Space-Efficient Interior Point Method, with Applications to Linear Programming and Maximum Weight Bipartite Matching

S. Cliff Liu
Carnegie Mellon University, Pittsburgh, PA, USA

Zhao Song
Adobe Research, San Jose, CA, USA

Hengjie Zhang
Columbia University, New York, NY, USA

Lichen Zhang
Massachusetts Institute of Technology, Cambridge, MA, USA

Tianyi Zhou
University of California San Diego, CA, USA

Abstract

We study the problem of solving linear program in the streaming model. Given a constraint matrix $A \in \mathbb{R}^{m \times n}$ and vectors $b \in \mathbb{R}^m, c \in \mathbb{R}^n$, we develop a space-efficient interior point method that optimizes solely on the dual program. To this end, we obtain efficient algorithms for various different problems:

- For general linear programs, we can solve them in $\widetilde{O}(\sqrt{n}\log(1/\epsilon))$ passes and $\widetilde{O}(n^2)$ space for an ϵ-approximate solution. To the best of our knowledge, this is the most efficient LP solver in streaming with no polynomial dependence on m for both space and passes.
- For bipartite graphs, we can solve the minimum vertex cover and maximum weight matching problem in $\widetilde{O}(\sqrt{m})$ passes and $\widetilde{O}(n)$ space.

In addition to our space-efficient IPM, we also give algorithms for solving SDD systems and isolation lemma in $\widetilde{O}(n)$ spaces, which are the cornerstones for our graph results.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Linear programming

Keywords and phrases Convex optimization, interior point method, streaming algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.88

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2009.06106.pdf`

Funding *Lichen Zhang*: Supported by NSF grant No. CCF-1955217 and NSF grant No. CCF-2022448.

Acknowledgements The authors would like to thank Jonathan Kelner for many helpful discussions.

1 Introduction

Given a constraint matrix $A \in \mathbb{R}^{m \times n}$, vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, the linear program problem asks us to solve the primal program (P) or its dual (D):

$$(P) = \max_{A^\top y \leq c, y \geq 0} b^\top y \text{ and } (D) = \min_{Ax \geq b} c^\top x \tag{1}$$

is one of the most fundamental problems in computer science and operational research. Many efforts have been dedicated to develop time-efficient linear program solvers in the past half a century, such as the simplex method [23], ellipsoid method [44] and interior point method [41]. In the last few years, speeding up linear program solve via interior point method (IPM) has been heavily studied [20, 55, 13, 35, 65, 25, 71]. The state-of-the-art IPM has the runtime of $O(m^{2+1/18} + m^{\omega})$ when $m \approx n$ and $O(mn + n^3)$ when $m \gg n$. To achieve these impressive improvements, most of these algorithms utilize randomized and dynamic data structures to maintain the primal and dual solutions simultaneously. While these algorithms are time-efficient, it is highly unlikely that they can be implemented in a *space-efficient* manner: maintaining the primal-dual formulation requires $\Omega(m + n^2)$ space, which is particularly unsatisfactory when $m \gg n$.

In this paper, we study the problem of solving a linear program in the streaming model: At each pass, we can query the i-th row of A and the corresponding of the b. The goal is to design an LP solver that is both space and pass-efficient. By efficient, our objective is to obtain an algorithm with no polynomial dependence on m, or more concretely, we present a robust IPM framework that uses only $\widetilde{O}(n^2)$ space and $\widetilde{O}(\sqrt{n}\log(1/\epsilon))$ passes.[1] To the best of our knowledge, this is the most efficient streaming LP algorithm that achieves a space and pass independent of m. Current best streaming algorithms for LP either require $\Omega(n)$ passes or $\Omega(n^2 + m^2)$ space for $O(\sqrt{n})$ passes. For the regime of tall dense LP ($m \gg n$), our algorithm achieves the best space and passes.

The key ingredient for obtaining these LP algorithms is a paradigm shift from the time-efficient primal-dual IPM to a less time-efficient dual-only IPM [64]. From a time perspective, dual-only IPM requires $\widetilde{O}(\sqrt{n}\log(1/\epsilon))$ iterations, with each iteration can be computed in $\widetilde{O}(mn + \mathrm{poly}(n))$ time. However, it is much more space-efficient than that of primal-dual approach. Specifically, we show that per iteration, it suffices to maintain an $n \times n$ Hessian matrix in place. To obtain $\widetilde{O}(\sqrt{n}\log(1/\epsilon))$ passes, we show that non-trivial quantities such as the Lewis weights [56, 21] can be computed recursively, in an in-place fashion with only $\widetilde{O}(n^2)$ space.

Now that we have a space and pass-efficient IPM for general LP in the streaming model, we instantiate it with applications for graph problems in the semi-streaming model. In the semi-streaming model, each edge is revealed along with its weight in an online fashion and might subject to an adversarial order, and the algorithm is allowed to make multiples passes over the stream in $\widetilde{O}(n)$ space.[2] We particularly focus on the *maximum weight bipartite matching* problem, in which the edges with weights are streamed to us, and the goal is to find a matching that maximizes the total weights in it. While there is a long line of research ([2, 36, 24, 3, 9] to name a few) on this problem, most algorithms can only compute an approximate matching, meaning that the weight is at least $(1 - \epsilon)$ of the maximum weight. For the case of exact matching, a recent work [6] provides an algorithm that takes $n^{4/3+o(1)}$ passes in $\widetilde{O}(n)$ space for computing a maximum *cardinality* matching. It remains an open question to compute an exact maximum *weight* bipartite matching in semi-streaming model, with $o(n)$ passes.

We answer this question by presenting a semi-streaming algorithm that uses $\widetilde{O}(n)$ space and $\widetilde{O}(\sqrt{m})$ passes, this means that as long as the graph is relatively sparse, i.e., $m = o(n^2)$, we achieve $o(n)$ passes. To obtain an $\widetilde{O}(n)$ space algorithm for *any graph*, we require

[1] We use $\widetilde{O}(\cdot)$ notation to hide polylogarithmic dependence on n and m.

[2] Some authors define the space in the streaming model to be the number of cells, where each cell can hold $O(\log n)$ bits or even a number with infinite precision. Our bounds remain unchanged even if each cell only holds $O(1)$ bits, i.e., when arithmetic only applies to $O(1)$-bits operands.

additional machinery; more specifically, for each iteration of our dual-only IPM, we need to compute the Newton step via a symmetric diagonal dominant (SDD) solve in $\widetilde{O}(n)$ space. Since the seminal work of Spielman and Teng [67], many efforts have been dedicated in designing a time-efficient SDD system solver [45, 46, 43, 19]. This solvers run in $\widetilde{O}(m)$ time with improved dependence on the logarithmic terms. However, all of them require $\widetilde{\Theta}(m)$ space. To achieve $\widetilde{O}(n)$ space, we make use of small-space spectral sparsifiers [38] as preconditioners to solve the system in a space and pass-efficient manner.

Finally, we note that with $\widetilde{O}(n)$ space, we essentially solve the dual problem, which is the *generalized minimum vertex cover* on bipartite graph. To turn a solution on vertices to a solution on edges, we utilize the isolation lemma [60] and implement it in $\widetilde{O}(n)$ bits via a construction due to [17].

1.1 Our contribution

In this section, we showcase three main results of this paper and discuss their consequences.

The first result regards solving a general linear program in the streaming model with $\widetilde{O}(n^2)$ space and $\widetilde{O}(\sqrt{n}\log(1/\epsilon))$ passes.

▶ **Theorem 1** (General LP, informal version of Theorem 7.4 in Full version [57]). *Given a linear program with m constraints and n variables and $m \geq n$ in the streaming model, there exists an algorithm that outputs an ϵ-approximate solution to the dual program (Eq. (1)) in $\widetilde{O}(n^2)$ space and $\widetilde{O}(\sqrt{n}\log(1/\epsilon))$ passes.*

By ϵ-approximate solution, we mean that the algorithm finds $x \in \mathbb{R}^n$ such that $c^\top x - c^\top x^* \leq \epsilon$, where x^* is the optimal solution. The key to obtain our result is a small space implementation of leverage score and Lewis weights, so that we can utilize the Lee-Sidford barrier [51], with the number of passes depending on the smaller dimension.

In conjunction with an SDD solver in $\widetilde{O}(n)$ space, our next result shows that in the semi-streaming model, we can solve the minimum vertex cover problem on a bipartite graph with $\widetilde{O}(\sqrt{m})$ passes.

▶ **Theorem 2** (Minimum vertex cover, informal version of Theorem 9.7 in Full version [57]). *Given a bipartite graph G with n vertices and m edges, there exists a streaming algorithm that computes a minimum vertex cover of G in $\widetilde{O}(\sqrt{m})$ passes and $\widetilde{O}(n)$ space with probability $1 - 1/\operatorname{poly}(n)$.*[3]

The reason we end up with $\widetilde{O}(\sqrt{m})$ passes instead of $\widetilde{O}(\sqrt{n})$ passes is that to compute some fundamental quantities such as leverage scores or Lewis weights, we need to solve $\Theta(m)$ SDD systems and result in a total of $\widetilde{O}(m\sqrt{n})$ passes. By using the logarithmic barrier, we only need to solve $O(1)$ SDD systems per iteration, which gives the $\widetilde{O}(\sqrt{m})$ passes.

We are now ready to present our result for bipartite matching in $\widetilde{O}(\sqrt{m})$ passes, which solves the longstanding problem of whether maximum weight matching can be solved in $o(n)$ passes for any $m = n^{2-c}$ with $c > 0$.

▶ **Theorem 3** (Maximum weight bipartite matching, informal version of Theorem 10.1 in Full version [57]). *Given a bipartite graph G with n vertices and m edges, there exists a streaming algorithm that computes an (exact) maximum weight matching of G in $\widetilde{O}(\sqrt{m})$ passes and $\widetilde{O}(n)$ space with probability $1 - 1/\operatorname{poly}(n)$.*

[3] We can actually solve a *generalized* version of the minimum vertex cover problem in bipartite graph: each edge e needs to be *covered* for at least $b_e \in \mathbb{Z}^+$ times, where the case of $b = \mathbf{1}_m$ is the classic minimum vertex cover.

Our matching result relies on turning the solution to the dual minimum vertex cover problem, to a primal solution for the maximum weight matching. We achieve so by an $\widetilde{O}(n)$ space implementation of the isolation lemma [60, 17].

1.2 Related work

Interior point method for solving LP. The interior point method was originally proposed by Karmarkar [41] for solving linear program. Since then, there is a long line of work on speeding up interior point method for solving classical optimization problems, e.g., linear program [68, 64, 69, 61, 22, 50, 51, 52, 20, 55, 53, 12, 13, 71, 35, 25, 65, 33]. In 1987, the running time of LP solver becomes $O(n^3)$ [68, 64]. In 1989, Vaidya proposed an $O(n^{2.5})$ LP solver based on a specific implementation of IPMs, known as the *central path algorithm* [68, 69]. Lee and Sidford show how to solve LP in $\sqrt{n}(\text{nnz}(A) + n^\omega)$ time [49, 50, 51], where ω is the exponent of matrix multiplication [70, 48, 4][4] (the first $\sqrt{n}$-iteration IPM). In 2019, [20] show how to solve LP in $n^\omega + n^{2.5-\alpha} + n^{2+1/6}$, where α is the dual exponent of matrix multiplication [31][5]. This is the first breakthrough result improving $O(n^{2.5})$ from 30 years ago. Later, [35] improved that running time to $n^\omega + n^{2.5-\alpha} + n^{2+1/18}$ by maintaining two layers of data-structure instead of one layer of data-structure as [20]'s algorithm. In 2020, [13] improved the running time of LP solver on tall matrices to mn when $m \geq \text{poly}(n)$. Another line of work focuses on solving linear program with small treewidth [25, 71] in time $\widetilde{O}(m\tau^2)$.

Small space algorithms for solving LP. Simplex algorithm is another popular approach to solve linear programs. It has an even better compatibility with streaming algorithms. For instance, [15] shows that the non-recursive implementation of Clarkson's algorithm [18] gives a streaming LP solver that uses $O(n)$ passes and $\widetilde{O}(n\sqrt{m})$ space. They also show that the recursive implementation gives a streaming LP solver that uses $n^{O(1/\delta)}$ passes and $(n^2 + m^\delta)\,\text{poly}(1/\delta)$ space. [7] proposes a streaming algorithm for solving n-dimensional LP that uses $O(nr)$ pass and $O(m^{1/r})\,\text{poly}(n, \log m)$ space, where $r \geq 1$ is a parameter. All above algorithms needs space depending on m.

Streaming algorithms for approximate matching. Maximum matching has been extensively studied in the streaming model for decades, where almost all of them fall into the category of approximation algorithms. For algorithms that only make one pass over the edges stream, researchers make continuous progress on pushing the constant approximation ratio above $1/2$, which is under the assumption that the edges are arrived in a uniform random order [37, 5, 27, 11]. The random-order assumption makes the problem easier (at least algorithmically). A more general setting is multi-pass streaming with adversarial edge arriving. Under this setting, the first streaming algorithm that beats the $1/2$-approximation of bipartite cardinality matching is [29], giving a $2/3 \cdot (1-\epsilon)$-approximation in $1/\epsilon \cdot \log(1/\epsilon)$ passes. The first to achieve a $(1-\epsilon)$-approximation is [59], which takes $(1/\epsilon)^{1/\epsilon}$ passes.[6] Since then, there is a long line of research in proving upper bounds and lower bounds on the number of passes to compute a maximum matching in the streaming model [2, 26, 32, 26, 36, 24, 3, 8, 10, 9] (see next subsection for more details). Notably, [2, 3] use linear programming and duality theory (see the next subsection for more details).

[4] Currently, $\omega \approx 2.37$.
[5] Currently, $\alpha \approx 0.31$.
[6] For the weighted case, there is a $(1/2-\epsilon)$-approximation algorithm that only takes one pass [62].

However, all the algorithms above can only compute an approximate maximum matching: to compute a matching whose size is at least $(1-\epsilon)$ times the optimal, one needs to spend $\text{poly}(1/\epsilon)$ passes (see [24, 3] and the references therein). For readers who are interested in the previous techniques for solving matching, we refer to Section A in full version [57] which contains a brief summary.

Recent developments for exact matching. Recently, [6] proposes an algorithm that computes a $(1-\epsilon)$-approximate maximum *cardinality* matching in $O(\epsilon^{-1}\log n\log\epsilon^{-1})$ passes and $\widetilde{O}(n)$ space. Their method leverages recent advances in ℓ_1-regression with several ideas for implementing it in small space, leading to a streaming algorithm with no dependence on ϵ in the space usage, and thus improving over [3]. The resulted semi-streaming algorithm computes an exact maximum *cardinality* matching (not for weighted) in $n^{3/4+o(1)}$ passes.

Streaming spectral sparsifer. Initialized by the study of cut sparsifier in the streaming model [1], a simple one-pass semi-streaming algorithm for computing a spectral sparsifier of any weighted graph is given in [42], which suffices for our applications in this paper. The problem becomes more challenging in a dynamic setting, i.e., both insertion and deletion of edges from the graph are allowed. Using the idea of linear sketching, [38] gives a single-pass semi-streaming algorithm for computing the spectral sparsifier in the dynamic setting. However, their brute-force approach to recover the sparsifier from the sketching uses $\Omega(n^2)$ time. An improved recover time is given in [39] but requires more spaces, e.g., $\epsilon^{-2}n^{1.5}\log^{O(1)}n$. Finally, [40] proposes a single-pass semi-streaming algorithm that uses $\epsilon^{-2}n\log^{O(1)}n$ space and $\epsilon^{-2}n\log^{O(1)}n$ recover time to compute an ϵ-spectral sparsifier which has $O(\epsilon^{-2}n\log n)$ edges. Note that $\Omega(\epsilon^{-2}n\log n)$ space is necessary for this problem [14].

SDD solver. There is a long line of work focusing on fast SDD solvers [66, 45, 46, 43, 19, 63, 47]. Spielman and Teng give the first nearly-linear time SDD solver, which is simplified with a better running time in later works. The current fastest SDD solver runs in $O(m\log^{1/2}n\,\text{poly}(\log\log n)\log(1/\epsilon))$ time [19]. All of them require $\widetilde{\Theta}(m)$ space.

2 Technical overview

We start with an overview of our IPM framework. We first note that many recent fast IPM algorithms do not fit into $\widetilde{O}(n^2)$ space. Algorithms such as [51, 35, 13] need to maintain *both* primal and dual solutions, thus require $\Omega(m)$ space. In fact, any algorithms that rely on the primal formulation will need $\Omega(m)$ space to maintain the solution. To bypass this issue, we draw inspiration from the state-of-the-art SDP solver [34]: in their setting, $m=\Omega(n^2)$, which means any operation on the dimension m will be too expensive to perform. They instead resort to the *dual-only* formulation. The dual formulation provides a more straightforward optimization framework on small dimension and makes it harder to maintain key quantities. This is exactly what we want: an algorithm that operates on the smaller dimension, removing the polynomial dependence on m. While efficient maintenance is the key to design time-efficient IPM, it is less a concern for us since our constraining resource is space, not time. To this end, we show that Renegar's IPM algorithm [64] can be implemented in a streaming fashion with only $\widetilde{O}(n^2)$ space. As the number of passes of an IPM crucially depends on the barrier function being used, the [64] algorithm only gives a pass bound of $\widetilde{O}(\sqrt{m}\log(1/\epsilon))$. To further improve the number of passes required, we show that the nearly-universal barrier of Lee and Sidford [51, 53] can also be implemented in $\widetilde{O}(n^2)$ space.

This involves computing Lewis weights in an extremely space-efficient manner. We present a recursive algorithm with $\widetilde{O}(1)$ depth, based on [28], that uses only $\widetilde{O}(n^2)$ space. This gives the desired $\widetilde{O}(\sqrt{n}\log(1/\epsilon))$ passes.

We now turn to our graph results, which is a novel combination of the space-efficient IPM, SDD solvers, duality and the isolation lemma. Note that for both graph problems only allow $\widetilde{O}(n)$ space, so it won't suffice to directly apply our IPM algorithms .

To give a better illustration of the $\widetilde{O}(n)$ space constraint, note that storing a matching already takes $\widetilde{\Theta}(n)$ space, meaning that we have only a polylogarithmic space overhead per vertex to store auxiliary information. The conventional way of solving maximum bipartite matching using an IPM solver would get stuck at the very beginning - maintaining the solution of the relaxed linear program, which is a fractional matching, already requires $\Omega(m)$ space for storing all LP constraints, which seems inevitable.

Our key insight is to show that solving the *dual* form of the above LP, which corresponds to the generalized (fractional) minimum vertex cover problem, is sufficient, and therefore only $\widetilde{O}(n)$ space is needed for maintaining a fractional solution. We use several techniques to establish this argument. The first idea is to use complementary slackness for the dual solution to learn which n edges will be in the final maximum matching and therefore reduce the size of the graph from m to n. However, this is not always the case: For instance, in a bipartite graph that admits a perfect matching, all left vertices form a minimum vertex cover, but the complementary slackness theorem gives no information on which edges are in the perfect matching. To circumvent this problem, we need to slightly perturb the weight on every edge, so that the minimum vertex cover (which is now unique) indeed provides enough information. We use the isolation lemma [60] to realize this objective.

It is then instructive to implement the isolation lemma in limited space. Perturbing the weight on every edge requires storing $\widetilde{O}(m)$ bits of randomness, since the perturbation should remain identical across two different passes. We bypass this issue by using the generalized isolation lemma proposed by [17], in which only $O(\log(Z))$ bits of randomness is needed, where Z is the number of candidates. In our case, $Z \leq n^n$ is the number of all possible matchings. So $\widetilde{O}(n)$ space usage perfectly fits into the semi-streaming model. We design an oracle that stores $\widetilde{O}(n)$ random bits and outputs the same perturbations for all edges in all passes.

Now that we can focus on solving the minimum vertex cover problem in $\widetilde{O}(n)$ space. When the constraint matrix is an incidence matrix, each iteration of our IPM can be implemented as an SDD (or Laplacian) solver, so it suffices to show how to solve SDD system in the semi-streaming model, which, to the best of our knowledge, has not been done prior to our work.

In the following subsections we elaborate on each of the above components:

- In Section 2.1, we provide a high-level picture of how our dual-only interior point method works.
- In Section 2.2, we show evidences that our interior point method can run in space independent of m for all of the three different barriers.
- In Section 2.3, we describe our contribution on our implementations of SDD solver, IPM, and the isolation lemma in the streaming model. We show a novel application of the isolation lemma to turn dual into primal.

2.1 Dual-only robust IPM

The cornerstone of our results is to design a robust IPM framework that works only on the dual formulation of the linear program. The framework fits in barriers including the logarithmic barrier, hybrid barrier and Lee-Sidford barrier. It is also robust enough as it can tolerate approximation errors in many quantities, while preserving the convergence behavior.

Algorithm 1 is a simplified version of our dual-only IPM. The earlier works of Renegar's algorithm [64] require the Newton's direction be computed exactly as $\Delta x = -H(x)^{-1}\nabla f_t(x)$, in order to get double exponential convergence rate of Newton's method. To strengthen its guarantee, we develop a more robust framework for this IPM. Specifically, we show that the Hessian of the barrier functions, the gradient and the Newton's direction can all be approximated. This requires a much more refined error analysis. Below, we carefully bound the compound errors caused by three layers of approximations.

First, from Δx to δ_x (Line 9), we allow our Hessian to be spectrally approximated within any small constant factor. This provides us enough leeway to implement the Hessian of barrier functions in a space-efficient manner. For example, the Hessian of the volumetric barrier is $H(x) = A_x^\top(3\Sigma_x - 2P_x^{(2)})A_x$, where Σ_x is a diagonal matrix and $P_x^{(2)}$ is taking entry-wise square of a dense projection matrix. But $\widetilde{H}(x) = A_x^\top \Sigma_x A_x$ is a 5-approximation of $H(x)$ and we can compute it in the same space as computing leverage scores.

Second, from δ_x (Line 9) to δ'_x (Line 11), we allow approximation on the gradient in the sense that it has small local norm with respect to the true gradient, i.e., $\|\nabla f_t(x) - \widetilde{\nabla} f_t(x)\|_{H(x)^{-1}} \leq 0.1$.[7] To give a concrete example, let $\sigma \in \mathbb{R}^m$ denote the leverage score vector, and suppose the Hessian matrix is in the form of $H(x) = A^\top \Sigma A$ and the gradient is $\nabla f(x) = A^\top \sigma$. The leverage score σ can then be approximated in an entry-wise fashion: each entry can tolerate a multiplicative $(1 \pm O(1/\sqrt{n}))$ error. This is because

$$\begin{aligned}
\|\nabla f_t(x) - \widetilde{\nabla} f_t(x)\|^2_{H(x)^{-1}} &= \mathbf{1}_m^\top(\Delta\Sigma)A(A^\top\Sigma A)^{-1}A^\top(\Delta\Sigma)\mathbf{1}_m \\
&= \mathbf{1}_m^\top(\Delta\Sigma)\Sigma^{-1/2}\Sigma^{1/2}A(A^\top\Sigma A)^{-1}A^\top\Sigma^{1/2}\Sigma^{-1/2}(\Delta\Sigma)\mathbf{1}_m \\
&\leq \mathbf{1}_m^\top(\Delta\Sigma)\Sigma^{-1}(\Delta\Sigma)\mathbf{1}_m \\
&= \sum_{i=1}^m \frac{(\sigma_i - \widetilde{\sigma}_i)^2}{\sigma_i} \\
&\leq \frac{0.01}{n} \cdot n = 0.01,
\end{aligned}$$

where the first inequality follows from property of projection matrix (for any projection matrix P, we have $P \preceq I$. Then we know $x^\top P x \leq x^\top x$ for all vector x), the last inequality follows from $\sum_{i=1}^m \sigma_i = n$.

Third, from δ'_x (Line 11) to $\widetilde{\delta}$ (Line 12), we can tolerate the approximation error on the Newton's direction $\|\widetilde{\delta} - \delta'_x\|_{H(x)} \leq 0.1$. This is crucial for our graph applications, since we need to use small space SDD solver to approximate the Newton's direction.

2.2 Solve LP in small space

In this section, we show how to implement our IPM in space not polynomially dependent on m for different barrier functions.

[7] For a vector y and a positive semidefinite matrix A, we define $\|y\|_A := \sqrt{y^\top A y}$.

Algorithm 1 A simplified version of our algorithm.

procedure OurAlgorithm($A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$)
  Choose $F(x) \in \mathbb{R}^n \to \mathbb{R}$ to be any θ^2-self concordant barrier function
  Let $f_t(x) := t \cdot c + \nabla F(x) \in \mathbb{R}^n$
  Let $H(x) := \nabla^2 F(x) \in \mathbb{R}^{n\times n}$
  Let T be the number of iterations
  Initialize x, t
  for $k \leftarrow 1$ to T **do**
    Let $\widetilde{H}(x)$ be any PSD matrix that $\frac{1}{\log m}\widetilde{H}(x) \preceq H(x) \preceq \widetilde{H}(x)$
    Let $\delta_x := -\widetilde{H}(x)^{-1} \cdot \nabla f_t(x)$
    Let $\widetilde{\nabla} f_t(x)$ be that $\|\widetilde{\nabla} f_t(x) - \nabla f_t(x)\|_{H(x)^{-1}} \leq 0.1$
    Let $\delta'_x := -\widetilde{H}(x)^{-1} \cdot \widetilde{\nabla} f_t(x)$
    Let $\widetilde{\delta}_x$ be any vector that $\|\widetilde{\delta}_x - \delta'_x\|_{H(x)} \leq 0.1$
    $x \leftarrow x + \widetilde{\delta}_x$
    $t \leftarrow t \cdot (1 + \theta^{-1})$
  end for
  Output x
end procedure

For three barriers (logarithmic, hybrid and Lee-Sidford), all of their Hessians take the form of $A^\top HA \in \mathbb{R}^{n\times n}$ for an $m \times m$ non-negative diagonal matrix H. For logarithmic barrier, $H_{i,i} = s_i(x)^{-2}$, as $s_i(x)$ can be computed in $O(1)$ space, it is not hard to see that the Gram matrix can be computed as $\sum_{i\in[m]} H_{i,i} \cdot a_i a_i^\top$ in $O(n^2)$ space.

The more interesting case is to consider the hybrid barrier and Lee-Sidford barrier. The gradient and Hessian of the hybrid barrier requires us to compute m leverage scores defined as $\mathrm{diag}(\sqrt{H}A(A^\top HA)^{-1}A^\top\sqrt{H})$. Forming this projection matrix will require a prohibitive m^2 space. To implement it in n^2 space, we rely on an observation that $\sigma_i = H_{i,i} \cdot a_i^\top (A^\top HA)^{-1} a_i$, thus, if we can manage to maintain $(A^\top HA)^{-1}$ in $O(n^2)$ space, then we can compute the leverage score. Similar to the logarithmic barrier scenario, $A^\top HA$ can be computed in 1 pass and $O(n^2)$ space, then the inverse can be computed in $O(n^2)$ space. Thus, we can supply the i-th leverage score in $O(n^2)$ space, and compute the gradient and Hessian in designated space constraint.

Given an oracle that can compute the i-th leverage score in $O(n^2)$ space, we can even implement the $\ell_{\log m}$ Lewis weights in $\widetilde{O}(n^2)$ space. To do so, we rely on an iterative scheme introduced in [28]. Unfortunately, as we are only allowed a space budget of $O(n^2)$, we cannot store the intermediate Lewis weights. To circumvent this issue, we develop a recursive algorithm to query Lewis weights from prior iterations. Each recursion takes $O(n^2)$ space, and the algorithm uses at most $O(\mathrm{poly}(\log m))$ iterations, therefore, we can compute the Lewis weights in $\widetilde{O}(\sqrt{n})$ space.

2.3 Semi-streaming maximum weight bipartite matching in $\widetilde{O}(\sqrt{m})$ passes

Recall that in the semi-streaming model, we are only allowed with $\widetilde{O}(n)$ space. For the IPMs we've developed before, we can not meet such space constraint. For general graphs, we have to invent more machinery to realize the $\widetilde{O}(n)$ space.

For matching, we start by noting that the constraint matrix $A \in \mathbb{R}^{m \times n}$ is a graph incidence matrix. This means that for logarithmic barrier, the Hessian matrix $A^\top S^{-2} A$ can be treated as a Laplacian matrix with edge weight s_i^{-2}. Therefore, computing the Newton direction reduces to perform an SDD solve in $\widetilde{O}(n)$ space.

SDD solver in the semi-streaming model. Though solving SDD system can be done in an extremely time-efficient manner, it is unclear how to compute them when only $\widetilde{O}(n)$ space is allowed. To circumvent this problem, we rely on two crucial observations. Let L_G denote the SDD matrix corresponding to the Hessian.

- Solving a system $L_G \cdot x = b$ will require $\widetilde{\Omega}(m)$ space, but multiplying L_G with a vector $v \in \mathbb{R}^n$ can be done in $O(n)$ space: as $L_G = \sum_{i \in [m]} \frac{a_i a_i^\top}{s_i^2}$, $L_G \cdot v$ can be computed as $a_i (a_i^\top v)/s_i^2$ in $O(n)$ space, and accumulate the sum over a pass of the graph.
- Suppose we have a sparse graph H with only $\widetilde{O}(n)$ edges, then the system $L_H \cdot x = b$ can be solved in $\widetilde{O}(n)$ space.

It turns out that these two observations are enough for us to solve a general SDD system in $\widetilde{O}(n)$ space. Given the graph G, we first compute a $(1 \pm \delta)$-spectral sparsifier with only $\widetilde{O}(\delta^{-2} n \log^{O(1)} n)$ edges in a single pass [40]. Let H denote this sparsifier, we then use L_H^{-1} as a preconditioner for solving our designated SDD system. More concretely, let $r_t := b - L_G \cdot x_t$ denote the residual at t-th iteration, we solve the system $L_H \cdot y_t = r_t$. As $y_t = L_H^{-1} \cdot b - L_H^{-1} L_G \cdot x_t$, we can then update the solution via the preconditioned-solution $x_{t+1} = x_t + y_t$. The residual is then $r_{t+1} = b - L_G \cdot x_{t+1} = b - L_G \cdot x_t - L_G \cdot y_t = r_t - L_G \cdot y_t$, i.e., we only need to implement one matrix-vector product with L_G. After $\widetilde{O}(1)$ iterations, we have refined an accurate enough solution for the SDD system.

From dual to primal. Though we can solve the dual in $\widetilde{O}(n)$ space, it only produces a solution to the minimum vertex cover and we need to transform it to a solution to maximum weight matching.

Turning an optimal dual solution to an optimal primal solution for general LP requires at least solving a linear system, which takes $O(n^\omega)$ time and $O(mn)$ space (Lee, Sidford and Wong [54]), which is unknown to be implemented in the semi-streaming model even for bipartite matching LP.[8] We bypass this issue by using the complementary slackness theorem to highlight n tight dual constraints and therefore sparsify the original graph from m edges to n edges without losing the optimal matching. However, this is only true if the solution to the primal LP is **unique**.

To give a better illustration, let us consider a simple example. Suppose the graph has a (maximum weight) perfect matching (see Figure 1 for example). Then the following trivial solution is optimal to the dual LP: choosing all vertices in V_L. Let us show what happens when applying the complementary slackness theorem. The complementary slackness theorem says that when y is a feasible primal solution and x is a feasible dual solution, then y is optimal to the primal and x is optimal to the dual **if and only if**

$$\langle y, Ax - \mathbf{1}_m \rangle = 0 \quad \text{and} \quad \langle x, \mathbf{1}_n - A^\top y \rangle = 0. \tag{2}$$

From the above case, we have $Ax - \mathbf{1}_m = 0$, so the first equality $\langle y, Ax - \mathbf{1}_m \rangle = 0$ puts no constraint on y. Therefore, any solution $y \geq \mathbf{0}_m$ to the linear system $a_i^\top y_i = 1,\ \forall i \in V_L$ is an optimal solution, where a_i is the i-th column of A. Note that this linear system has m variables and $|V_L|$ equations, which is still hard to find a solution in $\widetilde{O}(n)$ space.

[8] In general, the inverse of a sparse matrix can be dense, which means the standard Gaussian elimination method for linear system solving does not apply in the semi-streaming model.

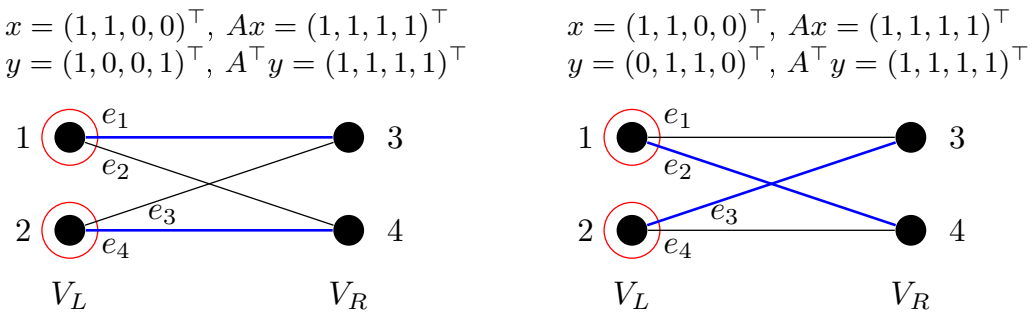

Figure 1 The red circle is a minimum vertex cover, which is an optimal dual solution. The blue edge is a maximum matching, which is an optimal primal solution. In both examples, the primal and dual satisfy complementary slackness Eq. (2).

Now consider perturbing the primal objective function by some vector $b \in \mathbb{R}^m$ such that the optimal solution to the following primal LP is **unique**:

$$\textbf{Primal} \quad \max_{y \in \mathbb{R}^m} b^\top y, \qquad \text{s.t. } A^\top y \leq \mathbf{1}_n \text{ and } y \geq \mathbf{0}_m.$$

Suppose we find the optimal solution x in the dual LP and we want to recover the optimal solution y in the primal LP. Again by plugging in the complementary slackness theorem, we get at most n equations from the second part $\langle x, \mathbf{1}_n - A^\top y \rangle = 0$. Since the optimal y is unique and y has dimension m, the first part $\langle y, Ax - \mathbf{1}_m \rangle$ must contribute to at least $m - n$ equations. Note that these equations have the form

$$y_i = 0, \ \forall i \in [m] \text{ s.t. } (Ax)_i - 1 > 0.$$

This means that the corresponding edges are *unnecessary* in order to get one maximum matching. As a result, we can reduce the number of edges from m to n, then compute a maximum matching in $\widetilde{O}(n)$ space without reading the stream.

Isolation lemma in the semi-streaming model. It remains to show how to perturb the objective so that the primal solution is unique. As the perturbation is over all edges, one natural idea is to randomly perturb them using $\widetilde{O}(m)$ bits of randomness. This becomes troublesome when the random bits need to be stored since the perturbation should remain consistent across different passes. We resolve this problem via the isolation lemma.

Let us recall the definition of the isolation lemma (see Section C in full version [57] for details).

▶ **Definition 4** (Isolation lemma). *Given a set system $(S, \mathcal{F})$ where $\mathcal{F} \subseteq \{0,1\}^S$. Given weight w_i to each element i in S, the weight of a set F in $\mathcal{F}$ is defined as $\sum_{i \in F} w_i$. The isolation lemma says there exists a scheme that can assign weight oblivious to $\mathcal{F}$, such that there is a unique set in $\mathcal{F}$ that has the minimum (maximum) weight under this assignment.*

The isolation lemma says that if we randomly choose weights, then with a good probability the uniqueness is ensured. However, this does not apply to the streaming setting since the weight vector is over all edges, which require $\Omega(m)$ space.

To apply isolation lemma for bipartite matching, we note that the set S is all the edges and the family $\mathcal{F}$ contains all possible matchings. The total number of possible matchings is at most $(n+1)^n$, as each vertex can choose none or one of the vertices to match. We

leverage this parameterization and make use of [17], which requires $\log(|F|)$ random bits. For matching, we only need $O(n \log n)$ bits, which suits in our space budget. To the best of our knowledge, this is the first use of isolation lemma in the streaming model.

2.4 Discussions

For matching, improving $\sqrt{m}$ passes to $\sqrt{n}$ passes will require us to compute fundamental quantities such as leverage scores and Lewis weights by solving $\widetilde{O}(1)$ SDD systems. As reachability [58] and single source shortest path [30, 16] can be solved in $n^{1/2+o(1)}$ passes in the semi-streaming model, we believe it is an important open problem to close the gap between bipartite matching and these problems.

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.

2 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *ICALP*, pages 526–538. Springer, 2011.

3 Kook Jin Ahn and Sudipto Guha. Access to data and number of iterations: Dual primal algorithms for maximum matching under resource constraints. *ACM Transactions on Parallel Computing (TOPC)*, 4(4):1–40, 2018.

4 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *SODA*, 2021.

5 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1616–1635. SIAM, 2019.

6 Sepehr Assadi, Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In *SODA*. arXiv preprint, 2022. `arXiv:2011.03495`.

7 Sepehr Assadi, Nikolai Karpov, and Qin Zhang. Distributed and streaming linear programming in low dimensions. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 236–253, 2019.

8 Sepehr Assadi, Gillat Kol, Raghuvansh R. Saxena, and Huacheng Yu. Multi-pass graph streaming lower bounds for cycle counting, max-cut, matching size, and other problems. In *FOCS*, 2020.

9 Sepehr Assadi, S. Cliff Liu, and Robert E. Tarjan. An auction algorithm for bipartite matching in streaming and massively parallel computation models. In *The SIAM Symposium on Simplicity in Algorithms (SOSA@SODA'21)*, 2021.

10 Sepehr Assadi and Ran Raz. Near-quadratic lower bounds for two-pass graph streaming algorithms. In *FOCS*, 2020.

11 Aaron Bernstein. Improved bounds for matching in random-order streams. In *ICALP*, 2020.

12 Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 259–278. SIAM, 2020.

13 Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *STOC*, 2020.

14 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.

15 Timothy M Chan and Eric Y Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.

16 Yi-Jun Chang, Martin Farach-Colton, Tsan-Sheng Hsu, and Meng-Tsung Tsai. Streaming complexity of spanning tree computation. In *37th international symposium on theoretical aspects of computer science (STACS)*, 2020.

17 Suresh Chari, Pankaj Rohatgi, and Aravind Srinivasan. Randomness-optimal unique element isolation with applications to perfect matching and related problems. *SIAM Journal on Computing*, 24(5):1036–1050, 1995.

18 Kenneth L Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM (JACM)*, 42(2):488–499, 1995.

19 Michael B Cohen, Rasmus Kyng, Gary L Miller, Jakub W Pachocki, Richard Peng, Anup B Rao, and Shen Chen Xu. Solving sdd linear systems in nearly $m \log^{1/2} n$ time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing(STOC)*, pages 343–352, 2014.

20 Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, 2019.

21 Michael B. Cohen and Richard Peng. Lp row sampling by lewis weights. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, 2015.

22 Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC)*, pages 451–460, 2008.

23 George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation*, Cowles Commission Monograph No. 13. ., 1951.

24 Shahar Dobzinski, Noam Nisan, and Sigal Oren. Economic efficiency requires interaction. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 233–242, 2014.

25 Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. In *STOC*, 2021. `arXiv:2011.05365`.

26 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1-2):490–508, 2012.

27 Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mah, Anup Rao, and Ryan A Rossi. Approximate maximum matching in random streams. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1773–1785. SIAM, 2020.

28 Maryam Fazel, Yin Tat Lee, Swati Padmanabhan, and Aaron Sidford. Computing lewis weights to high precision. In *SODA*, 2021.

29 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *ICALP*, pages 531–543. Springer, 2004.

30 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2009.

31 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms(SODA)*, pages 1029–1046, 2018.

32 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 468–485. SIAM, 2012.

33 Yuzhou Gu and Zhao Song. A faster small treewidth sdp solver. *arXiv preprint*, 2022. `arXiv:2211.06033`.

34 Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving tall dense sdps in the current matrix multiplication time. In *FOCS*, 2022.

35 Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. In *STOC*, 2021. arXiv:2004.07470.

36 Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1679–1697. SIAM, 2013.

37 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 734–751. SIAM, 2014.

38 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.

39 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.

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

41 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing(STOC)*, pages 302–311. ACM, 1984.

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

43 Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing (STOC)*, pages 911–920, 2013.

44 Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

45 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving sdd linear systems. In *FOCS*, pages 235–244, 2010.

46 Ioannis Koutis, Gary L Miller, and Richard Peng. A nearly-$m \log n$ time solver for sdd linear systems. In *52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 590–598, 2011.

47 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582. IEEE, 2016.

48 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation (ISSAC)*, pages 296–303. ACM, 2014.

49 Yin Tat Lee and Aaron Sidford. Path finding i: Solving linear programs with $\widetilde{O}(\sqrt{rank})$ linear system solves. *arXiv preprint*, 2013. arXiv:1312.6677.

50 Yin Tat Lee and Aaron Sidford. Path finding ii: An $\widetilde{O}(m\sqrt{n})$ algorithm for the minimum cost flow problem. *arXiv preprint*, 2013. arXiv:1312.6713.

51 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.

52 Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 230–249. IEEE, 2015.

53 Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt (rank) linear system solves. *arXiv preprint*, 2019. arXiv:1910.08033.

54 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2015.

55 Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*, 2019.

56 D. Lewis. Finite dimensional subspaces of l_p. *Studia Mathematica*, 1978.

57 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. *arXiv*, 2020. `arXiv:2009.06106`.

58 Yang P Liu, Arun Jambulapati, and Aaron Sidford. Parallel reachability in almost linear work and square root depth. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1664–1686. IEEE, 2019.

59 Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181. Springer, 2005.

60 Ketan Mulmuley, Umesh V Vazirani, and Vijay V Vazirani. Matching is as easy as matrix inversion. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing (STOC)*, pages 345–354, 1987.

61 Yu Nesterov and Arkadi Nemirovsky. Self-concordant functions and polynomial-time methods in convex programming. *Report, Central Economic and Mathematic Institute, USSR Acad. Sci*, 1989.

62 Ami Paz and Gregory Schwartzman. A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2153–2161, 2017.

63 Richard Peng and Daniel A Spielman. An efficient parallel solver for sdd linear systems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing (STOC)*, pages 333–342, 2014.

64 James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical Programming*, 40(1-3):59–93, 1988.

65 Zhao Song and Zheng Yu. Oblivious sketching-based central path method for solving linear programming. In *ICML*, 2021.

66 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90. `arXiv:cs/0310051`, divided into `arXiv:0809.3232`, `arXiv:0808.4134`, `arXiv:cs/0607105`, 2004.

67 Daniel A. Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM J. Matrix Analysis Applications*, 35(3):835–885, 2014.

68 Pravin M Vaidya. An algorithm for linear programming which requires $O(((m+n)n^2+(m+n)^{1.5}n)L)$arithmetic operations. In *FOCS*. IEEE, 1987.

69 Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *FOCS*. IEEE, 1989.

70 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 887–898. ACM, 2012.

71 Guanghao Ye. *Fast Algorithm for Solving Structured Convex Programs*. Bachelor's thesis, University of Washington, 2021.