Computing Flows in Subquadratic Space

Jan van den Brand
Georgia Institute of Technology, Atlanta, GA, USA

Zhao Song
Independent, Seattle, WA, USA

Albert Weng
Georgia Institute of Technology, Atlanta, GA, USA

Abstract

Space complexity is a critical factor in various computational models, including streaming, parallel/distributed computing, and communication complexity. We study the space complexity of the minimum-cost flow problem, a generalization of the st-max flow problem, focusing on computing flows in subquadratic space. In the general case with arbitrary capacities, minimum cost and *st*-maximum flows can use up to $\Omega(n^2)$ edges, so computing the flow on each edge (rather than just the size/cost) seems impossible in subquadratic space. Indeed, there are lower bounds proving quadratic space is needed to store the flow on every edge, which has been used to prove lower bounds on streaming algorithms. However, we show that these lower bounds can be circumvented, opening up improvements for streaming and communication complexity.

For a directed graph with integer capacities and costs bounded by W, we provide a $\widetilde{O}(n^{1.5}\log(W/\epsilon))$-space $\widetilde{O}(\sqrt{n}\log(W/\epsilon))$-pass streaming algorithm, which during the last pass returns the flow on each edge up to an additive error of ϵ. Crucially, the algorithm does not return the flow at the end of the last pass but returns the flow on an edge, as the edge is read in the stream. This allows us to circumvent existing $\Omega(n^2)$ space lower bounds. In the 2-party communication model, our algorithm implies $\widetilde{O}(n^{1.5}\log^2 W)$ bits of communication.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Combinatorial Optimization, Continuous Optimization, Graph Algorithms, Streaming and Sketching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.46

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2605.09547

Funding *Jan van den Brand*: NSF Award CCF-2338816 and CCF-2504994.
Albert Weng: NSF Award CCF-2338816.

1 Introduction

Space complexity is a important factor in streaming, parallel/distributed computing, communication complexity, and data structures, requiring computation and representation of objects using limited space. In this work, we study the space complexity of the minimum-cost flow problem, a generalization of the *st*-max flow problem. We focus on computing such flows in subquadratic space in the multi-pass streaming model, contrasting quadratic space lower bounds from [22, 6, 8]. In particular, our goal is to compute the actual flow on each edge, rather than just the cost of the min-cost flow or the size of the maximum flow.

In the minimum-cost flow problem, we are given a directed n-node graph $G = (V, E)$, a vertex-demand vector $d \in \mathbb{Z}^V$, edge capacities $u \in \mathbb{Z}^E_{\geq 0}$ and edge costs $c \in \mathbb{Z}^E$. The minimum-cost flow is a flow $f \in \mathbb{R}^E_{\geq 0}$ that satisfies the vertex-demands and edge capacities, while minimizing the cost $\sum_{e\in E} c_e \cdot f_e$. While the min-cut $W \subset V$ (and thus size of a max


flow) can be stored in $O(n)$ space, observe that flows in general can use $\Omega(n^2)$ edges. Thus it is not clear if one can compute the flow $f \in \mathbb{R}^E_{\geq 0}$ (i.e., the amount of flow on each edge) in subquadratic space.

Streaming Algorithms. In the streaming model, the input (i.e., edge set) is given to the algorithm one-by-one as a stream of data. The algorithm is only allowed a small amount of space and thus cannot store the entire input. In a multi-pass algorithm, one may take multiple passes over the input stream. This computational model is motivated by large amounts of data being faster to read sequentially than via random access, e.g., when the data is too large for working memory and is read directly from hard-drive. Bipartite matching (a simpler special case of max flow) is one of the most successfully studied problems in streaming literature, both in the approximate (see, e.g.,[32, 2, 31, 38, 48, 43, 3, 15, 9, 12, 5] and references therein) and exact setting [51, 9]. However, for flows much less is known. For undirected maximum flows, one can $(1+\epsilon)$-approximate the size of the max flow in $\widetilde{O}(n)$ space via duality by constructing a cut sparsifier (see, e.g., [1, 4, 49, 20, 22]). [58] show one can compute the exact st-min cut (i.e., size of max flow) on unweighted undirected graphs in $\widetilde{O}(n^{5/3})$ space. Computing the exact flow size/cost on weighted graphs in subquadratic space remains open.

When it comes to computing not just the size/cost of the flow but the flow $f \in \mathbb{R}^E_{\geq 0}$ itself, existing upper bounds are substantially weaker. While sparsification approaches allow a $(1+\epsilon)$-approximation of the size, only crude $\text{poly}(n)$ approximations are known for computing flow f in subquadratic space [22]. Indeed, this gap between size and flow f can be explained and [22] provide respective lower bounds.

For intuition, why representing (i.e., not just computing) flows in subquadratic space is impossible, consider a complete bipartite graph with edges oriented left to right, an extra vertex s connected to the left vertices, and an extra vertex t reachable from all right vertices. The edges from s and edges to t have very high capacity, so the max flow has $f_e = u_e$ for each edge e of the bipartite graph. Thus storing/representing all (or even most, in case of approximation) f_e in subquadratic space means we can store all (or most) capacities u_e in subquadratic space, but that is $\Theta(n^2)$ information ($u \in \mathbb{R}^E$ could be any n^2-sized binary bit string). This lower bound argument was formalized by [22][1], who proved that for every encoding algorithm $\mathcal{E}(G) \in \{0,1\}^d$ (computing a max-flow on G and then encoding it in d bit) and decoding algorithm $\mathcal{A}$ (where $\mathcal{A}(\mathcal{E}(G))$ returns a $(1-1/36)$-approximation of the max-flow) needs $d = \Omega(n^2)$. Thus it's impossible to encode max/min-cost flows f in subquadratic space. Importantly, the lower bound by [22] applies not just to streaming but to any encoding scheme. Thus the argument can also be used to argue about the communication complexity for communicating f.

Communication Complexity. In the 2-party (or multi-party) communication model, the edges of the input graph are distributed among multiple parties. The task is to solve some graph problem while using as little communication as possible. This area, too, had a lot of success for various graph problems [41, 17, 34, 29, 57, 14, 40, 30, 54], but flows remain elusive. For related problems like bipartite matching or transshipment, an $\widetilde{O}(n)$ upper bound is known [16, 41], crucially relying on the fact that the optimal solution consists of at most $O(n)$ edges. For unweighted undirected st-min cut, a $\widetilde{O}(n^{5/3})$ communication protocol is known [58], later improved to $\widetilde{O}(n^{11/7})$ [42]. We remark that on unweighted

[1] [22] stated this as ℓ_∞-regression, but the statement is equivalent to max-flow. Details in the full version.

graphs the max-flow can use at most $O(n^{1.5})$ edges[2], so the problem is substantially easier in the unweighted setting. A subquadratic communication algorithm for weighted flows (but also the simpler weighted *st*-min-cut problem where solutions have size $O(n)$) was stated as open problem in [16].

When it comes to flow f, rather than its cost/size, note that not just the *computation* is difficult, but just *communication* of the solution is already a challenge. Namely, assume one party already knows all the edges and the flow. How would that party communicate the flow to the other party? This is clearly a simpler problem than jointly computing the flow since the 1st party could just throw away the extra information. However, how would one communicate the flow if no low space representation of the flow is possible? In particular, the communicated messages would be an encoding of the flow.

Computing Flow In Subquadratic Space. We show that computation of the min-cost/max flows in subquadratic space is possible. In addition to the exact flow cost/size, our algorithm also returns the flow f_e on each edge $e \in E$. This is unexpected, as it appears to contradict the lower bound of [22]. We can circumvent the lower bound, because it assumes a decoding algorithm of form $\mathcal{D}(M)$ for encoding $M \in \{0,1\}^*$ of the flow, i.e., decoding must be performed without access to the graph. The lower bound does not rule out a decoding algorithm of form $\mathcal{D}(M, e)$ that receives edge e as part of the input in addition to some subquadratic space encoding $M \in \{0,1\}^*$ of the flow. The assumption that we have access to the edge during decoding is naturally satisfied in many low space models. In the communication model, each party knows their own (possibly $\Omega(n^2)$ sized) set of edges. For multi-pass streaming algorithms, the edges are available via another pass over the input. In particular, during the last pass over the edges, the algorithm returns the flow on each edge when it is read in the stream. This approach allows us to compute flows in subquadratic space, even though the flow uses up to $\Omega(n^2)$ many edges.

Our main technical contribution is such an encoding/decoding scheme adapted to the framework of "robust interior point methods" for linear programming. In the context of streaming algorithms, the mere existence of such encoding does not suffice, but we show that the computation of this encoding can be done in low space, too. Using the encoding with the robust interior point method of [19] (which solves linear programs/min-cost flow in $\widetilde{O}(\sqrt{n})$ iterations), we present a $\widetilde{O}(\sqrt{n})$-pass, $\widetilde{O}(n^{1.5})$-space streaming algorithm, which can be implemented with $\widetilde{O}(n^{1.5})$ communication in the 2-party model.

1.1 Our Results

We present the following result for computing min-cost flows in the multi-pass streaming model.

▶ **Theorem 1** (Min-cost Flow, Streaming). *There is a randomized multi-pass streaming algorithm that, given an input graph $G = (V, E)$ with integer capacities $u \in [1, W]^E$ and costs $c \in [1, W]^E$, vertex demand vector $b \in \mathbb{R}^V$ and accuracy parameter $\epsilon \in [0, 1]$, computes w.h.p. a min-cost flow in $\widetilde{O}(n^{1.5} \log(W/\epsilon))$ space and $\widetilde{O}(\sqrt{n} \log(W/\epsilon))$ passes over the input. The total time is $\widetilde{O}(mn \log^2(W/\epsilon))$.*

[2] See full version for details.

The algorithm returns a randomized $\widetilde{O}(n^{1.5}\log(W/\epsilon))$-space data structure that supports edge-queries: For any given edge $e = ((w, v), c_e, u_e)$, the query returns in $\widetilde{O}(\sqrt{n}\log(W/\epsilon))$ time an estimate $\overline{f}_e$ of the flow on that edge, with $0 \leq \overline{f}_e \leq u_e$ and $|\overline{f}_e - f_e| \leq \epsilon$ with high probability.

Alternatively, we can also assume that during the last pass, the algorithm returns for each given edge $e = ((w, v), c_e, u_e)$ the flow $\overline{f}_e$ on that edge.

While the streaming area often focuses on small number of passes (e.g., constant or polylog), it was proven that polynomial number of passes may be necessary for flows in the weighted case. In the presence of capacities as large as sub-exponential in n, [6] showed that a streaming algorithm for computing the size of the max-flow in p passes needs $\Omega(n^2/p^5)$ space. This lower bound was later improved to $\Omega(n^2/p^3)$ space [8]. For smaller polynomially-sized capacities, Theorem 1 is the first upper bound for computing the exact size of the max-flow in the weighted case (when interested in the size, we can simply round the result to nearest integer). It is also the first result to return the flow $f \in \mathbb{R}^E$ with high accuracy rather than the cut or cost/size of the flow.

Previous work was either highly inaccurate with polynomial error when returning the flow [22], or when exact, previous work was only for the unweighted undirected case [58] and returned only the cut but not the flow. Observe that for unweighted graphs, any max flow uses at most $O(n^{1.5})$ edges (see full version for details), but for weighted graphs the number of used edges can be as large as $O(n^2)$. So minimizing space complexity is generally harder for the weighted case.

Regarding Accuracy. If we care only about the cost of the flow (or size of max-flow), then the algorithm's output is exact by choosing $\epsilon \leq 1/3$ and then rounding the computed cost to the nearest integer.

If the minimum-cost flow is unique, then rounding each f_e to the nearest integer also results in the exact flow on each edge. If the flow is not unique, then the algorithm might compute a fractional flow and thus rounding is not guaranteed to provide an exact solution for the edges. However, given the logarithmic complexity dependence on ϵ, we can reduce the error to an arbitrary small $\epsilon = 1/\operatorname{poly}(n, W)$ on each edge. Thus we get an almost exact fractional min-cost flow $f \in \mathbb{R}^E_{\geq 0}$.

Usually, uniqueness can be guaranteed via Isolation Lemma [24, 28], which adds small random perturbation to the problem instance, but this requires large additional space to store $\Omega(n^2)$ random bits. It is open whether $o(n^2)$ random bits suffice to isolate maximum/min-cost flows, and in our streaming algorithm we cannot afford to store these. This issue does not occur in the setting of communication complexity, as each player can independently store their own perturbed edge weights.

Communication Complexity. There is a general reduction from 2-party communication to streaming, resulting in $O(\textit{space} \times \textit{passes})$ amount of communication, by simulating the streaming algorithm and sending the entire memory in each pass. While our streaming algorithm needs $\widetilde{O}(n^{1.5})$ total space, it generates only $\widetilde{O}(n)$ new information in each of the $\widetilde{O}(\sqrt{n})$ passes. Thus it suffices to send only the $\widetilde{O}(n)$ additional information in each pass, leading to $\widetilde{O}(n^{1.5})$ communication.

▶ **Theorem 2** (Min-cost Flow, Communication). *There is a randomized communication protocol that, given an input graph $G = (V, E)$ with integer capacities $u \in [1, W]^E$ and costs $c \in [1, W]^E$, vertex demand vector $b \in \mathbb{R}^V$, where the edges are split among two parties, computes w.h.p. an exact min-cost flow in $\widetilde{O}(n^{1.5}\log(W))$ communication. At the end, each party knows the amount of flow on each of their own edges.*

Unlike Theorem 1, there is no ϵ dependence in Theorem 2, because we can make the flow unique via Isolation-Lemma. A subquadratic communication algorithm for weighted flows (but also the simpler weighted st-min-cut problem where solutions have size $O(n)$) was stated as open problem in [16].

The same $\widetilde{O}(n^{1.5})$-communication bound is also obtained in concurrent work [37]. We remark that 2-party communication is an easier model than multi-pass streaming, because each party can use unlimited amount of space. In particular, [37] does not give a streaming algorithm because each party explicitly stores f_e on their own edges e, thus using $\Omega(n^2)$ space.

Space and Bit-Complexity. In above theorems, space is measured in words where each word can store a real number. The algorithms also work over Word-RAM and finite bit numbers, where each number is stored as fixed-point number using $\widetilde{O}(\log(nW/\epsilon))$-bit. Thus if we want to measure space complexity of Theorems 1 and 2 in bits, all space, communication and time complexities increase by a logarithmic $\widetilde{O}(\log(nW/\epsilon))$ factor.

1.2 Techniques

Our algorithm is based on solving the linear program representation of min-cost flow. The linear program representation of related problems such as bipartite matching or transshipment to obtain streaming algorithms has been used before, e.g., in [9, 51, 2]. A natural approach for matching is to maintain the smaller n-dimensional dual iterate (i.e., fractional vertex cover). However, this does not extend to min-cost/maximum flows, because the capacity constraints increase the dimension of the dual solution from n to $m+n$.

Our algorithm works by implementing the interior point method/central path method by [19], while maintaining the primal solution (i.e., flow) in low space, rather than the dual. One can show that for flows on the central path, an $\widetilde{O}(n)$ space representation is possible[3]. Unfortunately, this observation does not directly imply a streaming algorithm because in each iteration, the central path method constructs intermediate flows that are not sufficiently centered to be stored in $\widetilde{O}(n)$ space. This issue is exacerbated when using the fast $\widetilde{O}(\sqrt{n})$ iteration central path of the Lee-Sidford-barrier [50] as used in [19]. Their definition of centrality uses Lewis-weights whose efficient computation uses sketching. Due to the resulting approximation error, one cannot compute sufficiently centered flows to store them in $\widetilde{O}(n)$ space. Thus we develop a method that can store non-centered flows in small space.

We do this by storing the steps of each iteration rather than the flow x. In the context of flows, each step of the central path method is equivalent to augmenting the current flow x by an electric circulation. Rather than storing flow x directly, we store the sequence of $\widetilde{O}(\sqrt{n})$ augmenting electric flows. Each circulation is an m-dimensional vector (i.e., every edge carries some flow), but we show that each circulation can be stored in only $\widetilde{O}(n)$ space. Thus we obtain a $\widetilde{O}(n^{1.5})$ space representation of the min-cost flow.
The main technical ideas/ingredients for storing electric circulations are as follows.

(i) The $\widetilde{O}(\sqrt{n})$ iterations framework [50, 19] requires computation of Lewis-weights, a generalization of leverage scores and effective resistances. Via sketching, we can store the Lewis-weights in only $\widetilde{O}(n)$ space. Given any of the $O(n^2)$ edges, we can then query/compute the (approximate) Lewis-weight from only the $\widetilde{O}(n)$ space representation.

[3] Technical details: Centered flows are minimizers of some potential function $\Phi(x)$ subject to $\mathbf{A}^\top x = b$ for edge-vertex incidence matrix $\mathbf{A}$ and demand $b \in \mathbb{R}^V$. Minimizers satisfy $\nabla\Phi(x) = \mathbf{A}y$ for some $y \in \mathbb{R}^V$, thus by storing y we can reconstruct x.

(ii) While electric flows can be stored via vertex potentials (i.e., in $\widetilde{O}(n)$ space), electric circulations generally need $\Omega(m)$ space. That is because electric circulations are a combination/difference $g - f$ of an arbitrary flow $g \in \mathbb{R}^E$ and an electric flow $f \in \mathbb{R}^E$, where f and g satisfy the same vertex demands. To store the electric circulation $g - f$ in low space, we must store g in low space. Via the robust interior point framework [19], one can reduce the dimension of g from m to $\widetilde{O}(1)$, and can thus be stored in small space. The main barrier is showing that this projection $E \to \{1, ..., \widetilde{O}(1)\}$ can be computed from the stored information.

(iii) Electric flows can only be stored via vertex potentials if we know the edges' resistances. Retrieving the flow on edge $e = (u, v)$ from vertex potentials $y \in \mathbb{R}^V$ (i.e., $f_e = (y_v - y_u)/r_e$) requires knowing the edge's resistance r_e. In [50, 19], the resistance r_e is defined w.r.t. the edge's Lewis-weights, and the current congestion of that edge. This allows for an inductive argument: Given an edge e and its capacity c_e, assuming we know the amount of flow on edge e after t electric circulations, then we can compute the resistance r_e of that edge from the current congestion and the stored Lewis-weight. From the resistance, we can then obtain how much flow the next electric circulation will route through e. This then yields the flow on edge e after $t + 1$ electric circulations. After $\widetilde{O}(\sqrt{n})$ iterations of this argument, we know the final amount of flow on that edge, i.e., how much flow is routed on e by the min-cost flow.

Remark on General Linear Programs. Given that our work is based on interior point methods for general $n \times m$-dimensional ($m \geq n$) linear programs, it is a natural question whether our algorithm extends to other applications such as ℓ_1- or ℓ_∞-regression, MDPs, etc. Internally, our algorithm stores $\widetilde{O}(\sqrt{n})$ many n-dimensional vectors, and $\widetilde{O}(n)$ constraints of the linear program. So for a general linear program with k non-zeros per columns ($k = 2$ in case of flow), it would store $\widetilde{O}(n^{1.5} + kn)$ numbers. However, this assumes Real-RAM model, i.e., when ignoring the bit-complexity. When considering the actual number of bits (e.g., Word-RAM), the linear systems to be solved in each iteration pose additional technical challenges that do not occur when restricting to max/min-cost flow where the linear systems are simple Laplacians. Hence, this work focuses on the flow applications, where the (sparse) linear systems can be solved in low space in Word-RAM via Laplacian solvers.

1.3 Other Related Work

Before the semi-streaming lower bounds of $\Omega(n^2/p^5)$ [6] and $\Omega(n^2/p^3)$ [8] for p passes, there have been $n^{1+\Omega(1/p)}$ space lower bounds [21, 39]. For small number $o(\sqrt{\log n})$ passes, there is an $n^{2-o(1)}$ space lower bound for st-reachability, which extends to directed flows [25].

While our focus is on st-flows and thus st-min cuts, there has been work on global min cuts in the streaming model. There are $\widetilde{O}(1)$ pass $\widetilde{O}(n)$ space algorithms for global min cuts [56, 7, 58]. These also transfer to the 2-party communication setting, where $\Theta(n^{1.5})$ upper and lower bound are known for minimum vertex cut [17].

Our algorithm is based on solving linear programs in the streaming model. Other work using this approach (though not for flows, but instead for other linear programs such as bipartite matching) include [51, 2, 10, 23]. From the communication perspective, linear programs have been studied in [59, 36]. The crucial aspect is that previous work studies linear programs of form $\min c^\top$ subject to $\mathbf{A}^\top x = b, x \geq 0$ for tall matrices $\mathbf{A}$, but max-flow requires additional capacity constraints $x \leq u$. Encoding those within the matrix $\mathbf{A}$ turns the matrix from tall into an almost square matrix, thus losing any space/communication gains from the small width of $\mathbf{A}$.

2 Preliminaries

We write "with high probability" (w.h.p.) if the failure probability is at most $1/\operatorname{poly}(n)$. We use χ_e to denote the e-th standard unit vector. For an integer n, we use $[n]$ to denote the set $\{1, 2, \cdots, n\}$. We use $\widetilde{O}(\cdot)$ to hide $\operatorname{poly}(\log m)$ factors. In addition to $O(\cdot)$ notation, for two functions f, g, we use the shorthand $f \lesssim g$ (resp. $\gtrsim$) to indicate that $f \leq C \cdot g$ (resp. $\geq$) for an absolute constant C. For a vector $v \in \mathbb{R}^m$, we use $\|v\|_\infty$ to denote its ℓ_∞ norm, i.e., $\|v\|_\infty : \max_{i\in[m]} |v_i|$. We use $\|v\|_p$ to denote its ℓ_p norm, i.e., $\|v\|_p := (\sum_{i=1}^m |v_i|^p)^{1/p}$. For a positive semi-definite matrix $\mathbf{M}$ we write $\|v\|_{\mathbf{M}} := (v^\top \mathbf{M} v)^{1/2}$. For vectors $x, s, \tau \in \mathbb{R}^m$ we write $\mathbf{X}, \mathbf{S}, \mathbf{T}$ for the diagonal matrices with $\mathbf{X}_{i,i} = x_i, \mathbf{S}_{i,i} = s_i, \mathbf{T}_{i,i} = \tau_i$ on the diagonals for $i \in [m]$. Similarly, for function $\phi : \mathbb{R} \to \mathbb{R}$, we write $\Phi(x)$ for the diagonal matrix with $\Phi(x)_{i,i} = \phi(x_i)$ on the diagonals for $i \in [m]$. Given vectors $a, b \in \mathbb{R}^m$, we use ab to denote entry-wise products, i.e., $(ab)_i = a_i b_i$. We define $\cosh(x) := \frac{1}{2}(\exp(x) + \exp(-x))$. We define $\sinh(x) := \frac{1}{2}(\exp(x) - \exp(-x))$. Note that $\cosh(x)' = \sinh(x)$ and $\sinh(x)' = \cosh(x)$.

The *leverage scores* $\sigma(\mathbf{A}) \in \mathbb{R}^m_{>0}$ of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m \geq n)$ are defined as $\sigma(\mathbf{A})_i := (\mathbf{A}(\mathbf{A}^\top \mathbf{A})^{-1}\mathbf{A}^\top)_{i,i}$ for $i \in [m]$.

For $a, b \in \mathbb{R}$ and $\epsilon > 0$ we write $a \approx_\epsilon b$ when $\exp(-\epsilon)a \leq b \leq \exp(\epsilon)a$. We extend the notation to vectors when the approximation holds entry-wise. For PSD matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$ we write $\mathbf{A} \approx_\epsilon \mathbf{B}$ when for all vectors $v \in \mathbb{R}^n$ $v^\top \mathbf{A} v \approx_\epsilon v^\top \mathbf{B} v$, i.e., $\mathbf{A}, \mathbf{B}$ are spectral approximations of each other.

We say a vector $v \in \mathbb{R}^E$ is given in implicit representation with space $O(S)$ and query time $O(T)$, if we have a data structure that uses $O(S)$ space and supports queries that receive as input edge, edge-cost, and edge-capacity (e, c_e, u_e) and then return v_e in $O(T)$ time.

The line of work [1, 47, 4, 49, 44, 45, 46, 20, 51] developed streaming algorithms for spectral sparsifiers.

▶ **Lemma 3** ([49, Theorem 1.1]). *Let $\mathbf{A} \in \mathbb{R}^{E\times V}$ be an incidence matrix, $w \in \mathbb{R}^E$ be some weights where any w_e can be queried in time T_w, and let $\epsilon > 0$. Then within a single pass over the rows of $\mathbf{A}$ (i.e., edges), we can compute an $\widetilde{O}(n/\epsilon^2)$-sized spectral sparsifier $\mathbf{H} \approx_\epsilon \mathbf{A}^\top \mathbf{A}$. This takes $\widetilde{O}(mT_w)$ total time.*

This lemma was proven for weighted incidence matrices in [49] where w_e is given together with edge e in the stream. It directly implies the above, by instead querying w_e when edge e is read from the stream.

3 Technique Overview

Our algorithm and data structure are based on interior point methods for linear programs. In particular, we show that the algorithm of [19] can be implemented in $\widetilde{O}(n^{1.5})$ space. The main barrier for this is storing the m-dimensional flow vector in subquadratic space. Given that a lot of work on streaming and communication complexity for flows and matching is combinatorial in nature [6, 7, 58, 17, 41, 11, 27, 55, 33], we start with a more graph-oriented perspective for readers with that background. Section 3.1 sketches how we store our flow x in $\widetilde{O}(n^{1.5})$ space, without going too far into the details about the interior point method.

The next subsections 3.2 and 3.3 then give more details of our algorithms using the optimization perspective of interior point methods and central paths. In order to outline time and space complexity guarantees of our streaming result Theorem 1, we need more details about these optimization methods. After outlining these methods in Section 3.2, we explain in Section 3.3 how to implement this interior point methods in the streaming model, resulting in Theorem 1.

3.1 High-Level Idea: Storing Flows in $\widetilde{O}(n^{1.5})$ Space

Here we outline how to store a flow $x \in \mathbb{R}^E$ in low-space, such that when given any edge $e \in E$, its cost c_e, and capacity u_e, then we can query/compute x_e.

Consider the linear programming definition of minimum cost flow. For an $E \times V$ incidence matrix $\mathbf{A} \in \{-1, 0, 1\}^{E \times V}$, edge-cost vector $c \in \mathbb{R}^E$, demand vector $b \in \mathbb{R}^V$, and edge capacities $u \in \mathbb{R}^E$, the following linear program models minimum-cost flows.

$$\min_{x \in \mathbb{R}^E} c^\top x \text{ subject to } \mathbf{A}^\top x = b \quad 0 \le x \le u$$

A common idea for computing minimum cost flows is to start with some initial flow, then repeatedly augment the flow via negative cost cycles or circulations. When solving minimum cost flows via interior point methods such as [52, 50, 53, 13], these circulations are so called "electric circulations." That is, for some feasible flow $x \in \mathbb{R}^E$, we augment it by a circulation $\delta_x \in \mathbb{R}^E$ where

$$x \leftarrow x + \delta_x$$
$$\delta_x = \left(\mathbf{I} - \mathbf{R}_x^{-1}\mathbf{A}(\mathbf{A}^\top \mathbf{R}_x^{-1}\mathbf{A})^\dagger \mathbf{A}^\top\right) g_x.$$

Here $g_x \in \mathbb{R}^E$ is some flow and $\mathbf{R}_x$ is an $E \times E$ diagonal matrix with $(\mathbf{R}_x)_{e,e}$ being resistance of edge e. (Both g_x and $\mathbf{R}_x$ will be defined later to depend on x). Flow δ_x is a circulation, because it satisfies 0-demand $\mathbf{A}^\top \delta_x = 0$. It is an electric circulation because

$$f = -\mathbf{R}_x^{-1}\mathbf{A}(\mathbf{A}^\top \mathbf{R}_x^{-1}\mathbf{A})^\dagger \mathbf{A}^\top g_x \tag{1}$$

is an electric flow for demand $d = -\mathbf{A}^\top g_x \in \mathbb{R}^V$ and resistances $\mathbf{R}_x$ (i.e., among all demand d flows, f minimizes the energy $\sum_{e \in E} (\mathbf{R}_x)_{e,e} f_e^2$).

The idea of our low space representation is that we only store some initial flow $x^0 \in \mathbb{R}^E_{\ge 0}$, and then store all the augmenting circulations $\delta_x^1, \delta_x^2, \ldots$. We will argue that each $\delta_x^t \in \mathbb{R}^E$ can be stored in only $\widetilde{O}(n)$ space. It was proven in [50] that it takes $\widetilde{O}(\sqrt{n}\log(W/\epsilon))$ iterations/augmentations to find a min-cost flow whose cost is at most an additive ϵ off from optimal. Thus $\widetilde{O}(n^{1.5}\log(W/\epsilon))$ total space suffices[4]. We can then query the amount of flow on any edge $e \in E$ by computing $x_e^0 + \sum_{t=1}^{\widetilde{O}(\sqrt{n})} (\delta_x^t)_e$.

Crucial for our low space representation is that the $(t+1)$-th augmenting circulation δ_x^{t+1} depends on the t-th flow $x^t = x^0 + \sum_{k=1}^t \delta_x^k$. The fact that we can store circulation δ_x^{t+1} in only $\widetilde{O}(n)$ space relies heavily on the following property: when querying/computing $(\delta_x^{t+1})_e$ for some edge e, we already know the flow x_e^t on that edge. This assumption is given by induction: if we know x_e^{t-1}, then we can compute $(\delta_x^t)_e$, thus we know $x_e^t = x_e^{t-1} + (\delta_x^t)_e$, and so forth. This way we iteratively reconstruct x_e^t and $(\delta_x^t)_e$ from $t = 1, 2, 3, \ldots$ all the way to the final/optimal flow on that edge e.

To outline these low-space representations, let us start with the initial flow x^0.

Storing the initial flow x^0. We would like to start with an initial flow x_0 that is feasible and can be stored in low space. Fortunately, standard constructions for the initial flow in non-streaming settings [19, 50, 35, 18] can also be stored in low space and thus directly apply. Consider the following construction.

[4] For the rest of the overview, we will hide the $\log(W/\epsilon)$ term in $\widetilde{O}$-notation.

We pick $x_e^0 = u_e/2$, i.e., each edge carries half its capacity as flow. This flow x^0 does not yet satisfy the demand vector $b \in \mathbb{R}^V$, but we can fix this as follows: add an extra star to the graph (i.e., one vertex that is connected to every other vertex) and route the missing/additional flow along the star edges. The star edges are assigned very large $\mathrm{poly}(n\|c\|_\infty)$ cost so the optimal min-cost flow will not use them, i.e., addition of these edges to the graph does not change optimal solution.

We can store this additional star and the amount of flow x_e^0 on the star-edges in $\widetilde{O}(n)$ space since there are only n additional edges. For all edges e of the original graph, we can return $x_e^0 = u_e/2$ since the capacity u_e of the edge is given during the query.

Storing the Electric Flow f. To argue why we can store each electric circulation δ_x in $\widetilde{O}(n)$ space, we argue the storage of $g \in \mathbb{R}^E$ and the electric flow separately.

An electric flow f (see (1)) is given by vertex potentials $\delta_y := (\mathbf{A}^\top \mathbf{R}_x^{-1} \mathbf{A})^\dagger \mathbf{A}^\top g \in \mathbb{R}^V$ which can easily be stored in $\widetilde{O}(n)$ space. Then for any edge $e = (w, z)$, we can compute the electric flow f_e on that edge via

$$f_e = (\mathbf{R}_x^{-1} \mathbf{A} \delta_y)_e = ((\delta_y)_z - (\delta_y)_w)/(\mathbf{R}_x)_{e,e}.$$

However, this also needs access to $(\mathbf{R}_x)_{e,e}$. To guarantee $\widetilde{O}(\sqrt{n})$ iterations suffice, these resistances must be

$$(\mathbf{R}_x)_{e,e} = \tau_e \cdot \left(\frac{1}{x_e^2} - \frac{1}{(u_e - x_e)^2} \right).$$

Here term $\tau \in \mathbb{R}^E$ are so called Lewis-weights, which can be interpreted as a measure of importance for each edge e, and they are a generalization of effective resistance. Without this τ, the number of iterations would increase from $\widetilde{O}(\sqrt{n})$ to $\widetilde{O}(\sqrt{m}) = \widetilde{O}(n)$, and we could no longer guarantee subquadratic space.

The term $1/x_e^2 - 1/(u_e - x_e)^2$ is motivated by the resistance $(\mathbf{R}_x)_{e,e}$ going towards ∞ when x_e or $(u_e - x_e)$ are close to zero. When the resistance is high, then the electric flow will not route a lot of flow through that edge, which is needed to guarantee the capacity constraints $0 \leq x + \delta_x \leq u$.

Observe that, given edge e and its capacity u_e, we can compute x_e, so we can also compute $1/x_e^2 - 1/(u_e - x_e)^2$. If we can also compute τ_e and g_e, then we get $(\delta_x)_e$, and our iterative argument for computing $x_e^t = x_e^{t-1} + (\delta_x^t)_e$ for all $t = 1, ..., \widetilde{O}(\sqrt{n})$ goes through.

The difficulty is that for any edge e, both g_e and τ_e depend not just on x_e, but are a global property depending on the entire flow $x \in \mathbb{R}^E$. So we need to store additional information.

Storing Weights $\tau \in \mathbb{R}^E$. For some $p = 1 - 1/\Theta(\log m)$, the (regularized) ℓ_p-Lewis-weights from [50] are defined as the scaling $\tau \in \mathbb{R}^E_{\geq 0}$ that satisfy the recurrence relation

$$\tau = \sigma(\mathbf{T}^{1/2-1/p} \mathbf{V}^{-1/2} \mathbf{A}) + \frac{n}{m}$$

for $\mathbf{V}_{e,e} = 1/x_e^2 - 1/(u_e - x_e)^2$, $\mathbf{T}_{e,e} = \tau_e$. [19] proved that the algorithm still works when the equality is only satisfied as some $(1 \pm 1/O(\log n))$-approximation.

Cohen and Peng [26] proved that, if we start with some bad approximation w of the Lewis-weights, then the following routine improves the approximation quality by a $1 - p/2 \approx 1/2$ factor

$$w \leftarrow (w^{2/p-1}(\sigma(\mathbf{W}^{1/2-1/p} \mathbf{V}^{-1/2} \mathbf{A}) + n/m))^{p/2}. \tag{2}$$

Since $n/m \leq \tau \leq 1+n/m$, we can start with $w =$ all-1-vector, and use some $\widetilde{O}(1)$ iterations of (2) to get an accurate enough estimate of the Lewis-weight. If we let $w^{(0)}, w^{(1)}, ..., w^{(k)} =: \overline{\tau}$ be the sequence of values computed this way, then the final $\overline{\tau}$ will be a good approximation of the Lewis-weights. We store these values in low space via the recurrence

$$w_e^{(j)} = ((w_e^{(j-1)})^{2/p-1}(\sigma((\mathbf{W}^{(j-1)})^{1/2-1/p}\mathbf{V}^{-1/2}\mathbf{A})_e + n/m))^{p/2}$$

which allows us to query $w_e^{(j)}$ recursively for $j = 1, ..., k$, and thus store $\overline{\tau} =: w^{(k)} \in \mathbb{R}^E$ in low space, if we can store $\sigma((\mathbf{W}^{(j-1)})^{1/2-1/p}\mathbf{V}^{-1/2}\mathbf{A}) \in \mathbb{R}^E$ in low-space for $j = 1, ..., k = \widetilde{O}(1)$.

This entire approach still yields an accurate estimate if we only approximately compute $\sigma(\cdot)$. This is done via the ℓ_2-characterization of $\sigma(\cdot)$: for any matrix $\mathbf{M}$ we have

$$\sigma(\mathbf{M})_e := (\mathbf{M}(\mathbf{M}^\top\mathbf{M})^\dagger\mathbf{M}^\top)_{e,e} = \|\mathbf{M}(\mathbf{M}^\top\mathbf{M})^\dagger\mathbf{M}^\top\chi_e\|_2^2$$

where χ_e is a standard unit-vector. The ℓ_2-norm can be $(1 \pm \epsilon)$-approximated via an $O(\epsilon^{-2}\log n) \times |E|$ Johnson-Lindenstrauss sketching matrix $\mathbf{S}$, i.e., we store for $j = 1, 2, ..., k$

$$\mathbf{P}^{(j)} := \mathbf{S}^{(j)}(\mathbf{W}^{(j-1)})^{1/2-1/p}\mathbf{V}^{-1/2}\mathbf{A}(\mathbf{A}^\top(\mathbf{W}^{(j-1)})^{1-2/p}\mathbf{V}\mathbf{A})^{-1}\mathbf{A})^\dagger$$

where for $\epsilon = 1/\operatorname{polylog}(n)$ matrix $\mathbf{P}^j$ is $\widetilde{O}(1) \times n$ dimensional so takes only $\widetilde{O}(n)$ space. We can query for edge $e = (a, b)$ the value

$$\begin{aligned}\sigma((\mathbf{W}^{(j-1)})^{1/2-1/p}\mathbf{V}^{-1/2}\mathbf{A})_e &\approx \|\mathbf{P}^{(j)}\mathbf{A}^\top\mathbf{V}^{-1/2}(\mathbf{W}^{(j-1)})^{1/2-1/p}\chi_e\|_2 \\ &= \|\mathbf{P}^{(j)}(\chi_b - \chi_a)\|_2^2 \cdot v_e^{-1}(w_e^{(j-1)})^{1-2/p}.\end{aligned}$$

$$w_e^{(j)} = ((w_e^{(j-1)})^{2/p-1}(\|\mathbf{P}^{(j-1)}(\chi_b - \chi_a)\|_2^2 \cdot v_e^{-1}(w_e^{(j-1)})^{1-2/p} + n/m))^{p/2}. \tag{3}$$

Note that in addition to the $\widetilde{O}(n)$ space for $\mathbf{P}^{(1)}, ..., \mathbf{P}^{(k)}$, here we need v_e to compute $w_e^{(k)}$. We have access to that value since $v_e = 1/x_e^2 - 1/(u_e - x_e)^2$ and we already established that we can query x_e when edge e and its capacity u_e are given to us. While computation of x_e requires τ_e and computing τ_e requires x_e, there is actually no circular dependency here. That's because, if x^t,τ^t are the values computed during iteration number t (i.e., $x^t = x^{t-1} + \delta_x^t$ where δ_x^t uses τ^t), then querying x_e^t requires τ_e^t, but querying τ_e^t needs x_e^{t-1}. So after $t \leq \widetilde{O}(\sqrt{n})$ this loop terminates. This does mean though, that we need to keep track/store all previous values of τ.

In summary, we need $\widetilde{O}(n)$ space per iteration to store τ while supporting queries to τ_e, for a total of $\widetilde{O}(n^{1.5})$ space over $\widetilde{O}(\sqrt{n})$ iterations.

Storing the flow g. We here give only a very high-level description because details on how g is defined require further details about the interior point method. The exact choice of g is crucial for the $\widetilde{O}(\sqrt{n})$ iteration count, and storing this flow in low space relies on recent developments in IPM-theory: for the classic Lee-Sidford IPM [50], it is unclear how to store g in such low space, and our low space representation relies on the recently developed variants [19] that are robust against additional approximation errors.

In somewhat simplified terms, g is defined w.r.t. x, τ and reduced costs $s := c - \mathbf{A}\sum_t \delta_y^t$. (Note that for any edge $e = (w, z)$, we can also compute $s_e = c_e - \sum_t(\delta_y^t)_z - (\delta_y^t)_w$ since we already store the vertex potentials $\delta_y^t \in \mathbb{R}^V$ and cost c_e is given at query time.) Flow g also has the property that if for two edges $e \neq e'$ we have $(\tau_e, x_e, s_e) = (\tau_{e'}, x_{e'}, s_{e'})$, then $g_e = g_{e'}$ are the same. Further, because of recent insights in robust interior point methods, we know the algorithm still works when g is defined w.r.t. approximations $\overline{x}, \overline{s}, \overline{\tau}$ that differ

from the exact values by some $(1 \pm \epsilon)$ approximation for $\epsilon = 1/O(\log n)$. Thus we can round each τ_e, x_e, s_e to multiples of $1+\epsilon$, then $g \in \mathbb{R}^E$ contains only $\widetilde{O}(1)$ distinct values. We store these distinct values $\widetilde{g} \in \mathbb{R}^{\widetilde{O}(1)}$ in $\widetilde{O}(1)$ space.

Then during a query, when receiving edge e, we (i) compute x_e, s_e, τ_e, (ii) round them to the nearest multiple of $(1+\epsilon)$, (iii) access the corresponding $\widetilde{g}_i$.

This is somewhat simplified, since the construction of g from x, s, τ is nontrivial. However, those details require further discussion of how the IPM works, which are given in next subsection. With these details, we can then also explain how to implement the construction of these representations in the semi-streaming model. So far, we only discussed how to represent/store these values, but not how to compute them efficiently. This is outlined in Section 3.3, after giving details about the IPM in Section 3.2.

Remarks on 2-Party Communication. The blackbox reduction from 2-party communication to streaming simulates the streaming algorithm. To simulate one pass over the input, party A passes over their own edge set, then sends their entire memory to party B who continues the current pass over their half of the edges. This reduction would lead to only a trivial $\widetilde{O}(n^2)$ communication bound as we have $\widetilde{O}(\sqrt{n})$ iterations with $\widetilde{O}(n^{1.5})$ space. However, observe that the $\widetilde{O}(n^{1.5})$ space usage of our algorithm comes from storing each of $\delta_x^1, ..., \delta_x^{\widetilde{O}(\sqrt{n})}$ in $\widetilde{O}(n)$ space. There is no need to send the entire memory to simulate the streaming algorithm in the 2-party model, because both parties already have the representation of $\delta_x^1, ..., \delta_x^{t-1}$ when computing the representation of the new δ_x^t together when simulating the streaming algorithm. We will see in Section 3.3 that the streaming algorithm needs only $\widetilde{O}(n)$ additional space and $\widetilde{O}(1)$ passes to compute δ_x^t. This implies $\widetilde{O}(n)$ communication for each δ_x^t, i.e., $\widetilde{O}(n^{1.5})$ communication overall.

3.2 Review of LS Central Path/Interior Point Method

As context and motivation for our method, we start by discussing the method of [50]. Readers familiar with the $\widetilde{O}(\sqrt{n})$ iteration interior point methods from [50, 19] can skip to Section 3.3.

For matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$, vectors $b \in \mathbb{R}^n$, $c \in \mathbb{R}^m$, and upper bounds $u \in \mathbb{R}^m$, this IPM solves linear programs of the form

$$\min_{\substack{x \in \mathbb{R}^m : \mathbf{A}^\top x = b \\ 0 \le x_e \le u_e \forall e \in [m]}} c^\top x. \tag{4}$$

The central path method of [50] repeatedly solves the following minimization problem. In each iteration, parameter $\mu \in \mathbb{R}_{>0}$ is decreased, and then the new minimizer x_μ is found by performing one Newton-step from the old minimizer $x_{\mu'}$.

$$x_\mu := \operatorname*{argmin}_{\mathbf{A}^\top x = b} f_\mu(x), \quad f_\mu(x) := c^\top x + \mu \sum_{i=1}^m \tau(x)_e \phi_e(x_e), \tag{5}$$

where $\tau \in \mathbb{R}^m_{>0}$ and $\phi_e : \mathbb{R}_{>0} \to \mathbb{R}$ is defined as $\phi_e(z) = -\log(u_e - z) - \log(z)$. (For simplicity, we also write $\phi(x) \in \mathbb{R}^m$ for the vector with entries $\phi_e(x_e)$ for $e = 1, .., m$.) Note that as x_e approaches 0 or capacity u_e, the $\phi_e(x_e)$ term goes to ∞, which keeps the minimizer x_μ within the feasible region. As $\mu \to 0$, the term $c^\top x$ starts to dominate; thus x_μ converges towards the optimal solution of the linear program for $\mu \to \infty$. Here $\tau \in \mathbb{R}^m$ is a weight measuring the importance of each row of $\mathbf{A}$. [50] chooses τ to be the regularized l_p Lewis-weight for $p = 1 - \frac{1}{4\log(4m/n)}$. This is defined to be the solution to

$$\tau = \sigma(\mathbf{T}^{\frac{1}{2}-\frac{1}{p}}(\Phi''(x))^{-\frac{1}{2}}\mathbf{A}) + \frac{n}{m}, \tag{6}$$

where $\Phi''(x)$ is a diagonal matrix with the 2nd derivatives $\phi_e''(x_e) = 1/x_e^2 - 1/(u_e - x_e)^2$ on the diagonal for $e = 1, ..., m$.

The curve x_μ for $\mu \in (\infty, 0)$ is referred to as the central path. It turns out that it is easy to find a point x close to x_μ on this path for large μ (see full version for details). Starting from this point, we iteratively take steps to decrease μ and move x closer to the new minimizer x_μ.

To do this, we must define a measure for how far x is from x_μ. To satisfy $x = \operatorname{argmin}_{\mathbf{A}^\top x = b} f_\mu(x)$ (Equation (5)), optimality conditions tell us that we need $\mathbf{A}^\top x = b$ and the gradient $\nabla f_\mu(x) = c + \mu\mathbf{T}\phi'(x)$ must be orthogonal to the feasible hyperplane $\{x \mid \mathbf{A}^\top x = b\}$; i.e., $c + \mathbf{A}y + \mu\mathbf{T}\phi'(x) = 0$ for some $y \in \mathbb{R}^n$. We can rewrite this as $s = \mu\mathbf{T}\phi'(x) = 0$, where $s := c + \mathbf{A}y$. Now, to define what it means to follow the central path, we say that a triple (x, s, μ) is *central* if

$$\mathbf{A}^\top x = b, \quad s = \mathbf{A}y + c, \quad s + \mu\mathbf{T}\phi'(x) = 0. \tag{7}$$

As we cannot expect to be exactly centered, we measure centrality via the following potential function:

$$\Psi\left(\frac{s + \mu\tau(x)\phi'(x)}{\mu\tau(x)\sqrt{\phi''(x)}}\right) := \sum_{e=1}^{m} \cosh\left(\lambda\left(\frac{s_e + \mu\tau(x)_e\phi_e'(x_e)}{\mu\tau(x)_e\sqrt{\phi_e''(x_e)}}\right)\right), \tag{8}$$

where $\lambda = O(\log n)$. Note that the numerator is 0 (which is equivalent to minimizing (8)) if and only if (7) is satisfied.

Our goal is now to take steps where we alternate between decreasing μ (which increases (8)) and updating (x, y, s) such that (8) decreases again. This way we approximately trace the curve x_μ.

In the central path method by [19], the improvements in each iterations are of the following form. Given $(x^{t-1}, s^{t-1}, \mu^{t-1})$ at iteration number $t-1$, the next values for iteration t are given by

$$x^t \leftarrow x^{t-1} + \delta_x^t, \quad s^t \leftarrow s^{t-1} + \delta_s^t, \quad \mu^t \leftarrow \left(1 - \frac{1}{\widetilde{O}(\sqrt{n})}\right)\mu^{t-1}.$$

To compute the movements δ_x^t, δ_s^t we first calculate the weights $\tau^t \in \mathbb{R}^m$ of our current point x^{t-1}, and set the vector $g^t \in \mathbb{R}^m$ to be the gradient of (8), which intuitively moves x as close as possible to x_μ on the central path. [19] have shown that instead of exact calculation of τ^t, it suffices to compute weight $\overline{\tau}^t$ that satisfy recurrence (6) only with some $(1 \pm 1/O(\log n))$-approximation. Then, the movements are given by

$$\begin{aligned}
&\mathbf{H}^t \approx_{1/O(\log m)} (\mathbf{A}^\top(\overline{\mathbf{T}}^t)^{-1}\Phi''(x^{t-1})^{-1}\mathbf{A}) \\
&\delta_x^t = \Phi''(x^{t-1})^{-1/2}g^t - (\overline{\mathbf{T}}^t)^{-1}\Phi''(x^{t-1})^{-1}\mathbf{A}(\delta_y^t + \delta_c^t) \\
&\delta_y^t = (\mathbf{H}^t)^{-1}\mathbf{A}^\top\Phi''(x^{t-1})^{-1/2}g^t \\
&\delta_s^t = \mu^t\mathbf{A}\delta_y^t \\
&\delta_c^t = (\mathbf{H}^t)^{-1}(\mathbf{A}^\top x^{t-1} - b) \\
&g^t = \operatorname*{argmax}_{w:\ \|w\|_{\overline{\tau}+\infty}} \langle w, -\nabla\Psi\left(\frac{s + \mu\overline{\tau}(x)\phi'(x)}{\mu\overline{\tau}(x)\sqrt{\phi''(x)}}\right)\rangle
\end{aligned} \tag{9}$$

where $\mathbf{H}^t$ is a spectral approximation, allowing us to solve the linear system approximately (i.e., we can use fast Laplacian system solvers). Because of this approximation, we can no longer guarantee $\mathbf{A}^\top x = b$, but inclusion of vector δ_c^t guarantees $\mathbf{A}^\top x$ stays close to b. Here g^t is essentially the gradient of our potential function Ψ (since we want to reduce that potential in each iteration), but we are taking a step that is bounded in a certain $(\tau + \infty)$-norm[5] (see full version for details). Calculation of this g^t in low space will later be one of the main issues discussed in Section 3.3.

In [50] it was shown that $\widetilde{O}(\sqrt{n})$ iterations of the central path method (9) suffice, i.e. one must compute (9) for $t = 1, 2, ..., \widetilde{O}(\sqrt{n})$. The naive implementation of (9) takes $\Omega(m)$ space which is $\Omega(n^2)$ in worst-case, since the vectors $x^t, s^t, g^t \in \mathbb{R}^m$ are m-dimensional. However, $O(n^2)$ space for min-cost flow is trivial, because then we could solve the problem in a single pass by storing the entire input graph. Therefore, our goal is to implement (9) using less than $O(n^2)$ space.

In the following, we outline how we can implement all calculations in (9) in the semi-streaming model. In particular, we show that for any one t, (9) can be calculated in just $\widetilde{O}(1)$ passes over the input. Since we have $t = 1, 2, ..., \widetilde{O}(\sqrt{n})$, our algorithm takes $\widetilde{O}(\sqrt{n})$ passes in total.

3.3 Streaming Implementation

Our goal is to calculate (9) a total of $\widetilde{O}(\sqrt{n})$ times in a semi-streaming setting with $\widetilde{O}(1)$ passes over the input per iteration, and using only $\widetilde{O}(n^{1.5})$ total space.

We already outlined in Section 3.1 that we can store the values of the t-th iteration in total space $\widetilde{O}(nt)$ by storing each iteration in $\widetilde{O}(n)$ space. The precise values being stored are:

- $y^\ell, \delta_y^\ell, \delta_c^\ell \in \mathbb{R}^n$ for $\ell = 1, ..., t$. This uses $O(nt)$ space.
  This implicitly represents $s^\ell \in \mathbb{R}^m$ because for any edge $e = (w, z)$ and its cost c_e, we can compute $s_e^\ell = c_e - (\mathbf{A}y^\ell)_e = c_e - (y_z^\ell - y_w^\ell)$.
- We store τ^ℓ for $\ell = 1, ..., t$ implicitly in $\widetilde{O}(nt)$ space via a certain recursive JL-sketch.
- We store g^t for $\ell = 1, ..., t$ implicitly in $\widetilde{O}(t)$ space. (Details to be discussed further below).
  This implicitly represents $x^\ell \in \mathbb{R}^m$ because for any edge $e = (w, z)$ and its cost/capacity c_e, u_e, we can recursively compute

$$x_e^\ell = x_e^{\ell-1} + \phi_e''(x_e^{\ell-1})^{-1/2} g_e^\ell - (\overline{\tau}_e^\ell)^{-1} \phi_e''(x_e^{\ell-1})(\mathbf{A}(\delta_y^\ell + \delta_c^\ell))_e.$$

The main remaining question is how to compute the above $\widetilde{O}(n)$ representation for the next iteration $t + 1$ in the streaming model with only $\widetilde{O}(1)$ passes over the input.

Lewis weights $\overline{\tau}^t \in \mathbb{R}^m$. In Section 3.1 we established that in each iteration of the interior point method, the Lewis-weights are computed via $\widetilde{O}(1)$ repetitions of

$$w_e^{(j)} \leftarrow ((w_e^{(j-1)})^{2/p-1}(\sigma((\mathbf{W}^{(j-1)})^{1/2-1/p}\mathbf{M})_e + \frac{n}{m}))^{p/2}. \tag{10}$$

where $\mathbf{M} = \Phi''(x^t)\mathbf{A}$. We start with $w^0 = 1$-vector, and assign $\overline{\tau}^t = w^{(k)}$ for some $k = \widetilde{O}(1)$. For each t, all $w^{(0)}, ..., w^{(k)}$ are stored, so that we can compute $w_e^{(j)}$ recursively from $w_e^{(j-1)}$ whenever an edge e is received.

[5] $\|v\|_{\tau+\infty} := \|v\|_\infty + C \log(4m/n) \cdot \|v\|_\tau$ where $\|v\|_\tau = (\sum_i \tau_i \cdot v_i^2)^{1/2}$ and C is some large constant.

Here the main issue is calculating the $\widetilde{O}(n)$ space representation of $\sigma((\mathbf{W}^{(j-1)})^{1/2-1/p}\mathbf{M}) \in \mathbb{R}^m$. We discussed in Section 3.1 that this value can be represented via a Johnson-Lindenstrauss matrix $\mathbf{S}^{(j)}$ and by computing

$$\mathbf{S}^{(j)}(\mathbf{W}^{(j-1)})^{1/2-1/p}\mathbf{M}(\mathbf{M}^\top(\mathbf{W}^{(j-1)})^{1-2/p}\mathbf{M})^{-1}. \tag{11}$$

Since an approximation suffices, we can replace $\mathbf{M}^\top(\mathbf{W}^{(j-1)})^{1-2/p}\mathbf{M}$ by a spectral approximation, computed in one pass via Lemma 3 (for any given edge e, we have access to the corresponding row of $\mathbf{M} = \Phi''(x^t)\mathbf{A}$ because we have access to x_e^t.)
Once we have the spectral sparsifier, we can compute (11) from left to right via one additional pass over the input. This is explained in more detail in the full version.

Constructing $y^t, \delta_y^t, \delta_c^t$. These vectors are n-dimensional and can thus be computed/stored explicitly. We have by (9) that

$$\begin{aligned}
\mathbf{H}^t &\approx_{1/O(\log m)} (\mathbf{A}^\top(\overline{\mathbf{T}}^t)^{-1}\Phi''(x^{t-1})^{-1}\mathbf{A}) \\
\delta_y^t &= (\mathbf{H}^t)^{-1}\mathbf{A}^\top\Phi''(x^{t-1})^{-1/2}g^t \\
\delta_c^t &= (\mathbf{H}^t)^{-1}(\mathbf{A}^\top x^{t-1} - b) \\
y^t &= y^{t-1} + \delta_y^t
\end{aligned} \tag{12}$$

Note that when receiving edge e on the stream, we can query x_e^{t-1} (and thus $\Phi''(x^{t-1})_e$) and $\overline{\tau}_e^t$ via previous paragraph. Thus $\mathbf{H}^t$ can be constructed in one pass over the input, by constructing spectral sparsifier via Lemma 3. This takes $\widetilde{O}(n)$ space. We can then compute δ_y^t, δ_c^t from right to left in one additional pass over the input.

Constructing gradient $g^t \in \mathbb{R}^m$. We are left with discussing how to construct vector g^t from (9), which can be seen as some gradient. We uses a potential function of the form $\Phi(v^t) := \sum_{e\in[m]} \cosh(\lambda(v_e^t))$ for some fixed $\lambda = O(\log m)$ and vector $v^t \in \mathbb{R}^m$ where

$$v_e^t := \frac{s_e^t + \mu^t\overline{\tau}_e^t\phi_e'(x_e^t)}{\mu^t\overline{\tau}_e^t\sqrt{\phi_e''(x_e^t)}}$$

for each $e \in [m]$. Here the gradient is $\nabla\Phi(v) := -\sum_{e\in[m]} \lambda\sinh(\lambda(v_e^t))$. Observe that to obtain $(\nabla\Phi(v^t))_e$ for any $e \in [m]$, we only need to compute v_e^t. Further, [19] proves that one does not need to compute the exact gradient $\nabla\Phi(v^t)$. Instead, it suffices to compute $\nabla\Phi(\widetilde{v}^t)$ where $\widetilde{v}_e^t$ is the appropriate vector in terms of $\widetilde{x}^t, \widetilde{s}^t, \widetilde{\tau}^t$, which are each element-wise $(1 \pm \epsilon)$ multiplicative approximations of $x^t, s^t, \tau(x^t)$ respectively for some $\epsilon = O(1/\log m)$.

To obtain g^t from $\nabla\Phi(\widetilde{v}^t)$, [19] defines

$$g^t = \nabla\Psi(\widetilde{v})^{\flat(\widetilde{\tau})} := \operatorname*{argmax}_{\|h\|_{\widetilde{\tau}+\infty}=1} h^\top\nabla\sum_{e=1}^m \cosh(\lambda\widetilde{v}_e). \tag{13}$$

The maximization problem in (13) can be efficiently solved when rounding each entry of $\widetilde{v}^t$ and $\widetilde{\tau}^t$ to a multiple of some small $\delta = O(\epsilon)$. The additional approximation error from this rounding process can be charged to the approximation $\widetilde{\tau}^t \approx \tau^t$, $\widetilde{v}^t \approx v^t$. Since the gradient is bounded, this discretization essentially reduces the optimization problem onto some smaller polylog-dimensional space, because we obtain at most some polylog distinct multiples of some $\epsilon = 1/O(\log n)$.

Observe that within a single pass over the input, we can count how many entries of $\widetilde{v}^t$ are rounded to the same value. That is, within a single pass we compute two $\widetilde{O}(1)$-dimensional vectors: $\widetilde{w}^t$ containing the rounded values, and $\widetilde{k}^t$ where $\widetilde{k}_i^t$ counts how many entries have

been rounded to $\widetilde{w}_i^t$. We can now solve (13) locally in $\widetilde{O}(1)$ space, because (i) we know the output vector also has only $\widetilde{O}(1)$ distinct values, (ii) we can compute the norm of the m-dimensional vector h in (13) by using the counting vector (i.e., we know how often each entry will be duplicated).

Summary, space efficient implementation. In each iteration of the central path method, we must compute (9). We have implicit access to x^t, s^t which allows us to compute implicit access to g^t and $\overline{\tau}^t$. We can then construct the spectral sparsifier $\mathbf{H}^t$. With these, we can then within a constant number of passes through the stream compute $\delta_y^{t+1}, \delta_c^{t+1}, y^{t+1}$. We thus have all information that is required for the implicit representation of x^{t+1}, s^{t+1} so we can proceed with the next iteration. After $\widetilde{O}(\sqrt{n})$ iterations, our algorithm terminates. Since each iteration takes $\widetilde{O}(1)$ passes, we take $\widetilde{O}(\sqrt{n})$ passes in total. The space requirement is $\widetilde{O}(n^{1.5})$ since we store $y^t, \delta_c^t, \delta_y^t, \overline{\tau}^t, g^t$ for $t = 1, ..., \widetilde{O}(\sqrt{n})$.

References

1 Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *International Colloquium on Automata, Languages, and Programming*, pages 328–338. Springer, 2009. doi:10.1007/978-3-642-02930-1_27.

2 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 526–538. Springer, 2011. doi:10.1007/978-3-642-22012-8_42.

3 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *ACM Trans. Parallel Comput.*, 4(4):17:1–17:40, 2018. doi:10.1145/3154855.

4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, pages 5–14. ACM, 2012. doi:10.1145/2213556.2213560.

5 Sepehr Assadi. A simple (1 - ϵ)-approximation semi-streaming algorithm for maximum (weighted) matching. In Merav Parter and Seth Pettie, editors, *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 337–354. SIAM, 2024. doi:10.1137/1.9781611977936.31.

6 Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Polynomial pass lower bounds for graph streaming algorithms. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 265–276. ACM, 2019. doi:10.1145/3313276.3316361.

7 Sepehr Assadi and Aditi Dudeja. A simple semi-streaming algorithm for global minimum cuts. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 172–180. SIAM, 2021. doi:10.1137/1.9781611976496.19.

8 Sepehr Assadi, Prantar Ghosh, Bruno Loff, Parth Mittal, and Sagnik Mukhopadhyay. Polynomial pass semi-streaming lower bounds for k-cores and degeneracy. In *CCC*, volume 300 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.CCC.2024.7.

9 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 627–669. SIAM, 2022. doi:10.1137/1.9781611977073.29.

10 Sepehr Assadi, Nikolai Karpov, and Qin Zhang. Distributed and streaming linear programming in low dimensions. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 236–253. ACM, 2019. doi:10.1145/3294052.3319697.

11 Sepehr Assadi, Sanjeev Khanna, and Yang Li. On estimating maximum matching size in graph streams. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1723–1742. SIAM, 2017. doi:10.1137/1.9781611974782.113.

12 Sepehr Assadi and Janani Sundaresan. Hidden permutations to the rescue: Multi-pass streaming lower bounds for approximate matchings. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 909–932. IEEE, 2023. doi:10.1109/FOCS57990.2023.00058.

13 Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *FOCS*. https://arxiv.org/pdf/2003.04863, 2020.

14 László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory (preliminary version). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 337–347. IEEE Computer Society, 1986. doi:10.1109/SFCS.1986.15.

15 Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *SIAM J. Comput.*, 50(3):815–856, 2021. doi:10.1137/19M1286955.

16 Joakim Blikstad, Jan van den Brand, Yuval Efron, Sagnik Mukhopadhyay, and Danupon Nanongkai. Nearly optimal communication and query complexity of bipartite matching. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1174–1185. IEEE, 2022. doi:10.1109/FOCS54457.2022.00113.

17 Joakim Blikstad, Yonggang Jiang, Sagnik Mukhopadhyay, and Sorrachai Yingchareonthawornchai. Global vs. s-t vertex connectivity beyond sequential: Almost-perfect reductions and near-optimal separations. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 2305–2316. ACM, 2025. doi:10.1145/3717823.3718316.

18 Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 543–556. ACM, 2022. doi:10.1145/3519935.3520068.

19 Jan van den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and l1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 859–869, 2021. doi:10.1145/3406325.3451108.

20 Charles Carlson, Alexandra Kolla, Nikhil Srivastava, and Luca Trevisan. Optimal lower bounds for sketching graph cuts. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2565–2569. SIAM, 2019. doi:10.1137/1.9781611975482.158.

21 Amit Chakrabarti, Prantar Ghosh, Andrew McGregor, and Sofya Vorotnikova. Vertex ordering problems in directed graph streams. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1786–1802. SIAM, 2020. doi:10.1137/1.9781611975994.109.

22 Amit Chakrabarti, Jeffrey Jiang, David Woodruff, and Taisuke Yasuda. Streaming algorithms for ℓ_p flows and ℓ_p regression. In *ICLR*. OpenReview.net, 2025.

23 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. In Joseph S. B. Mitchell and Günter Rote, editors, *Proceedings of the 21st ACM Symposium on Computational Geometry, Pisa, Italy, June 6-8, 2005*, pages 180–189. ACM, 2005. doi:10.1145/1064092.1064121.

24 Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM J. Comput.*, 24(5):1036–1050, 1995. doi:10.1137/S0097539793250330.

25 Lijie Chen, Gillat Kol, Dmitry Paramonov, Raghuvansh R. Saxena, Zhao Song, and Huacheng Yu. Almost optimal super-constant-pass streaming lower bounds for reachability. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 570–583. ACM, 2021. doi:10.1145/3406325.3451038.

26 Michael B. Cohen and Richard Peng. L_p row sampling by lewis weights. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 183–192. ACM, 2015. doi:10.1145/2746539.2746567.

27 Michael S. Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, volume 28 of *LIPIcs*, pages 96–104. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

28 Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 451–460. ACM, 2008. doi:10.1145/1374376.1374441.

29 Shahar Dobzinski, Noam Nisan, and Sigal Oren. Economic efficiency requires interaction. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 233–242. ACM, 2014. doi:10.1145/2591796.2591815.

30 Pavol Duris and Pavel Pudlák. On the communication complexity of planarity. In János Csirik, János Demetrovics, and Ferenc Gécseg, editors, *Fundamentals of Computation Theory, International Conference FCT'89, Szeged, Hungary, August 21-25, 1989, Proceedings*, volume 380 of *Lecture Notes in Computer Science*, pages 145–147. Springer, 1989. doi:10.1007/3-540-51498-8_14.

31 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1-2):490–508, 2012. doi:10.1007/S00453-011-9556-8.

32 Sebastian Eggert, Lasse Kliemann, and Anand Srivastav. Bipartite graph matchings in the semi-streaming model. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 492–503. Springer, 2009. doi:10.1007/978-3-642-04128-0_44.

33 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. doi:10.1016/J.TCS.2005.09.013.

34 Maxime Flin and Parth Mittal. $(\Delta+1)$ vertex coloring in $O(n)$ communication. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 416–424. ACM, 2024. doi:10.1145/3662158.3662796.

35 Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 516–527. IEEE, 2021. doi:10.1109/FOCS52979.2021.00058.

36 Mehrdad Ghadiri, Yin Tat Lee, Swati Padmanabhan, William Swartworth, David P. Woodruff, and Guanghao Ye. Improving the bit complexity of communication for distributed convex optimization. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1130–1140. ACM, 2024. doi:10.1145/3618260.3649787.

37 Hossein Gholizadeh and Yonggang Jiang. A subquadratic two-party communication protocol for minimum cost flow. *CoRR*, abs/2510.03427, 2025. doi:10.48550/arXiv.2510.03427.

38 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485. SIAM, 2012. doi:10.1137/1.9781611973099.41.

39 Venkatesan Guruswami and Krzysztof Onak. Superlinear lower bounds for multipass graph processing. *Algorithmica*, 76(3):654–683, 2016. doi:10.1007/S00453-016-0138-7.

40 András Hajnal, Wolfgang Maass, and György Turán. On the communication complexity of graph properties. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 186–191. ACM, 1988. doi:10.1145/62212.62228.

41 Gábor Ivanyos, Hartmut Klauck, Troy Lee, Miklos Santha, and Ronald de Wolf. New bounds on the classical and quantum communication complexity of some graph properties. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 148–159. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.148.

42 Yonggang Jiang, Danupon Nanongkai, and Pachara Sawettamalya. Minimum s t cuts with fewer cut queries. In Kasper Green Larsen and Barna Saha, editors, *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pages 258–296. SIAM, 2026. doi:10.1137/1.9781611978971.12.

43 Michael Kapralov. Better bounds for matchings in the streaming model. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697. SIAM, 2013. doi:10.1137/1.9781611973105.121.

44 Michael Kapralov, Yin Tat Lee, Cameron Musco, Christopher Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM J. Comput.*, 46(1):456–477, 2017. doi:10.1137/141002281.

45 Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, and Jakab Tardos. Fast and space efficient spectral sparsification in dynamic streams. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1814–1833. SIAM, 2020. doi:10.1137/1.9781611975994.111.

46 Michael Kapralov, Navid Nouri, Aaron Sidford, and Jakab Tardos. Dynamic streaming spectral sparsification in nearly linear time and space. In *arXiv preprint*, 2019.

47 Jonathan A Kelner and Alex Levin. Spectral sparsification in the semi-streaming setting. In *STACS*, 2011.

48 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, volume 7408 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2012. doi:10.1007/978-3-642-32512-0_20.

49 Rasmus Kyng, Jakub Pachocki, Richard Peng, and Sushant Sachdeva. A framework for analyzing resparsification algorithms. In *SODA*, pages 2032–2043. SIAM, 2017. doi:10.1137/1.9781611974782.132.

50 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $O(\sqrt{rank})$ iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014. doi:10.1109/FOCS.2014.52.

51 S Cliff Liu, Zhao Song, Hengjie Zhang, Lichen Zhang, and Tianyi Zhou. Space-efficient interior point method, with applications to linear programming and maximum weight bipartite matching. In *ICALP*, 2023.

52 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 253–262. IEEE, 2013. doi:10.1109/FOCS.2013.35.

53 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016. doi:10.1109/FOCS.2016.70.

54 Nikhil S. Mande, Manaswi Paraashar, Swagato Sanyal, and Nitin Saurabh. On the communication complexity of finding a king in a tournament. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, August 28-30, 2024, London School of Economics, London, UK*, volume 317 of *LIPIcs*, pages 64:1–64:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.APPROX/RANDOM.2024.64.

55 Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2005. doi:10.1007/11538462_15.

56 Sagnik Mukhopadhyay and Danupon Nanongkai. Weighted min-cut: sequential, cut-query, and streaming algorithms. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 496–509. ACM, 2020. doi:10.1145/3357713.3384334.

57 Christos H. Papadimitriou and Michael Sipser. Communication complexity. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 196–200. ACM, 1982. doi:10.1145/800070.802192.

58 Aviad Rubinstein, Tselil Schramm, and S. Matthew Weinberg. Computing exact minimum cuts without knowing the graph. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.39.

59 Santosh S. Vempala, Ruosong Wang, and David P. Woodruff. The communication complexity of optimization. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1733–1752. SIAM, 2020. doi:10.1137/1.9781611975994.106.