra in streaming models.

1 Introduction

The recent explosion in the number and scale of real-world structured data sets including the web, social networks, and other relational data has created a pressing need to efficiently process and analyze massive graphs. This has sparked the study of graph algorithms that meet the constraints of the standard streaming model: restricted memory and the ability to make only one pass (or few passes) over adversarially ordered data. However, many results for graph streams have been negative, as many foundational problems require either substantial working memory or a prohibitive number of passes over the data [1]. Apparently most graph algorithms fundamentally require flexibility in the way they query edges, and therefore the combination of adversarial order and limited memory makes many problems intractable.

To circumvent these negative results, variants and relaxations of the standard graph streaming model have been proposed, including the Semi-Streaming [2],

⋆ This work was supported in part by NSF grants CCF-0915922 and CNS-0721491, and in part by grants from Yahoo! Research, Google, and Cisco, Inc.
⋆⋆ Supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program.

W-Stream [3], Sort-Stream [4], Random-Order [1], and Best-Order [5] models. In Semi-Streaming, memory requirements are relaxed, allowing space proportional to the number of vertices in the stream but not the number of edges. The W-Stream model allows the algorithm to write temporary streams to aid in computation. And, as their names suggest, the Sort-Stream, Random-Order, and Best-Order models relax the assumption of adversarially ordered input. The Best-Order model, for example, allows the input stream to be re-ordered arbitrarily to minimize the space required for the computation.

In this paper, our starting point is a relaxation of the standard model, closest to that put forth by Chakrabarti *et al.* [6], called the *annotation model*. Motivated by recent work on outsourcing of database processing, as well as commercial cloud computing services such as Amazon EC2, the annotation model allows access to a powerful advisor, or *helper* who observes the stream concurrently with the algorithm. Importantly, in many of our motivating applications, the helper is not a trusted entity: the commercial stream processing service may have executed a buggy algorithm, experienced a hardware fault or communication error, or may even be deliberately deceptive [5, 6]. As a result, we require our protocols to be *sound*: our *verifier* must detect any lies or deviations from the prescribed protocol with high probability.

The most general form of the annotation model allows the helper to provide additional annotations in the data stream *at any point* to assist the verifier, and one of the cost measures is the total length of the annotation. In this paper, however, we focus on the case where the helper's annotation arrives as a single message after both the helper and verifier have seen the stream. The helper's message is also processed as a stream, since it may be large; it often (but not always) includes a re-ordering of the stream into a convenient form, as well as additional information to guide the verifier. This is therefore stronger than the Best-Order model, which only allows the input to be reordered and no more; but it is weaker than the more general *online* model, because in our model the annotation appears only after the input stream has finished.

We argue that this model is of interest for several reasons. First, it requires minimal coordination between helper and verifier, since it is not necessary to ensure that annotation and stream data are synchronized. Second, it captures the case when the verifier uploads data to the cloud as it is collected, and later poses questions over the data to the helper. Under this paradigm, the annotation must come *after* the stream is observed. Third, we know of no non-trivial problems which separate the general online and our "at-the-end" versions of the model, and most prior results are effectively in this model.

Besides being practically motivated by outsourced computations, annotation models are closely related to Merlin-Arthur proofs with space-bounded verifiers, and studying what can (and cannot) be accomplished in these models is of independent interest.

Relationship to Other Work. Annotation models were first explicitly studied by Chakrabarti *et al.* in [6], and focused primarily on protocols for canonical problems in numerical streams, such as Selection, Frequency Moments, and

Frequent Items. The authors also provided protocols for some graph problems: counting triangles, connectedness, and bipartite perfect matching. The Best-Order Stream Model was put forth by Das Sarma et al. in [5]. They present protocols requiring logarithmic or polylogarithmic space (in bits) for several problems, including perfect matching and connectivity. Historical antecedents for this work are due to Lipton [7], who used fingerprinting methods to verify polynomial-time computations in logarithmic space. Recent work verifies shortest-path computations using cryptographic primitives, using polynomial space for the verifier [8].

Our Contributions. We identify two qualitatively different approaches to producing protocols for problems on graphs with n nodes and m edges. In the first, the helper directly proves matching upper and lower bounds on a quantity. Usually, proving one of the two bounds is trivial: the helper provides a feasible solution to the problem. But proving *optimality* of the feasible solution can be more difficult, requiring the use of structural properties of the problem. In the second, we simulate the execution of a non-streaming algorithm, using the helper to maintain the algorithm's internal data structures to control the amount of memory used by the verifier. The helper must provide the contents of the data structures so as to limit the amount of annotation required.

Using the first approach (Section 3), we show that only constant space and annotation linear in the input size m is needed to determine whether a directed graph is a DAG and to compute the size of a maximum matching. We describe this as an $(m, 1)$ protocol, where the first entry refers to the annotation size (which we also call the hcost) and the second to the memory required for the verifier (which we also call the vcost). Our maximum matching result significantly extends the bipartite perfect matching protocol of [6], and is tight for dense graphs, in the sense that there is a lower bound on the *product* of hcost and vcost of $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ bits for this problem. Second, we define a streaming version of the linear programming problem, and provide an $(m, 1)$ protocol. By exploiting duality, we hence obtain $(m, 1)$ protocols for many graph problems with totally unimodular integer programming formulations, including shortest s-t path, max-flow, min-cut, and minimum-weight bipartite perfect matching. We also show all are tight by proving lower bounds of $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ bits for all four problems. A more involved protocol obtains *optimal* tradeoffs between annotation cost and working memory for dense LPs and matrix-vector multiplication; this complements recent results on approximate linear algebra in streaming models (see e.g. [9, 10]).

For the second approach (Section 4), we make use of the idea of "memory checking" due to Blum et al. [11], which allows a small-space verifier to outsource data storage to an untrusted server. We present a general simulation theorem based on this checker, and obtain as corollaries tight protocols for a variety of canonical graph problems. In particular, we give an $(m, 1)$ protocol for verifying a minimum spanning tree, an $(m + n \log n, 1)$ protocol for single-source shortest paths, and an $(n^3, 1)$ protocol for all-pairs shortest paths. We provide a lower bound of $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ bits for the latter two problems, and an identical lower bound for MST when the edge weights can be given incrementally. While

powerful, this technique has its limitations: there does not seem to be any generic way to obtain the same kind of tradeoffs observed above. Further, there are some instances where direct application of memory checking does not achieve the best bounds for a problem. We demonstrate this by presenting an $(n^2 \log n, 1)$ protocol to find the diameter of a graph; this protocol leverages the ability to use randomized methods to check computations more efficiently than via generating or checking a deterministic witness. In this case, we rely on techniques to verify matrix-multiplication in quadratic time, and show that this is tight via a nearly matching lower bound for diameter of $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$.

In contrast to problems on numerical streams, where it is often trivial to obtain $(m, 1)$ protocols by replaying the stream in sorted order, it transpires that achieving linear-sized annotations with logarithmic space is more challenging for many graph problems. Simply providing the solution (e.g. a graph matching or spanning tree) is insufficient, since we have the additional burden of demonstrating that this solution is *optimal*. A consequence is that we are able to provide solutions to several problems for which no solution is known in the best-order model (even though one can reorder the stream in the best-order model so that the "solution" edges arrive first).

2 Model and Definitions

Consider a data stream $\mathcal{A} = \langle a_1, a_2, \ldots, a_m \rangle$ with each a_i in some universe $\mathcal{U}$. Consider a probabilistic *verifier* $\mathcal{V}$ who observes $\mathcal{A}$ and a deterministic *helper* $\mathcal{H}$ who also observes $\mathcal{A}$ and can send a message $\hbar$ to $\mathcal{V}$ after $\mathcal{A}$ has been observed by both parties. This message, also referred to as an *annotation*, should itself be interpreted as a data stream that is parsed by $\mathcal{V}$, which may permit $\mathcal{V}$ to use space sublinear in the size of the annotation itself. That is, $\mathcal{H}$ provides an annotation $\hbar(\mathcal{A}) = (\hbar_1(\mathcal{A}), \hbar_2(\mathcal{A}), \ldots \hbar_\ell(\mathcal{A}))$.

We study randomized streaming protocols for computing functions $f(\mathcal{A}) \to \mathbb{Z}$. Specifically, assume $\mathcal{V}$ has access to a private random string $\mathcal{R}$ and at most $w(m)$ machine words of working memory, and that $\mathcal{V}$ has one-way access to the input $\mathcal{A} \cdot \hbar$, where $\cdot$ represents concatenation. Denote the output of protocol $\mathcal{P}$ on input $\mathcal{A}$, given helper $\hbar$ and random string $\mathcal{R}$, by $out(\mathcal{P}, \mathcal{A}, \mathcal{R}, \hbar)$. We allow $\mathcal{V}$ to output $\perp$ if $\mathcal{V}$ is not convinced that the annotation is valid. We say that $\hbar$ is *valid* for $\mathcal{A}$ with respect to $\mathcal{P}$ if $Pr_{\mathcal{R}}(out(\mathcal{P}, \mathcal{A}, \mathcal{R}, \hbar) = f(\mathcal{A})) = 1$, and we say that $\hbar$ is δ-*invalid* for $\mathcal{A}$ with respect to $\mathcal{P}$ if $Pr_{\mathcal{R}}(out(\mathcal{P}, \mathcal{A}, \mathcal{R}, \hbar) \neq \perp) \leq \delta$. We say that $\hbar$ is a *valid helper* if $\hbar$ is valid for all $\mathcal{A}$. We say that $\mathcal{P}$ is a valid protocol for f if

1. There exists at least one valid helper $\hbar$ with respect to $\mathcal{P}$ and
2. For all helpers $\hbar'$ and all streams $\mathcal{A}$, either $\hbar'$ is valid for $\mathcal{A}$ or $\hbar'$ is $\frac{1}{3}$-invalid for $\mathcal{A}$.

Conceptually, $\mathcal{P}$ is a valid protocol for f if for each stream $\mathcal{A}$ there is *at least* one way to convince $\mathcal{V}$ of the true value of $f(\mathcal{A})$, and $\mathcal{V}$ rejects all other annotations

as invalid (this differs slightly from [6] to allow for multiple $\hbar$'s that can convince $\mathcal{V}$). The constant $\frac{1}{3}$ can be any constant less than $\frac{1}{2}$.

Let $\hbar$ be a valid helper chosen to minimize the length of $\hbar(\mathcal{A})$ for all $\mathcal{A}$. We define the help cost $\text{hcost}(\mathcal{P})$ to be the maximum length of $\hbar$ over all $\mathcal{A}$ of length m, and the verification cost $\text{vcost}(P) = w(m)$, the amount of working memory used by the protocol P. All costs are expressed in machine words of size $\Theta(\log m)$ bits, i.e. we assume any quantity polynomial in the input size can be stored in a constant number of words; in contrast, lower bounds are expressed in bits. We say that $\mathcal{P}$ is an (h, v) protocol for f if $\mathcal{P}$ is valid and $\text{hcost}(\mathcal{A}) = O(h + 1)$, $\text{vcost}(\mathcal{A}) = O(v + 1)$. While both hcost and vcost are natural costs for such protocols, we often aim to achieve a vcost of $O(1)$ and then minimize hcost. In other cases, we show that hcost can be decreased by increasing vcost, and study the tradeoff between these two quantities.

In some cases, f is not a function of $\mathcal{A}$ alone; instead it depends on $\mathcal{A}$ *and* h. In such cases, $\mathcal{V}$ should simply *accept* if convinced that the annotation has the correct properties, and output $\perp$ otherwise. We use the same terminology as before, and say that $\mathcal{P}$ is a valid protocol if there is a valid helper and any $\hbar'$ that is not valid for $\mathcal{A}$ is $\frac{1}{3}$-invalid for $\mathcal{A}$.

In this paper we primarily consider graph streams, which are streams whose elements are edges of a graph G. More formally, consider a stream $\mathcal{A} = \langle e_1, e_2, \ldots, e_m \rangle$ with each $e_i \in [n] \times [n]$. Such a stream defines a (multi)graph $G = (V, E)$ where $V = \{v_1, ..., v_n\}$ and E is the (multi)set of edges that naturally corresponds to $\mathcal{A}$. We use the notation $\{i : m(i)\}$ for the multiset in which i appears with multiplicity $m(i)$. Finally, we will sometimes consider graph streams with directed edges, and sometimes with weighted edges; in the latter case each edge $e_i \in [n] \times [n] \times \mathbb{Z}_+$.

2.1 Fingerprints

Our protocols make careful use of *fingerprints*, permutation-invariant hashes that can be efficiently computed in a streaming fashion. They determine in small space (with high probability) whether two streams have identical frequency distributions. They are the workhorse of algorithms proposed in earlier work on streaming models with an untrusted helper [5–7, 13]. We sometimes also need the fingerprint function to be linear.

Definition 1 (Fingerprints). *A fingerprint of a multiset $M = \{i : m(i)\}$ where each $i \in [q]$ for some known upper bound q is defined as a computation over the finite field with p elements, $\mathbb{F}_p$, as $\mathfrak{f}_{p,\alpha}(M) = \sum_{i=1}^{q} m(i)\alpha^i$, where α is chosen uniformly at random from $\mathbb{F}_p$. We typically leave p, α implicit, and just write $\mathfrak{f}(M)$.*

Some properties of $\mathfrak{f}$ are immediate: it is linear in M, and can easily be computed incrementally as elements of $[q]$ are observed in a stream one by one. The main property of $\mathfrak{f}$ is that $\Pr[\mathfrak{f}(M) = \mathfrak{f}(M') | M \neq M'] \leq q/p$ over the random choice of α (due to standard properties of polynomials over a field). Therefore, if p is sufficiently large, say, polynomial in q and in an (assumed) upper bound on

the multiplicities $m(i)$, then this event happens with only polynomially small probability. For cases when the domain of the multisets is not $[q]$, we either establish a bijection to $[q]$ for an appropriate value of q, or use a hash function to map the domain onto a large enough $[q]$ such that there are no collisions with high probability (whp). In all cases, p is chosen to be $O(1)$ words.

A common subroutine of many of our protocols forces $\mathcal{H}$ to provide a "label" $l(u)$ for each node upfront, and then replay the edges in E, with each edge (u, v) annotated with $l(u)$ and $l(v)$ so that each instance of each node v appears with the *same* label $l(v)$.

Definition 2. *We say a list of edges E' is* label-augmented *if (a) E' is preceded by a sorted list of all the nodes $v \in V$, each with a value $l(v)$ and $\deg(v)$, where $l(v)$ is the label of v and $\deg(v)$ is claimed to be the degree of v; and (b) each edge $e = (u, v)$ in E' is annotated with a pair of symbols $l(e, u)$ and $l(e, v)$. We say a list of label-augmented edges E' is valid if for all edges $e = (u, v)$, $l(e, u) = l(u)$ and $l(e, v) = l(v)$; and $E' = E$, where E is the set of edges observed in the stream A.*

Lemma 1 (Consistent Labels). *There is a valid $(m, 1)$ protocol that accepts any valid list of label-augmented edges.*

Proof. $\mathcal{V}$ uses the annotation from Definition 2 (a) to make a fingerprint of the multiset $S_1 := \{(u, l(u)) : \deg(u)\}$. $\mathcal{V}$ also maintains a fingerprint f_1 of all $(u, l(e, u))$ pairs seen while observing the edges of L. If $f_1 = \mathfrak{f}(S_1)$ then (whp) each node u must be presented with label $l(e, u) = l(u)$ every time it is reported in an edge e (and moreover u must be reported in exactly $\deg(u)$ edges), else the multiset of observed (node, label) pairs would not match S_1. Finally, $\mathcal{V}$ ensures that $E' = E$ by checking that $\mathfrak{f}(E) = \mathfrak{f}(E')$. □

3 Directly Proving Matching Upper and Lower Bounds

3.1 Warmup: Topological Ordering and DAGs

A (directed) graph G is a DAG if and only if G has a *topological ordering*, which is an ordering of V as $v_1, \ldots v_n$ such that for every edge (v_i, v_j) we have $i < j$ [14, Section 3.6]. Hence, if G is a DAG, $\mathcal{H}$ can prove it by providing a topological ordering. If G is not a DAG, $\mathcal{H}$ can provide a directed cycle as witness.

Theorem 1. *There is a valid $(m, 1)$ protocol to determine if a graph is a DAG.*

Proof. If G is not a DAG, $\mathcal{H}$ provides a directed cycle C as (v_1, v_2), (v_2, v_3) $\ldots$ (v_k, v_1). To ensure $C \subseteq E$, $\mathcal{H}$ then provides $E \setminus C$, allowing $\mathcal{V}$ to check that $\mathfrak{f}(C \cup (E \setminus C)) = \mathfrak{f}(E)$.
If G is a DAG, let $v_1, \ldots v_n$ be a topological ordering of G. We require $\mathcal{H}$ to replay the edges of G, with edge (v_i, v_j) annotated with the ranks of v_i and v_j i.e. i and j. We ensure $\mathcal{H}$ provides consistent ranks via the Consistent Labels protocol of Lemma 1, with the ranks as "labels". If any edge (v_i, v_j) is presented with $j > i$, $\mathcal{V}$ rejects immediately. □

3.2 Maximum Matching

We give an $(m, 1)$ protocol for maximum matching which leverages the combinatorial structure of the problem. Previously, matching was only studied in the bipartite case, where an $(m, 1)$ protocol and a lower bound of $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ bits for dense graphs were shown [6, Theorem 11]. The same lower bound applies to the more general problem of maximum matching, so our protocol is tight up to logarithmic factors.

The protocol shows matching upper and lower bounds on the size of the maximum matching. Any feasible matching presents a lower bound. For the upper bound we appeal to the Tutte-Berge formula [15, Chapter 24]: the size of a maximum matching of a graph $G = (V, E)$ is equal to $\frac{1}{2} \min_{V_S \subseteq V}(|V_S| - \text{occ}(G - V_S) + |V|)$, where $G - V_S$ is the subgraph of G obtained by deleting the vertices of V_S and all edges incident to them, and $\text{occ}(G - V_S)$ is the number of components in the graph $G - V_S$ that have an odd number of vertices. So for any set of nodes V_S, $\frac{1}{2}(|V_S| - \text{occ}(G - V_S) + |V|)$ is an upper bound on the size of the maximum matching, and there exists some V_S for which this quantity equals the size of a maximum matching M. Conceptually, providing both V_S and M, $\mathcal{H}$ proves that the maximum matching size is M. Additionally, $\mathcal{H}$ has to provide a proof of the value of $\text{occ}(G - V_S)$ to $\mathcal{V}$. We omit a full proof; the main technique needed is careful application of fingerprints.

Theorem 2. *There is a valid $(m, 1)$ protocol for maximum matching. Moreover, any protocol for max-matching requires $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ bits.*

3.3 Linear Programming and TUM Integer Programs

We present protocols to solve linear programming problems in our model leveraging the theory of LP duality. This leads to non-trivial schemes for a variety of graph problems.

Definition 3. *Given a data stream $\mathcal{A}$ containing entries of vectors $\mathbf{b} \in \mathbb{R}^b$, $\mathbf{c} \in \mathbb{R}^c$, and non-zero entries of a $b \times c$ matrix A in some arbitrary order, possibly interleaved. Each item in the stream indicates the index of the object it pertains to. The LP streaming problem on $\mathcal{A}$ is to determine the value of the linear program $\min\{\mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$.*

We present our protocol as if each entry of each object appears at most once (if an entry does not appear, it is assumed to be zero). When this is not the case, the final value for that entry is interpreted as the *sum* of all corresponding values in the stream.

Theorem 3. *There is a valid $(|A|, 1)$ protocol for the LP streaming problem, where $|A|$ is the number of non-zero entries in the constraint matrix A of $\mathcal{A}$.*

Proof. The protocol shows an upper bound by providing a primal-feasible solution $\mathbf{x}$, and a lower bound by providing a dual-feasible solution $\mathbf{y}$. When the

value of both solutions match, $\mathcal{V}$ is convinced that the optimal value has been found.

From the stream, $\mathcal{V}$ fingerprints the sets $S_A = \{(i, j, A_{i,j})\}$, $S_B = \{(i, \mathbf{b}_i)\}$ and $S_C = \{(i, \mathbf{c}_j)\}$. Then $\mathcal{H}$ provides all pairs of values $\mathbf{c}_j, \mathbf{x}_j, 1 \leq j \leq c$, with each $\mathbf{x}_j$ additionally annotated with $|A_{\cdot j}|$, the number of non-zero entries in column j of A. This allows $\mathcal{V}$ to fingerprint the multiset $S_X = \{(j, \mathbf{x}_j) : |A_{\cdot j}|\}$ and calculate the solution cost $\sum_{j=1}^{b} \mathbf{c}_j \mathbf{x}_j$.

To prove feasibility, for each row i of A, $A_{i\cdot}$, $\mathcal{H}$ sends $\mathbf{b}_i$, then (the non-zero entries of) $A_{i\cdot}$ so that A_{ij} is annotated with $\mathbf{x}_j$. This allows the ith constraint to be checked easily in constant space. $\mathcal{V}$ fingerprints the values given by $\mathcal{H}$ for A, $\mathbf{b}$, and $\mathbf{c}$, and compares them to those for the stream. A single fingerprint of the multiset of values presented for $\mathbf{x}$ over all rows is compared to $\mathfrak{f}(S_X)$. The protocol accepts $\mathbf{x}$ as feasible if all constraints are met and all fingerprint tests pass.

Correctness follows by observing that the agreement with $\mathfrak{f}(A)$ guarantees (whp) that each entry of A is presented correctly and no value is omitted. Since $\mathcal{H}$ presents each entry of $\mathbf{b}$ and $\mathbf{c}$ once, in index order, the fingerprints $\mathfrak{f}(S_B)$ and $\mathfrak{f}(S_C)$ ensure that these values are presented correctly. The claimed $|A_{\cdot j}|$ values must be correct: if not, then the fingerprints of either S_X or S_A will not match the multisets provided by $\mathcal{H}$. $\mathfrak{f}(S_X)$ also ensures that each time $\mathbf{x}_j$ is presented, the same value is given (similar to Lemma 1).

To prove that $\mathbf{x}$ is primal-optimal, it suffices to show a feasible solution $\mathbf{y}$ to the dual A^T so that $c^T\mathbf{x} = b^T\mathbf{y}$. Essentially we repeat the above protocol on the dual, and check that the claimed values are again consistent with the fingerprints of S_A, S_B, S_C. □

For any graph problem that can be formulated as a linear program in which each entry of A, $\mathbf{b}$, and $\mathbf{c}$ can be derived as a linear function of the nodes and edges, we may view each edge in a graph stream $\mathcal{A}$ as providing an update to values of one or more entries of A, $\mathbf{b}$, and $\mathbf{c}$. Therefore, we immediately obtain a protocol for problems of this form via Theorem 3. More generally, we obtain protocols for problems formulated as *totally unimodular integer programs* (TUM IPs), since optimality of a feasible solution is shown by a matching feasible solution of the dual of its LP relaxation [16].

Corollary 1. *There is a valid* $(|A|, 1)$ *protocol for any graph problem that can be formulated as a linear program or TUM IP in which each entry of* A, $\mathbf{b}$, *and* $\mathbf{c}$ *is a linear function of the nodes and edges of graph.*

This follows immediately from Theorem 3 and the subsequent discussion: note that the linearity of the fingerprinting builds fingerprints of S_A, S_B and S_C, so $\mathcal{H}$ presents only their (aggregated) values, not information from the unaggregated graph stream.

Corollary 2. *Shortest* $s - t$ *path, max-flow, min-cut, and minimum weight bipartite perfect matching (MWBPM) all have valid* $(m, 1)$ *protocols. For all four problems, a lower bound of* $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ *bits holds for dense graphs.*

Proof. The upper bound follows from the previous corollary because all the problems listed possess formulations as TUM IPs and moreover the constraint matrix in each case has $O(m+n)$ non-zero entries. For example, for max-flow, $\mathbf{x}$ gives the flow on each edge, and the weight of each edge in the stream contributes (linearly) to constraints on the capacity of that edge, and the flow through incident nodes.

The lower bound for MWBPM, max-flow, and min-cut holds from [6, Theorem 11] which argues $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ bits for bipartite perfect matching, and straightforward reductions of bipartite perfect matching to all three problems, see e.g. [14, Theorem 7.37]. The lower bound for shortest $s-t$ path follows from a straightforward reduction from INDEX, for which a lower bound linear in $\text{hcost} \cdot \text{vcost}$ was proven in [6, Theorem 3.1]. Given an instance (x, k) of INDEX where $x \in \{0,1\}^{n^2}$, $k \in [n^2]$, we construct graph G, with $V_G = [n+2]$, and $E_G = E_A \cup E_B$. Alice creates $E_A = \{(i,j) : x_{f(i,j)=1}\}$ from x alone, where f is a 1-1 correspondence $[n] \times [n] \to [n^2]$. Bob creates $E_B = \{(n+1, i), (j, n+2)\}$ using $f(i,j) = k$. The shortest path between nodes $n+1$ and $n+2$ is 3 if $x_k = 1$ and is 4 or more otherwise. This also implies that any approximation within $\sqrt{4/3}$ requires $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ (better inapproximability constants may be possible). □

Conceptually, the above protocols for solving the LP streaming problem are straightforward: $\mathcal{H}$ provides a primal solution, potentially repeating it once for each row of A to prove feasibility, and repeats the protocol for the dual. There are efficient protocols for the problems listed in the corollary since the constraint matrices of their IP formulations are sparse. For dense constraint matrices, however, the bottleneck is proving feasibility. We observe that computing $A\mathbf{x}$ reduces to computing b inner-product computations of vectors of dimension c. There are $(c^\alpha, c^{1-\alpha})$ protocols to verify such inner-products [6]. But we can further improve on this since one of the vectors is held constant in each of the tests. This reduces the space needed by $\mathcal{V}$ to run these checks in parallel; moreover, we prove a lower bound of $\text{hcost} \cdot \text{vcost} = \Omega(\min(c,b)^2)$ bits, and so obtain an *optimal* tradeoff for square matrices, up to logarithmic factors. We omit the proof for space reasons.

Theorem 4. *Given a $b \times c$ matrix A and a c dimensional vector $\mathbf{x}$, the product $A\mathbf{x}$ can be verified with a valid $(bc^\alpha, c^{1-\alpha})$ protocol. Moreover, any such protocol requires $\text{hcost} \cdot \text{vcost} = \Omega(\min(c,b)^2)$ bits for dense matrices.*

Corollary 3. *For $c \geq b$ there is a valid $(c^{1+\alpha}, c^{1-\alpha})$ protocol for the LP streaming problem.*

Proof. This follows by using the protocol of Theorem 4 to verify $A\mathbf{x} \leq \mathbf{b}$ and $A^T\mathbf{y} \geq \mathbf{c}$ within the protocol of Theorem 3. The cost is $(bc^\alpha + cb^\alpha, c^{1-\alpha} + b^{1-\alpha})$, so if $c \geq b$, this is dominated by $(c^{1+\alpha}, c^{1-\alpha})$ (symmetrically, if $b > c$, the cost is $(b^{1+\alpha}, b^{1-\alpha})$). □

Our protocol for linear programming relied on only two properties: strong duality, and the ability to compute the value of a solution $\mathbf{x}$ and check feasibility via matrix-vector multiplication. Such properties also hold for more general convex optimization problems, such as quadratic programming and a large class of

second-order cone programs. Thus, similar results apply for these mathematical programs, motivated by applications in which weak peripheral devices or sensors perform error correction on signals. We defer full details from this presentation.

Theorem 4 also implies the existence of protocols for graph problems where *both* hcost and vcost are sublinear in the size of the input (for dense graphs). These include:

– An $(n^{1+\alpha}, n^{1-\alpha})$ protocol for verifying that λ is an eigenvalue of the adjacency matrix A or the Laplacian L of G: $\mathcal{H}$ provides the corresponding eigenvector x, and $\mathcal{V}$ can use the protocol of Theorem 4 to verify that $Ax = \lambda x$ or $Lx = \lambda x$.

– An $(n^{1+\alpha}, n^{1-\alpha})$ protocol for the problem of determining the effective resistance between designated nodes s and t in G where the edge weights are resistances. The problem reduces to solving an $n \times n$ system of linear equations [17].

4 Simulating Non-Streaming Algorithms

Next, we give protocols by appealing to known non-streaming algorithms for graph problems. At a high level, we can imagine the helper running an algorithm on the graph, and presenting a "transcript" of operations carried out by the algorithm as the proof to $\mathcal{V}$ that the final result is correct. Equivalently, we can imagine that $\mathcal{V}$ runs the algorithm, but since the data structures are large, they are stored by $\mathcal{H}$, who provides the contents of memory needed for each step. There may be many choices of the algorithm to simulate and the implementation details of the algorithm: our aim is to choose ones that result in smaller annotations.

Our main technical tool is the off-line memory checker of Blum et al. [11], which we use to efficiently verify a sequence of accesses to a large memory. Consider a *memory transcript* of a sequence of read and write operations to this memory (initialized to all zeros). Such a transcript is *valid* if each read of address i returns the last value written to that address. The protocol of Blum et al. requires each read to be accompanied by the timestamp of the last write to that address; and to treat each operation (read or write) as a read of the old value followed by the write of a new value. Then it suffices to ensure that a fingerprint of all write operations (augmented with timestamps) matches a fingerprint of all read operations (using the provided timestamps), along with some simple local checks on timestamps. Consequently, any valid (timestamp-augmented) transcript is accepted by $\mathcal{V}$, while any invalid transcript is rejected by $\mathcal{V}$ with high probability.

We use this memory checker to obtain the following general simulation result.

Theorem 5. *Suppose P is a graph problem possessing a non-randomized algorithm $\mathcal{M}$ in the random-access memory model that, when given $G = (V, E)$ in adjacency list or adjacency matrix form, outputs $P(G)$ in time $t(m, n)$, where $m = |E|$ and $n = |V|$. Then there is an $(m + t(m, n), 1)$ protocol for P.*

Proof (sketch). $\mathcal{H}$ first repeats (the non-zero locations of) a valid adjacency list or matrix representation G, as writes to the memory (which is checked by $\mathcal{V}$); $\mathcal{V}$ uses fingerprints to ensure the edges included in the representation precisely correspond to those that appeared in the stream, and can use local checks to ensure the representation is otherwise valid. This requires $O(m)$ annotation and effectively initializes memory for the subsequent simulation. Thereafter, $\mathcal{H}$ provides a valid augmented transcript T' of the read and write operations performed by algorithm $\mathcal{M}$; $\mathcal{V}$ rejects if T' is invalid, or if any read or write operation executed in T' does not agree with the prescribed action of $\mathcal{M}$. As only one read or write operation is performed by $\mathcal{M}$ in each timestep, the length of T' is $O(t(m,n))$, resulting in an $(m + t(m,n), 1)$ protocol for P. □

Although Theorem 5 only allows the simulation of deterministic algorithms, $\mathcal{H}$ can non-deterministically "guess" an optimal solution S and prove optimality by invoking Theorem 5 on a (deterministic) algorithm that merely checks whether S is optimal. Unsurprisingly, it is often the case that the best-known algorithms for verifying optimality are more efficient than those finding a solution from scratch (see e.g. the MST protocol below); therein lies much of the power of the simulation theorem.

Theorem 6. *There is a valid* $(m, 1)$ *protocol to find a minimum cost spanning tree; a valid* $(m + n \log n, 1)$ *protocol to verify single-source shortest paths; and a valid* $(n^3, 1)$ *protocol to verify all-pairs shortest paths.*

Proof. We first prove the bound for MST. Given a spanning tree T, there exists a linear-time algorithm $\mathcal{M}$ for verifying that T is minimum (see e.g. [18]). Let $\mathcal{M}'$ be the linear-time algorithm that, given G and a subset of edges T in adjacency matrix form, first checks that T is a spanning tree by ensuring $|T| = n-1$ and T is connected (by using e.g. breadth-first search), and then executes $\mathcal{M}$ to ensure T is minimum. We obtain an $(m, 1)$ protocol for MST by having $\mathcal{H}$ provide a minimum spanning tree T and using Theorem 5 to simulate algorithm $\mathcal{M}'$.

The upper bound for single-source shortest path follows from Theorem 5 and the fact that there exist implementations of Djikstra's algorithm that run in time $m + n \log n$. The upper bound for all-pairs shortest paths also follows from Theorem 5 and the fact that the Floyd-Warshall algorithm runs in time $O(n^3)$. □

We now provide near-matching lower bounds for all three problems.

Theorem 7. *Any protocol for verifying single-source or all pairs shortest paths requires* $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ *bits. Additionally, if edge weights may be specified incrementally, then an identical lower bound holds for MST.*

Proof. The lower bounds for single-source and all-pairs shortest paths are inherited from shortest $s - t$ path (Corollary 2).

To prove the lower bound for MST, we present a straightforward reduction from an instance of INDEX, (x, k), where $x \in \{0,1\}^{n^2}$, $k \in [n^2]$. Alice will construct a graph G, with $V_G = [n]$, and $E_G = E_A$. Bob will then construct two

graphs, G_1 and G_2, with $E_{G_1} = E_A \cup E_{B_1}$ and $E_{G_2} = E_A \cup E_{B_2}$. If edge (i,j) is in $E_A \cap E_{B_1}$, then we interpret this to mean that the weight of edge (i,j) in E_{G_1} is the *sum* of its weights in E_A and E_{B_1}. Below, we will write (i,j,w) to denote an edge between nodes i and j with weight w.

Alice creates $E_A = \{(i,j,1) : x_{f(i,j)=1}\}$ from x alone, where f is a bijection $[n] \times [n] \to [n^2]$. Bob creates $E_{B_1} = \{(u,v,3) : f(u,v) \neq k\}$, and $E_{B_2} = E_{B_1} \cup \{(i,j,1)\}$, where (i,j) is the edge satisfying $f(i,j) = k$. Edge (i,j), if it exists, is the lowest-weight edge in E_{G_1}, and hence (i,j) is in any min-cost spanning tree of G_1 if and only if $x_k = 1$. In contrast, (i,j) is always in the min-cost spanning tree of G_2. Therefore, if $x_k = 1$, then the minimum spanning tree of G_2 will be of higher cost than that of G_1, because the weight of (i,j) is 1 in E_{G_1} and 2 in E_{G_2}. And if $x_k = 0$, then the mininmum spanning tree of G_2 will be of lower cost than that of G_1, because the weight of edge (i,j) will be ∞ in G_1 and 1 in G_2. Thus, by comparing the cost of the MSTs of G_1 and G_2, Bob can extract the value of x_k. The lower bound now follows from the hardness of INDEX [6, Theorem 3.1]. □

Diameter. The diameter of G can be verified via the all-pairs shortest path protocol above, but the next protocol improves over the memory checking approach.

Theorem 8. *There is a valid* $(n^2 \log n, 1)$ *protocol for computing graph diameter. Further, any protocol for diameter requires* $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ *bits.*

Proof. [6, Theorem 5.2] gives an $(n^2 \log l, 1)$ protocol for verifying that $A^l = B$ for a matrix A presented in a data stream and for any positive integer l. Note that if A is the adjacency matrix of G; then $(I + A)^l_{ij} \neq 0$ if and only if there is a path of length at most l from i to j. Therefore, the diameter of G is equal to the unique $l > 0$ such that $(I + A)^l_{ij} \neq 0$ for all (i,j), while $(I + A)^{l-1}_{ij} = 0$ for some (i,j). Our protocol requires $\mathcal{H}$ to send l to $\mathcal{V}$, and then run the protocol of [6, Theorem 5.2] twice to verify that l is as claimed. Since the diameter is at most $n - 1$, this gives an $(n^2 \log n, 1)$ protocol.

We prove the lower bound via a reduction from an instance of INDEX, (x,k), where $x \in \{0,1\}^{n^2/4}$, $k \in [n^2/4]$. Alice creates a bipartite graph $G = (V,E)$ from x alone: she includes edge (i,j) in E if and only if $x_{f(i,j)} = 1$, where f is a bijection from edges to indices. Bob then adds to G two nodes L and R, with edges from L to each node in the left partite set, edges from R to each node in the right partite set, and an edge between L and R. This ensures that the graph is connected, with diameter at most 3. Finally, Bob appends a path of length 2 to node i, and a path of length 2 to node j, where $f(i,j) = k$. If $x_k = 0$, then the diameter is now 7, while if $x_k = 1$, the diameter is 5. The lower bound follows from the hardness of INDEX [6, Theorem 3.1] (this also shows that any protocol to approximate diameter better than $\sqrt{1.4}$ requires $\text{hcost} \cdot \text{vcost} = \Omega(n^2)$ bits; no effort has been made to optimize the inapproximability constant). □

5 Conclusion and Future Directions

In this paper, we showed that a host of graph problems possess streaming protocols requiring only constant space and linear-sized annotations. For many applications of the annotation model, the priority is to minimize vcost, and these protocols achieve this goal. However, these results are qualitatively different from those involving numerical streams in the earlier work [6]: for the canonical problems of heavy hitters, frequency moments, and selection, it is trivial to achieve an $(m, 1)$ protocol by having $\mathcal{H}$ replay the stream in sorted ("best") order. The contribution of [6] is in presenting protocols obtaining optimal tradeoffs between hcost and vcost in which both quantities are sublinear in the size of the input. There are good reasons to seek these tradeoffs. For example, consider a verifier with access to a few MBs or GBs of working memory. If an $(m, 1)$ protocol requires only a few KBs of space, it would be desirable to use more of the available memory to significantly reduce the running time of the verification protocol.

In contrast to [6], it is non-trivial to obtain $(m, 1)$ protocols for the graph problems we consider, and we obtain tradeoffs involving sublinear values of hcost and vcost for some problems with an algebraic flavor (e.g. matrix-vector multiplication, computing effective resistances, and eigenvalues of the Laplacian). We thus leave as an open question whether it is possible to obtain such tradeoffs for a wider class of graph problems, and in particular if the use of memory checking can be adapted to provide tradeoffs.

A final open problem is to ensure that the work of $\mathcal{H}$ is scalable. In motivating settings such as Cloud computing environments, the data is very large, and $\mathcal{H}$ may represent a *distributed* cluster of machines. It is a challenge to show that these protocols can be executed in a model such as the MapReduce framework.

Acknowledgements. We thank Moni Naor for suggesting the use of memory checking.

References

1. A. McGregor, "Graph mining on streams," in *Encyc. of Database Systems*. Springer, 2009.
2. J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, "On graph problems in a semi-streaming model," *Theor. Comput. Sci.*, vol. 348, no. 2, pp. 207–216, 2005.
3. C. Demetrescu, I. Finocchi, and A. Ribichini, "Trading off space for passes in graph streaming problems," in *SODA*, 2006, pp. 714–723.
4. G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl, "On the streaming model augmented with a sorting primitive," in *FOCS*, 2004, pp. 540–549.
5. A. Das Sarma, R. J. Lipton, and D. Nanongkai, "Best-order streaming model," in *Theory and Applications of Models of Computation*, 2009, pp. 178–191.
6. A. Chakrabarti, G. Cormode, and A. Mcgregor, "Annotations in data streams," in *ICALP*, 2009, pp. 222–234.
7. R. J. Lipton, "Efficient checking of computations," in *STACS*, 1990, pp. 207–215.
8. M. Yiu, Y. Lin, and K. Mouratidis, "Efficient verification of shortest path search via authenticated hints," in *ICDE*, 2010.

9. K. L. Clarkson and D. P. Woodruff, "Numerical linear algebra in the streaming model," in *STOC*, 2009, pp. 205–214.
10. T. Sarlos, "Improved approximation algorithms for large matrices via random projections," in *IEEE FOCS*, 2006.
11. M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor, "Checking the correctness of memories," pp. 90–99, 1995.
12. R. Freivalds, "Fast probabilistic algorithms," in *MFCS*, 1979, pp. 57–69.
13. R. J. Lipton, "Fingerprinting sets," Princeton University, Tech. Rep. Cs-tr-212-89, 1989.
14. J. Kleinberg and E. Tardos, *Algorithm Design*, 2005.
15. A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, 2003.
16. ——, *Theory of linear and integer programming*, 1986.
17. B. Bollobas, *Modern Graph Theory*. Springer, 1998.
18. V. King, "A simpler minimum spanning tree verification algorithm," *Algorithmica*, vol. 18, no. 2, pp. 263–270, 1997.