Streaming Verification of Graph Properties*†

Amirali Abdullah[1], Samira Daruki[2], Chitradeep Dutta Roy[3], and Suresh Venkatasubramanian[4]

1 **Department of Mathematics, University of Michigan, USA**
2 **School of Computing, University of Utah, USA**
3 **School of Computing, University of Utah, USA**
4 **School of Computing, University of Utah, USA**

Abstract

Streaming interactive proofs (SIPs) are a framework for outsourced computation. A computationally limited streaming client (the verifier) hands over a large data set to an untrusted server (the prover) in the cloud and the two parties run a protocol to confirm the correctness of result with high probability. SIPs are particularly interesting for problems that are hard to solve (or even approximate) well in a streaming setting. The most notable of these problems is finding maximum matchings, which has received intense interest in recent years but has strong lower bounds even for constant factor approximations. In this paper, we present efficient streaming interactive proofs that can verify maximum matchings *exactly*. Our results cover all flavors of matchings (bipartite/non-bipartite and weighted). In addition, we also present streaming verifiers for approximate metric TSP. In particular, these are the first efficient results for weighted matchings and for metric TSP in any streaming verification model.

1998 ACM Subject Classification F.1.2 Modes of Computation, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases streaming interactive proofs, verification, matching, travelling salesman problem, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.3

1 Introduction

The shift from direct computation to outsourcing in the cloud has led to new ways of thinking about massive scale computation. In the *verification* setting, computational effort is split between a computationally weak client (the *verifier*) who owns the data and wants to solve a desired problem, and a more powerful server (the *prover*) which performs the computations. Here the client has only limited (streaming) access to the data, as well as a bounded ability to talk with the server (measured by the amount of communication), but wishes to verify the correctness of the prover's answers. This model can be viewed as a streaming modification of a classic interactive proof system (a streaming IP, or SIP), and has been the subject of a number of papers [26, 47, 23, 17, 22, 16, 38, 39] that have established sublinear (verifier) space and communication bounds for classic problems in streaming and data analysis.

In this paper, we present streaming interactive proofs for graph problems that are traditionally hard for streaming, such as for the maximum matching problem (in bipartite and general graphs, both weighted and unweighted) as well for approximating the traveling

* A full version of the paper is available at `http://arxiv.org/abs/1602.08162`.
† This research was supported in part by the NSF under grants IIS-1251049, CNS-1302688.

Table 1 Our Results. All bounds expressed in bits, upto constant factors. For the matching results, $\rho = \min(n, C)$ where C is the cardinality of the optimal matching (weighted or unweighted). Note that for the MST, the verification is for a $(1+\epsilon)$-approximation. For the TSP, the verification is for a $(3/2+\varepsilon)$-approximation. (*) γ' is a linear function of γ and is strictly more than 1 as long as γ is a sufficiently large constant.

Problem	$\log n$ rounds		$\gamma = O(1)$ rounds	
	Verifier Space	Communication	Verifier Space	Communication
Triangle Counting	$\log^2 n$	$\log^2 n$	$\log n$	$n^{1/\gamma} \log n$
Matchings (all versions)	$\log^2 n$	$(\rho + \log n) \log n$	$\log n$	$(\rho + n^{1/\gamma'}) \log n$ (*)
Connectivity	$\log^2 n$	$n \log n$	$\log n$	$n \log n$
Minimum Spanning Tree	$\log^2 n$	$n \log^2 n/\varepsilon$	$\log n$	$n \log^2 n/\varepsilon$
Travelling Salesperson	$\log^2 n$	$n \log^2 n/\varepsilon$	$\log n$	$n \log^2 n/\varepsilon$

salesperson problem. In particular, we present protocols that verify a matching *exactly* in a graph using polylogarithmic space and polylogarithmic communication apart from the matching itself. In all our results, we consider the input in the *dynamic streaming model*, where graph edges are presented in arbitrary order in a stream and we allow both deletion and insertion of edges. All our protocols use either $\log n$ rounds of communication or (if the output size is sufficiently large or we are willing to tolerate superlogarithmic communication) constant rounds of communication.

To prove the above results, we also need SIPs for sub-problems like connectivity, minimum spanning tree and triangle counting. While it is possible to derive similar (and in some cases better) results for these subroutines using known techniques [30], we require explicit protocols that return structures that can be used in the computation pipeline for the TSP. Furthermore, our protocols for these problems are much simpler than what can be obtained by techniques in [30], which require some effort to obtain precise bounds on the size and depth of the circuits corresponding to more complicated parallel algorithms. We summarize our results in Table 1. Due to space constraints subproblems like triangle counting, connectivity, bipartiteness and MST are presented in the full version [1].

Significance of our Results

While the streaming model of computation has been extremely effective for processing numeric and matrix data, its ability to handle large graphs is limited, even in the so-called *semi-streaming* model where the streaming algorithm is permitted to use space quasilinear in the number of vertices. Recent breakthroughs in graph sketching [43] have led to space-efficient approximations for many problems in the semi-streaming model but canonical graph problems like matchings have been shown to be provably hard.

It is known [36] that no better than a $1 - 1/e$ approximation to the maximum cardinality matching is possible in the streaming model, even with space $\tilde{O}(n)$. It was also known that even allowing limited communication (effectively a single message from the prover) required a space-communication product of $\Omega(n^2)$ [16, 22]. Our results show that even allowing a few more rounds of communication dramatically improves the space-communication tradeoff for matching, as well as yielding *exact* verification. We note that streaming algorithms for matching vary greatly in performance and complexity depending in whether the graph is weighted or unweighted, bipartite or nonbipartite. In contrast, our results apply to all forms of matching. Interestingly, the special case of *perfect matching*, by virtue of being in RNC [37], admits an efficient SIP via results by Goldwasser, Kalai and Rothblum [30] and Cormode,

Thaler and Yi [23]. Similarly for triangle counting, the best streaming algorithm [5] yields an additive εn^3 error estimate in polylogarithmic space, and again in the annotation model (effectively a single round of communication) the best result yields a space-communication tradeoff of $n^2 \log^2 n$, which is almost exponentially worse than the bound we obtain. We note that counting triangles is a classic problem in the sublinear algorithms literature, and identifying optimal space and communication bounds for this problem was posed as an open problem by Graham Cormode in the Bertinoro sublinear algorithms workshop [21]. Our bound for verifying a $3/2+\epsilon$ approximation for the TSP in dynamic graphs is also interesting: a trivial 2-approximation in the semi-streaming model follows via the MST, but it is open to improve this bound (even on a grid) [46].

In general, our results can be viewed as providing further insight into the tradeoff between space and communication in sublinear algorithms. The annotation model of verification provides $\Omega(n^2)$ lower bounds on the space-communication product for the problems we consider: in that light, the fact that we can obtain polynomially better bounds with only constant number of rounds demonstrates the power of just a few rounds of interaction. We note that as of this paper, virtually all of the canonical hard problems for streaming algorithms (Index [17], Disjointness [9, 10], Boolean Hidden Matching [28, 15, 40]) admit efficient SIPs. A SIP for Index was presented in [17] and we present SIPs for Disjointness and Boolean Hidden Matching in the full version [1]. Our model is also different from a standard multi-pass streaming framework, since communication must remain sublinear in the input and in fact in all our protocols the verifier still reads the input exactly once.

From a technical perspective, our work continues the *sketching* paradigm for designing efficient graph algorithms. All our results proceed by building linear sketches of the input graph. The key difference is that our sketches are not approximate but algebraic: based on random evaluation of polynomials over finite fields. Our sketches use higher dimensional linearization ("tensorization") of the input, which might itself be of interest. They also compose: indeed, our solutions are based on building a number of simple primitives that we combine in different ways.

2 Related Work

Outsourced computation

Work on outsourced computation comes in three other flavors in addition to SIPs: firstly, there is work on reducing the verifier and prover complexity without necessarily making the verifier a sublinear algorithm [30, 29, 35], in some cases using cryptographic assumptions to achieve their bounds. Another approach is the idea of *rational proofs* [8, 19, 32, 31], in which the verifier uses a payment function to give the prover incentive to be honest. Moving to sublinear verifiers, there has been research on designing SIPs where the verifier runs in sublinear *time* [33, 45].

Streaming Graph Verification

All prior work on streaming graph verification has been in the annotation model, which in practice resembles a 1-round SIP (a single message from prover to verifier after the stream has been read). In recent work, Thaler [47] gives protocols for counting triangles, and computing maximum cardinality matching with both $n \log n$ space and communication cost. For matching, Chakrabarti *et al.* [16] show that any annotation protocol with space cost $O(n^{1-\delta})$ requires communication cost $\Omega(n^{1+\delta})$ for any $\delta > 0$. They also show that any

annotation protocol for graph connectivity with space cost $O(n^{1-\delta})$ requires communication cost $\Omega(n^{1+\delta})$ for any $\delta > 0$.

It is also proved that every protocol for this problem in the annotation model requires $\Omega(n^2)$ product of space and communication. This is optimal upto logarithmic factors. Furthermore, they conjecture that achieving smooth tradeoffs between space and communication cost is impossible, i.e. it is not known how to reduce the space usage to $o(n \log n)$ without blowing the communication cost up to $\Omega(n^2)$ or vice versa [16, 47]. Note that in all our protocols, the product of space and communication is $O(n \operatorname{poly} \log n)$.

Streaming Graph Algorithms

In the general dynamic streaming model, $\operatorname{poly} \log 1/\varepsilon$-pass streaming algorithms [2, 3] give $(1+\varepsilon)$-approximate answers and require $\tilde{O}(n)$ space in one pass. The best results for matching are [20] (a parametrized algorithm for computing a maximal matching of size k using $\tilde{O}(nk)$ space) and [7, 42] which gives a streaming algorithm for recovering an n^{ϵ}-approximate maximum matching by maintaining a linear sketch of size $\tilde{O}(n^{2-3\epsilon})$ bits. In the single-pass insert-only streaming model, Epstein et al. [27] give a constant (4.91) factor approximation for weighted graphs using $O(n \log n)$ space. Crouch and Stubbs [24] give a $(4+\epsilon)$-approximation algorithm which is the best known result for weighted matchings in this model. Triangle counting in streams has been studied extensively [11, 13, 14, 34, 44]. For dynamic graphs, the most space-efficient result is the one by [5] that provides the aforementioned additive εn^3 bound in polylogarithmic space. The recent breakthrough in sketch-based graph streaming [4] has yielded $\tilde{O}(n)$ semi-streaming algorithms [43] for computing the connectivity, bipartiteness and minimum spanning trees of dynamic graphs.

3 Preliminaries

We will work in the *streaming interactive proof* (SIP) model first proposed by Cormode et al. [23]. In this model, there are two players, the prover P and the verifier V. The input consists of a *stream* τ of items from a universe $\mathcal{U}$. Let f be a function mapping τ to any finite set $\mathcal{S}$. A k-message SIP for f works as follows:

1. V and P read the input stream and perform some computation on it.
2. V and P then exchange k messages, after which V either outputs a value in $\mathcal{S} \cup \{\perp\}$, where $\perp$ denotes that V is not convinced that the prover followed the prescribed protocol.

V is randomized. There must exist a prover strategy that causes the verifier to output $f(\tau)$ with probability $1 - \varepsilon_c$ for some $\varepsilon_c \leq 1/3$. Similarly, for all prover strategies, V must output a value in $\{f(\tau), \perp\}$ with probability $1 - \varepsilon_s$ for some $\varepsilon_s \leq 1/3$. The values ε_c and ε_s are respectively referred to as the completeness and soundness errors of the protocol. The protocols we design here will have perfect completeness ($\varepsilon_c = 0$).[1] We note that the annotated stream model of Chakrabarti et al. [16] essentially corresponds to one-message SIPs.[2]

[1] The constant 1/3 appearing in the completeness and soundness requirements is chosen by convention [6]. The constant 1/3 can be replaced with any other constant in $(0, 1)$ without affecting the theory in any way.

[2] Technically, the annotated data streaming model allows the annotation to be interleaved with the stream updates, while the SIP model does not allow the prover and verifier to communicate until after the stream has passed. However, almost all known annotated data streaming protocols do not utilize the ability to interleave the annotation with the stream, and hence are actually 1-message SIPs, but without any interaction from the verifier to prover side.

Input Model

We will assume the input is presented as stream updates to a vector. In general, each element of this stream is a tuple (i, δ), where each i lies in a universe $\mathcal{U}$ of size u, and $\delta \in \{+1, -1\}$. The data stream implicitly defines a frequency vector $\mathbf{a} = (a_1, \ldots, a_u)$, where a_i is the sum of all δ values associated with i in the stream. The stream update (i, δ) is thus the implicit update $\mathbf{a}[i] \leftarrow \mathbf{a}[i] + \delta$. In this paper, the stream consists of edges drawn from $\mathcal{U} = [n] \times [n]$ along with weight information as needed. As is standard, we assume that edge weights are drawn from $[n^c]$ for some constant c. We allow edges to be inserted and deleted but the final edge multiplicity is 0 or 1, and also mandate that the length of the stream is polynomial in n. Finally, for weighted graphs, we further constrain that the edge weight updates be atomic, i.e. that an edge along with its full weight be inserted or deleted at each step.

There are three parameters that control the complexity of our protocols: the vector length u, the length of stream s and the maximum size of a coordinate $M = max_i \mathbf{a}_i$. In the protocols discussed in this paper M will always be upper bounded by some polynomial in u, i.e. $\log M = O(\log u)$. All algorithms we present use linear sketches, and so the stream length s only affects verifier running time. In full version of the paper [1] we discuss how to reduce the verifier update time to polylogarithmic on each step.

Costs

A SIP has two costs: the verifier space, and the total communication, expressed as the number of bits exchanged between V and P. We will use the notation (A, B) to denote a SIP with verifier space $O(A)$ and total communication $O(B)$. We will also consider the number of rounds of communication between V and P. The basic versions of our protocols will require $\log n$ rounds, and we later show how to improve this to a constant number of rounds while maintaining the same space and similar communication cost otherwise.

4 Overview of our Techniques

For all the problems that we discuss the input is a data stream of edges of a graph where for an edge e an element in the stream is of the form (i, j, Δ). Now all our protocols proceed as follows. We define a domain $\mathcal{U}$ of size u and a frequency vector $\mathbf{a} \in \mathbb{Z}^u$ whose entries are indexed by elements of $\mathcal{U}$. A particular protocol might define a number of such vectors, each over a different domain. Each stream element will trigger a set of indices from $\mathcal{U}$ at which to update $\mathbf{a}$. For example in case of matching, we derive this constraint universe from the LP certificate, whereas for counting triangles our universe is derived from all $O(n^3)$ possible three-tuples of the vertices.

The key idea in all our protocols is that since we cannot maintain $\mathbf{a}$ explicitly due to limited space, we instead maintain a linear *sketch* of $\mathbf{a}$ that varies depending on the problem being solved. This sketch is computed as follows. We will design a polynomial that acts as a *low-degree* extension of f over an extension field $\mathbb{F}$ and can be written as $p(x_1, \ldots, x_d) = \sum_{u \in \mathcal{U}} a[u] g_u(x_1, x_2, \ldots, x_d)$. The crucial property of this polynomial is that it is *linear* in the entries of $\mathbf{a}$. This means that polynomial evaluation at any fixed point $\mathbf{r} = (r_1, r_2, \ldots, r_d)$ is easy in a stream: when we see an update $a[u] \leftarrow a[u] + \Delta$, we merely need to add the expression $\Delta g_u(\mathbf{r})$ to a running tally. Our sketch will always be a polynomial evaluation at a *random* point $\mathbf{r}$. Once the stream has passed, V and the prover P will engage in a conversation that might involve further sketches as well as further updates to the current sketch. In our descriptions, we will use the imprecise but convenient shorthand "increment

$\mathbf{a}[u]$" to mean "update a linear sketch of some low-degree extension of a function of $\mathbf{a}$". It should be clear in each context what the specific function is.

As mentioned earlier, a single stream update of the form (i, j, Δ) might trigger updates in many entries of $\mathbf{a}$, each of which will be indexed by a multidimensional vector. We will use the wild-card symbol '$*$' to indicate that all values of that coordinate in the index should be considered. For example, suppose $\mathcal{U} \subseteq [n] \times [n] \times [n]$. The instruction "update $\mathbf{a}[(i, *, j)]$" should be read as "update all entries $\mathbf{a}[t]$ where $t \in \{(i, s, j) \mid s \in [n], (i, s, j) \in \mathcal{U}\}$". We show later how to do these updates implicitly, so that verifier time remains suitably bounded.

5 Some Useful Protocols

We will make use of two basic tools in our algorithms: Reed-Solomon fingerprints for testing vector equality, and the streaming SumCheck protocol of Cormode et al. [23]. We summarize the main properties of these protocols here: for more details, the reader is referred to the original papers.

Multi-Set Equality (MSE)

We are given streaming updates to the entries of two vectors $\mathbf{a}, \mathbf{a}' \in \mathbb{Z}^u$ and wish to check $\mathbf{a} = \mathbf{a}'$. Reed-Solomon fingerprinting is a standard technique to solve MSE using only logarithmic space.

▶ **Theorem 1** (MSE, [22]). *Suppose we are given stream updates to two vectors $\mathbf{a}, \mathbf{a}' \in \mathbb{Z}^u$ guaranteed to satisfy $|\mathbf{a}_i|, |\mathbf{a}'_i| \leq M$ at the end of the data stream. Let $t = \max(M, u)$. There is a streaming algorithm using $O(\log t)$ space, satisfying the following properties: (i) If $\mathbf{a} = \mathbf{a}'$, then the streaming algorithm outputs 1 with probability 1. (ii) If $\mathbf{a} \neq \mathbf{a}'$, then the streaming algorithm outputs 0 with probability at least $1 - 1/t^2$.*

The SumCheck Protocol

We are given streaming updates to a vector $\mathbf{a} \in \mathbb{Z}^u$ and a univariate polynomial $h \colon \mathbb{Z} \to \mathbb{Z}$. The Sum Check problem (SumCheck) is to verify a claim that $\sum_i h(\mathbf{a}_i) = K$.

▶ **Lemma 2** (SumCheck, [23]). *There is a SIP to verify that $\sum_{i \in [u]} h(\mathbf{a}_i) = K$ for some claimed K. The total number of rounds is $O(\log u)$ and the cost of the protocol is $(\log(u) \log |\mathbb{F}|, deg(h) \log(u) \log |\mathbb{F}|)$.*

Here are the two other protocols that act as building blocks for our graph verification protocols.

Inverse Protocol (Finv)

Let $\mathbf{a} \in \mathbb{Z}^u$ be a (frequency) vector. The *inverse frequency* function F_k^{-1} for a fixed k is the number of elements of $\mathbf{a}$ that have frequency k: $F_k^{-1}(\mathbf{a}) = |\{i \mid \mathbf{a}_i = k\}|$. Let $h_k(i) = 1$ for $i = k$ and 0 otherwise. We can then define $F_k^{-1}(\mathbf{a}) = \sum_i h_k(\mathbf{a}_i)$. Note that the domain of h_k is $[M]$ where $M = \max_i \mathbf{a}_i$. We will refer to the problem of verifying a claimed value of F_k^{-1} as Finv. By using Lemma 2, there is a simple SIP for Finv. We restate the related results here [23].

▶ **Lemma 3** (Finv, [23]). *Given stream updates to a vector $\mathbf{a} \in \mathbb{Z}^u$ such that $\max_i \mathbf{a}_i = M$ and a fixed integer k there is a SIP to verify the claim $F_k^{-1}(\mathbf{a}) = K$ with cost $(\log^2 u, M \log^2 u)$ in $\log u$ rounds.*

Remark 1. Note that the same result holds if instead of verifying an inverse query for a single frequency k, we wish to verify it for a set of frequencies. Let $S \subset [M]$ and let $F_S^{-1} = |\{i|\mathbf{a}_i \in S\}|$. Then using the same idea as above, there is a SIP for verifying a claimed value of F_S^{-1} with costs given by Lemma 3.

Remark 2. Note that in the protocols presented in this paper later, the input to the FINV is not the graph edges itself, but instead the FINV is applied to the derived stream updates triggered by each input stream elements. As stated before, a single stream update of the form (i, j, Δ) might trigger updates in many entries of vector $\mathbf{a}$, which is defined based on the problem.

Subset Protocol

We now present a new protocol for a variant of the vector equality test described in Theorem 1. While this problem has been studied in the annotation model, it requires space-communication product of $\Omega(u^2)$ communication in that setting.

▶ **Lemma 4** (SUBSET)**.** *Let $E \subset [u]$ be a set of elements, and let $S \subset [u]$ be another set owned by P. There is a SIP to verify a claim that $S \subset E$ with cost $(\log^2 u, (|S| + \log u) \log u)$ in $\log u$ rounds.*

Proof. Consider a vector $\bar{\mathbf{a}}$ with length u, in which the verifier does the following updates: for each element in set E, increment the corresponding value in vector $\bar{\mathbf{a}}$ by $+1$ and for each element in set S, decrements the corresponding value in vector $\bar{\mathbf{a}}$ by -1. Let the vector $\mathbf{a} \in \{0, 1\}^u$ be the characteristic vector of E, and let $\mathbf{a}'$ be the characteristic vector of S. Thus, $\bar{\mathbf{a}} = \mathbf{a} - \mathbf{a}'$. By applying F_{-1}^{-1} protocol on $\bar{\mathbf{a}}$, verifier can determine if $S \subset E$ or not. Note that in vector $\bar{\mathbf{a}}$, $M = 1$. Then the protocol cost follows by Lemma 3. ◀

6 SIP for MAX-MATCHING in Bipartite Graphs

We now present a SIP for maximum cardinality matching in bipartite graphs. The prover P needs to generate two certificates: an actual matching, and a proof that this is optimal. By König's theorem [41], a bipartite graph has a maximum matching of size k if and only if it has a minimum vertex cover of size k. Therefore, P's proof consists of two parts:

(a) Send the claimed optimal matching $M \subset E$ of size k.
(b) Send a vertex cover $S \subset V$of size k.

V has three tasks:

(i) Verify that M is a matching and that $M \subset E$.
(ii) Verify that S covers all edges in E.
(iii) Verify that $|M| = |S|$.

We describe protocols for first two tasks and the third task is trivially solvable by counting the length of the streams and can be done in $\log n$ space. V will run the three protocols in parallel.

Verifying a Matching

Verifying that $M \subset E$ can be done by running the SUBSET protocol from Lemma 4 on E and the claimed matching M. A set of edges M is a matching if each vertex has degree at most 1 on the subgraph defined by M. Interpreted another way, let τ_M be the stream of endpoints of edges in M. Then each item in τ_M must have frequency 1. This motivates the

following protocol, based on Theorem 1. V treats τ_M as a sequence of updates to a frequency vector $\mathbf{a} \in \mathbb{Z}^{|V|}$ counting the number of occurrences of each vertex. V then asks P to send a stream of all the vertices incident on edges of M as updates to a different frequency vector $\mathbf{a}'$. V then runs the MSE protocol to verify that these are the same.

Verifying that S is a Vertex Cover

The difficulty with verifying a vertex cover is that V no longer has streaming access to E. However, we can once again reformulate the verification in terms of frequency vectors. S is a vertex cover if and only if each edge of E is incident to some vertex in S. Let $\mathbf{a}, \mathbf{a}' \in \mathbb{Z}^{\binom{n}{2}}$ be vectors indexed by $\mathcal{U} = \{(i,j), i,j \in V, i < j\}$. On receiving the input stream edge $e = (i, j, \Delta), i < j$, V increments $\mathbf{a}[(i,j)]$ by Δ.

For each vertex $i \in S$ that P sends, we increment all entries $\mathbf{a}'[(i,*)]$ and $\mathbf{a}'[(*,i)]$. Now it is easy to see that S is a vertex cover if and only there are no entries in $\mathbf{a} - \mathbf{a}'$ with value 1 (because these entries correspond to edges that have *not* been covered by a vertex in S). This yields the following verification protocol.

1. V processes the input edge stream for the F_1^{-1} protocol, maintaining updates to a vector $\mathbf{a}$.
2. P sends over a claimed vertex cover S of size c^* one vertex at a time. For each vertex $i \in S$, V *decrements* all entries $\mathbf{a}[(i,*)]$ and $\mathbf{a}[(*,i)]$.
3. V runs Finv to verify that $F_1^{-1}(\mathbf{a}) = 0$.

The bounds for this protocol follow from Lemmas 3, 4 and Theorem 1.

▶ **Theorem 5.** *Given an input bipartite graph with n vertices, there exists a streaming interactive protocol for verifying the maximum-matching with $\log n$ rounds of communication, and cost $(\log^2 n, (c^* + \log n) \log n)$, where c^* is the size of the optimal matching.*

7 SIP for Maximum-Weight-Matching in General Graphs

We now turn to the most general setting: of maximum weight matching in general graphs and the bipartite case is moved to the full version [1]. This of course subsumes the easier case of maximum cardinality matching in general graphs, and while there is a slightly simpler protocol for that problem based on the Tutte-Berge characterization of maximum cardinality matchings [48, 12], we will not discuss it here.

We will use the odd-set based LP-duality characterization of maximum weight matchings due to Cunningham and Marsh. Let $\mathbb{O}(V)$ denote the set of all odd-cardinality subsets of V. Let $y_i \in [n^c]$ define non-negative integral weight on vertex v_i, $z_U \in [n^c]$ define a non-negative integral weight on an *odd-cardinality* subset $U \in \mathbb{O}(V)$, $w_{ij} \in [n^c]$ define the weight of an edge $e = (i,j)$ and $c^* \in [n^{c+1}]$ be the weight of a maximum weight matching on G. We define y and z to be *dual feasible* if $y_i + y_j + \sum_{\substack{U \in \mathbb{O}(V) \\ i,j \in U}} z_U \geq w_{i,j}, \forall i,j$.

A collection of sets is said to be *laminar*, if any two sets in the collection are either disjoint or nested (one is contained in the other). Note that such a family must have size linear in the size of the ground set. Standard LP-duality and the Cunningham-Marsh theorem state that:

▶ **Theorem 6** ([25])**.** *For every integral set of edge weights W, and choices of dual feasible integral vectors y and z, $c^* \leq \sum_{v \in V} y_v + \sum_{U \in \mathbb{O}(V)} z_U \left\lfloor \frac{1}{2}|U| \right\rfloor$. Furthermore, there exist vectors y and z that are dual feasible such that $\{U : z_U > 0\}$ is laminar and for which the above upper bound achieves equality.*

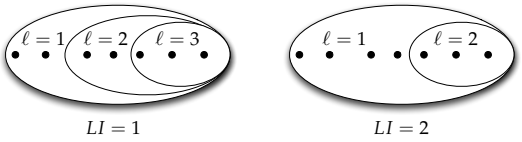

Figure 1 A Laminar family.

We design a protocol that will verify that each dual edge constraint is satisfied by the dual variables. The laminar family $\{U : z_U > 0\}$ can be viewed as a collection of nested subsets (each of which we call a *claw*) that are disjoint from each other. Within each claw, a set U can be described by giving each vertex v in order of increasing *level* $\ell(v)$: the number of sets v is contained in (see Figure 1).

The prover will describe a set U and its associated z_U by the tuple $(LI, \ell, r_U, \partial U)$, where $1 \leq LI \leq n$ is the index of the claw U is contained in, $\ell = \ell(U)$, $r_U = \sum_{U' \supseteq U'} z_{U'}$ and $\partial U = U \setminus \cup_{U'' \subset U} U''$. For an edge $e = (i, j)$ let $r_e = \sum_{i,j \in U, U \in \mathbb{O}(V)} z_U$ represent the weight assigned to an edge by weight vector z on the laminar family. Any edge whose endpoints lie in different claws will have $r_e = 0$. For a vertex v, let $r_v = \min_{v \in U} r_U$. For an edge $e = (v, w)$ whose endpoints lie in the same claw, it is easy to see that $r_e = \min(r_v, r_w)$, or equivalently that $r_e = r_{\arg\min(\ell(v),\ell(w))}$. For such an edge, let $\ell_{e,\downarrow} = \min(\ell(u), \ell(v))$ and $\ell_{e,\uparrow} = \max(\ell(u), \ell(v))$. We will use $LI(e) \in [n]$ to denote the index of the claw that the endpoints of e belong to.

The Protocol

V prepares to make updates to a vector $\mathbf{a}$ with entries indexed by $\mathcal{U} = \mathcal{U}_1 \cup \mathcal{U}_2$. $\mathcal{U}_1$ consists of all tuples of the form $\{(i, j, w, y, y', LI, \ell, \ell', r)\}$ and $\mathcal{U}_2$ consists of all tuples of the form $\{(i, j, w, y, y', 0, 0, 0, 0)\}$ where $i < j, i, j, LI, \ell, \ell' \in [n]$, $y, y', r, w \in [n^c]$ and tuples in $\mathcal{U}_1$ must satisfy 1) $w \leq y + y' + r$ and 2) it is *not* simultaneously true that $y + y' \geq w$ and $r > 0$. Note that $\mathbf{a} \in \mathbb{Z}^u$ where $u = O(n^{4c+5})$ and all weights are bounded by n^c.

1. V prepares to process the stream for an F_5^{-1} query. When V sees an edge update of form (e, w_e, Δ), it updates all entries $\mathbf{a}[(e, w_e, *, *, *, *, *, *)]$.
2. P sends a list of vertices (i, y_i) in order of increasing i. For each (i, y_i), V increments by 1 the count of all entries $\mathbf{a}[(i, *, *, y_i, *, *, *, *, *]$ and $\mathbf{a}[(*, i, *, *, y_i, *, *, *, *)]$ with indices drawn from $\mathcal{U}_1$. Note that P only sends vertices with nonzero weight, but since they are sent in increasing order, V can infer the missing entries and issue updates to $\mathbf{a}$ as above. V also maintains the sum of all y_i.
3. P sends the description of the laminar family in the form of tuples $(LI, \ell, r_U, \partial U)$, sorted in lexicographic order by LI and then by ℓ. V performs the following operations.
   a. V increments all entries of the form $(i, *, *, y_i, *, 0, 0, 0, 0)$ or $(*, i, *, *, y_i, 0, 0, 0, 0)$ by 2 to account for edges which are satisfied by only vector y.
   b. V maintains the sum Σ_R of all r_U seen thus far. If the tuple is deepest level for a given claw (easily verified by retaining a one-tuple lookahead) then V adds r_U to a running sum $\Sigma_{\max}$.
   c. V verifies that the entries appear in sorted order and that r_U is monotone increasing.
   d. V updates the fingerprint structure from Theorem 1 with each vertex in ∂U.
   e. For each $v \in \partial U$, V increments (subject to our two constraints on the universe) all entries of $\mathbf{a}$ indexed by tuples of the form $(e, w_e, *, *, LI, *, \ell, *)$ and all entries indexed by tuples of the form $(e, w_e, *, *, LI, \ell, *, r_U)$, where e is any edge containing v as an endpoint.

f. V ensures all sets presented are odd by verifying that for each LI, all $|\partial U|$ except the last one are even.

4. P sends V all vertices participating in the laminar family in ascending order of vertex label. V verifies that the fingerprint constructed from this stream matches the fingerprint constructed earlier, and hence that all the claws are disjoint.
5. V runs a verification protocol for $F_5^{-1}(\mathbf{a})$ and accepts if $F_5^{-1}(\mathbf{a}) = m$, returning Σ_r and $\Sigma_{\max}$.

Define c^s as the certificate size, which is upper bounded by the matching cardinality. Then:

▶ **Theorem 7.** *Given dynamic updates to a weighted graph on n vertices with all weights bounded polynomially in n, there is a SIP with cost $(\log^2 n, (c^s + \log n)\log n)$, where c^s is the cardinality of maximum matching, that runs in $\log n$ rounds and verifies the size of a maximum weight matching.*

Proof. In parallel, V and P run protocols to verify a claimed matching as well as its optimality. The correctness and resource bounds for verifying the matching follow from Section 6. We now turn to verifying the optimality of this matching. The verifier must establish the following facts:

(i) P provides a valid laminar family of odd sets.
(ii) The lower and upper bounds are equal.
(iii) All dual constraints are satisfied.

Since the verifier fingerprints the vertices in each claw and then asks P to replay all vertices that participate in the laminar structure, it can verify that no vertex is repeated and therefore that the family is indeed laminar. Each ∂U in a claw can be written as the difference of two odd sets, except the deepest one (for which $\partial U = U$. Therefore, the cardinality of each ∂U must be even, except for the deepest one. V verifies this claim, establishing that the laminar family comprises odd sets.

Consider the term $\sum_U z_U \lfloor |U|/2 \rfloor$ in the dual cost. Since each U is odd, this can be rewritten as $(1/2)(\sum_u z_u|U| - \sum_U z_U)$. Consider the odd sets $U_0 \supset U_1 \supset \ldots \supset U_l$ in a single claw. We have $r_{U_j} = \sum_{i \le j} z_{U_i}$, and therefore $\sum_j r_{U_j} = \sum_j \sum_{i \le j} z_{U_i}$. Reordering, this is equal to $\sum_{i \le j} \sum_j z_{U_i} = \sum_i z_{U_i}|U_i|$. Also, $r_{U_l} = \sum_i z_{U_i}$. Summing over all claws, $\Sigma_r = \sum_U z_U|U|$ and $\Sigma_{\max} = \sum_U z_U$. Therefore, $\sum_i y_i + \Sigma_r - \Sigma_{\max}$ equals the cost of the dual solution provided by P.

Finally we turn to validating the dual constraints. Consider an edge $e = (i, j)$ whose dual constraints are satisfied: i.e. P provides y_i, y_j and z_U such that $y_i + y_j + r_{ij} \ge w_e$. Firstly, consider the case when $r_{ij} > 0$. In this case, the edge belongs to some claw LI. Let its lower and upper endpoints vertex levels be s, t, corresponding to odd sets U_S, U_t. Consider now the entry of $\mathbf{a}$ indexed by $(e, y_i, y_j, LI, s, t, r_{ij})$. This entry is updated when e is initially encountered and ends up with a net count of 1 at the end of input processing. It is incremented twice when P sends the (i, y_i) and (j, y_j). When P sends U_s this entry is incremented because $r_{ij} = r_{U_s} = \min(r_{U_S}, r_{U_t})$ and when P sends U_t this entry is incremented because U_t has level t, returning a final count of 5. If $r_{ij} = 0$ (for example when the edge crosses a claw), then the entry indexed by $(e, w_e, y_i, y_j, 0, 0, 0, 0)$ is incremented when e is read. It is not updated when P sends (i, y_i) or (j, y_j). When P sends the laminar family, V increments this entry by 2 twice (one for each of i and j) because we know that $y_i + y_j \ge w_e$. In this case, the entry indexed by $(i, j, w_e, y_i, y_j, 0, 0, 0, 0)$ will be exactly 5. Thus, for each satisfied edge there is exactly one entry of $\mathbf{a}$ that has a count of 5.

Conversely, suppose e is not satisfied by the dual constraints, for which a necessary condition is that $y_i + y_j < w_e$. Firstly, note that any entry indexed by $(i, j, w_e, *, *, 0, 0, 0, 0)$ will receive only two increments: one from reading the edge, and another from one of y_i and y_j but not both. Secondly, consider any entry with an index of the form $(i, j, w_e, *, *, LI, *, *, *)$ for $LI > 0$. Each such entry gets a single increment from reading e and two increments when P sends (i, y_i) and (j, y_j). However, it will not receive an increment from the second of the two updates in Step 3(e), because $y_i + y_j + r_{ij} < w_e$ and so its final count will be at most 4. The complexity of the protocol follows from the complexity for FINV, SUBSET and the matching verification described in Section 6. ◀

8 Streaming Interactive Proofs for Approximate Metric TSP

We can apply our protocols to another interesting graph streaming problem: that of computing an approximation to the min cost travelling salesman tour. The input here is a weighted complete graph of distances. We briefly recall the Christofides heuristic: compute a MST T on the graph and add to T all edges of a min-weight perfect matching on the odd-degree vertices of T. The classical Christofides result shows that the sum of the costs of this MST and induced min-weight matching is a 3/2 approximation to the TSP cost. In the SIP setting, we have protocols for both of these problems. The difficulty however is in the dependency: the matching is built on the odd-degree vertices of the MST, and this would seem to require the verifier to maintain much more states as in the streaming setting. We show that this is not the case, and in fact we can obtain an efficient SIP for verifying a $(3/2+\epsilon)$-approximation to the TSP. To summarize our SIP for verifying an approximate MST, here first we state two main results which are used as subroutine: Connectivity and Bipartiteness.

For establishing the number of connected components in graph G, we devise SIPs to verify spanning trees, as well as the disjointness and maximality of any claimed connected components by prover. We show the results here and move full details of the protocols to the full version [1].

▶ **Lemma 8.** *Given an input graph G with n vertices, there exists a SIP protocol for verifying the number of connected components G_i with $(\log n)$ rounds of communication, and $(\log^2 n, n \log n)$ cost.*

By applying the verification protocol for connectivity on both the input graph G and the *bipartite double cover* [4] of G we obtain the following results for testing bipartiteness:

▶ **Lemma 9.** *Given an input graph G with n vertices, there exists a SIP protocol for testing bipartiteness on G with $(\log n)$ rounds of communication, and $(\log^2 n, n \log n)$ cost.*

Now for MST protocol, we follow a reduction for stratifying graph edges by weight and counting the number of connected components at each level introduced in [18] and later generalized to streaming setting [4]. This finally yields us:

▶ **Theorem 10.** *Given a weighted graph with n vertices, there is a SIP protocol for verifying MST within $(1+\epsilon)$-approximation with $(\log n)$ rounds of communication, and $(\log^2 n, n \log^2 n/\epsilon)$ cost.*

We note here that while we could have used known parallel algorithms for connectivity and MST combined with the protocol of Goldwasser et al. [30] and the technique of Cormode, Thaler and Yi [23] to obtain similar results, we need an explicit and simpler protocol with an output that we can fit into the overall TSP protocol.

What remains is how we verify a min-cost perfect matching on the odd-degree nodes of the spanning tree. We employ the procedure described in Section 7 for maximum weight matching along with a standard equivalence to min-cost perfect matching. In addition to validating all the LP constraints, we also have to make sure that they pertain solely to vertices in ODD. We do this as above by using the fingerprint for ODD to ensure that we only count satisfied constraints on edges in ODD. We present the details of TSP in the full version of the paper [1]. Finally, the approximate TSP cost is the sum of the min-weight perfect matching on ODD and the MST cost on the graph.

▶ **Theorem 11.** *Given a weighted complete graph with n vertices, in which the edge weights satisfy the triangle inequality, there exists a streaming interactive protocol for verifying optimal TSP cost within $(\frac{3}{2}+\epsilon)$-approximation with $(\log n)$ rounds of communication, and $(\log^2 n, n\log^2 n/\varepsilon)$ cost.*

References

1 A. Abdullah, S. Daruki, C. D. Roy, and S. Venkatasubramanian. Streaming verification of graph properties. *CoRR*, abs/1602.08162, 2016. URL: `http://arxiv.org/abs/1602.08162`.

2 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *arXiv preprint arXiv:1307.4359*, 2013.

3 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Information and Computation*, 222:59–79, 2013.

4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *SODA*, pages 459–467. SIAM, 2012.

5 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st Symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.

6 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

7 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Tight bounds for linear sketches of approximate matchings. *arXiv preprint arXiv:1505.01467*, 2015.

8 Pablo Daniel Azar and Silvio Micali. Rational proofs. In *STOC*, pages 1017–1028. ACM, 2012.

9 Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.

10 Ziv Bar-Yossef. *The complexity of massive data set computations*. PhD thesis, University of California at Berkeley, 2002.

11 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *SODA*, pages 623–632. Society for Industrial and Applied Mathematics, 2002.

12 C. Berge. Sur le couplage maximum d'un graphe. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 247:258–259, 1958.

13 Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. How hard is counting triangles in the streaming model? In *Automata, Languages, and Programming*, pages 244–254. Springer, 2013.

14 Luciana S Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. Counting triangles in data streams. In *Proceedings of the 25th ACM*

SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pages 253–262. ACM, 2006.

15 Marc Bury and Chris Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. *arXiv preprint arXiv:1505.02019*, 2015.

16 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Annotations in data streams. In *Automata, Languages and Programming*, pages 222–234. Springer, 2009.

17 Amit Chakrabarti, Graham Cormode, Andrew McGregor, Justin Thaler, and Suresh Venkatasubramanian. Verifiable stream computation and arthur–merlin communication. In *CCC*, 2015.

18 Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005.

19 Jing Chen, Samuel McCauley, and Shikha Singh. Rational proofs with multiple provers. *CoRR*, abs/1504.08361, 2015. URL: `http://arxiv.org/abs/1504.08361`.

20 Rajesh Chitnis, Graham Cormode, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. Parameterized streaming: maximal matching and vertex cover. In *SODA*, pages 1234–1251. SIAM, 2015.

21 Graham Cormode. Bertinoro workshop 2011, problem 47. URL: `http://sublinear.info/index.php?title=Open_Problems:47`.

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

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

24 Michael Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. *APPROX/RANDOM*, 28:96–104, 2014.

25 W. H. Cunningham and A. B. Marsh. A primal algorithm for optimum matching. In *Polyhedral Combinatorics*, pages 50–72. Springer, 1978.

26 Samira Daruki, Justin Thaler, and Suresh Venkatasubramanian. Streaming verification in data analysis. In *Algorithms and Computation*, pages 715–726. Springer Berlin Heidelberg, 2015.

27 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011.

28 Hossein Esfandiari, Mohammad T Hajiaghayi, Vahid Liaghat, Morteza Monemizadeh, and Krzysztof Onak. Streaming algorithms for estimating the matching size in planar graphs and beyond. In *SODA*, pages 1217–1233. SIAM, 2015.

29 Stephen A. Fenner, Rohit Gurjar, and Thomas Thierauf. Bipartite perfect matching is in quasi-nc. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:177, 2015. URL: `http://eccc.hpi-web.de/report/2015/177`.

30 Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC'08*, pages 113–122, New York, NY, USA, 2008. ACM.

31 Siyao Guo, Pavel Hubácek, Alon Rosen, and Margarita Vald. Rational arguments: single round delegation with sublinear verification. In *Proc. ITCS*, pages 523–540. ACM, 2014.

32 Siyao Guo, Pavel Hubácek, Alon Rosen, and Margarita Vald. Rational sumchecks. In *Theory of Cryptography*, pages 319–351. Springer, 2016.

33 Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In Tim Roughgarden, editor, *Proc. ITCS*, pages 133–142. ACM, 2015.

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

35 Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: The power of no-signaling proofs. Cryptology ePrint Archive, Report 2013/862, 2013. URL: `http://eprint.iacr.org/`.

36 Michael Kapralov. Better bounds for matchings in the streaming model. In *SODA*, pages 1679–1697. SIAM, 2013.

37 R. M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, January 1986.

38 Hartmut Klauck. On arthur merlin games in communication complexity. In *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*, pages 189–199. IEEE, 2011.

39 Hartmut Klauck and Ved Prakash. An improved interactive streaming algorithm for the distinct elements problem. In *Automata, Languages, and Programming*, pages 919–930. Springer, 2014.

40 Dmitry Kogan and Robert Krauthgamer. Sketching cuts in graphs and hypergraphs. In *Proc. ITCS*, pages 367–376. ACM, 2015.

41 D. König. Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére. *Matematikai és Természettudományi Értesít***o**, 34:104–119, 1916.

42 Christian Konrad. Maximum matching in turnstile streams. *arXiv preprint arXiv:1505.01460*, 2015.

43 Andrew McGregor. Graph stream algorithms: A survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

44 Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment*, 6(14):1870–1881, 2013.

45 Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proc. STOC*, 2016.

46 Christian Sohler. Dortmund workshop on streaming algorithms, problem 52. 2012. URL: `http://sublinear.info/index.php?title=Open_Problems:52`.

47 Justin Thaler. Semi-streaming algorithms for annotated graph streams. In *Proc. ICALP*, 2016.

48 W. T. Tutte. The factorization of linear graphs. *The Journal of the London Mathematical Society, Ser. 1*, 22(2):107–111, 1947.