2014 IEEE Annual Symposium on Foundations of Computer Science

Path-Finding Methods for Linear Programming

Solving Linear Programs in $\tilde{O}(\sqrt{rank})$ Iterations and Faster Algorithms for Maximum Flow

Yin Tat Lee
Department of Mathematics
MIT
Cambridge, USA
Email: yintat@mit.edu

Aaron Sidford
Department of EECS
MIT
Cambridge, USA
Email: sidford@mit.edu

***Abstract*—In this paper, we present a new algorithm for solving linear programs that requires only $\tilde{O}(\sqrt{\mathbf{rank}(\mathbf{A})}L)$ iterations where A is the constraint matrix of a linear program with m constraints, n variables, and bit complexity L. Each iteration of our method consists of solving $\tilde{O}(1)$ linear systems and additional nearly linear time computation. Our method improves upon the previous best iteration bounds by factor of $\tilde{\Omega}((m/\mathbf{rank}\,(\mathbf{A}))^{1/4})$ for methods with polynomial time computable iterations and by $\tilde{\Omega}((m/\mathbf{rank}\,(\mathbf{A}))^{1/2})$ for methods which solve at most $\tilde{O}(1)$ linear systems in each iteration each achieved over 20 years ago.**

Applying our techniques to the linear program formulation of maximum flow yields an $\tilde{O}(|E|\sqrt{|V|}\log^2 U)$ time algorithm for solving the maximum flow problem on directed graphs with $|E|$ edges, $|V|$ vertices, and capacity ratio U. This improves upon the previous fastest running time of $O(|E|\min\{|E|^{1/2}, |V|^{2/3}\}\log\left(|V|^2/|E|\right)\log(U))$ achieved over 15 years ago by Goldberg and Rao and improves upon the previous best running times for solving dense directed unit capacity graphs of $O(|E|\min\{|E|^{1/2}, |V|^{2/3}\})$ achieved by Even and Tarjan over 35 years ago and a running time of $\tilde{O}(|E|^{10/7})$ achieved recently by Mądry.

***Keywords*-linear program; maximum flow; interior point;**

I. Introduction

Given a matrix, $\mathbf{A} \in \mathbb{R}^{m\times n}$, and vectors, $\vec{b} \in \mathbb{R}^m$ and $\vec{c} \in \mathbb{R}^n$, solving a linear program[1]

$$\min_{\vec{x}\in\mathbb{R}^n \,:\, \mathbf{A}\vec{x}\geq\vec{b}} \vec{c}^T\vec{x} \tag{1}$$

is a core algorithmic task in both the theory and practice of computer science.

Since Karmarkar's breakthrough result in 1984, proving that interior point methods can solve linear programs in polynomial time, interior point methods have been an incredibly active area of research [30]. Now, the fastest asymptotic running times for solving (1) in many regimes are interior point methods. State of the art interior point methods require either $O(\sqrt{m}L)$ iterations of solving linear systems [35] or $O((m\,\mathrm{rank}(\mathbf{A}))^{1/4}L)$ iterations of a more expensive polynomial-time operation [42], [44], [46], [2].

However, in a breakthrough result of Nesterov and Nemirovski in 1994, they showed that there exists a *universal barrier* function that if computable would allow (1) to be solved in $O(\sqrt{\mathrm{rank}(\mathbf{A})}L)$ iterations [29]. Unfortunately, this barrier is more difficult to compute than the solutions to (1). Despite this existential result, the $O((m\,\mathrm{rank}(\mathbf{A}))^{1/4}L)$ iteration bound for polynomial time linear programming methods has not been improved in over 20 years.

In this paper we present a new interior point method that solves general linear programs in $\tilde{O}(\sqrt{\mathrm{rank}(\mathbf{A})}L)$ iterations thereby matching the theoretical limit proved by Nesterov and Nemirovski up to polylogarithmic factors.[2] Furthermore, we show how to achieve this convergence rate while only solving $\tilde{O}(1)$ linear systems and performing additional $\tilde{O}(\mathrm{nnz}(\mathbf{A}))$ work in each iteration. [3]

A. The Maximum Flow Problem

Applying our techniques to the linear programming formulation of the maximum flow problem we achieve a running time of $\tilde{O}(|E|\sqrt{|V|}\log^2(U))$ for solving the problem on a graph with $|E|$ edges, $|V|$ vertices, and capacity ratio U. This improves upon the previous fastest running time of $\tilde{O}(|E|\min\{|E|^{1/2}, |V|^{2/3}\}\log(U))$ achieved over 15 years ago by Goldberg and Rao [11][4] and the previous fastest running times for solving dense directed unit capacity graphs of $O(|E|\min\{|E|^{1/2}, |V|^{2/3}\})$ achieved by Even and Tarjan [7] over 35 years ago and of $\tilde{O}(|E|^{10/7})$ achieved recently by Mądry [25].

Interestingly, for this linear program our interior point method converges provably better than the rate predicted by Nesterov and Nemirovski's existential result. To the best of the author's knowledge this is the first general convergence rate of interior point methods of this kind.

[1] This is the dual of a linear program written in standard form and it encompasses all linear programs. As papers vary in their notation for m and n differs, we state our results in terms of $\sqrt{\mathrm{rank}(\mathbf{A})}$.

[2] We use $\tilde{O}(\cdot)$ to hide $\mathrm{polylog}(n, m, |V|, |E|)$ factors.

[3] Throughout this paper L denotes the standard "bit complexity" of the linear program, a quantity less than the number of bits needed to represent (1). For integral $\mathbf{A}$, $\vec{b}$, and $\vec{c}$ this quantity is often defined as $L \stackrel{\text{def}}{=} \log(1+d_{max})+\log(1+\max\{\|\vec{c}\|_\infty, \|\vec{b}\|_\infty\})$ where d_{max} is the largest absolute value of the determinant of a square sub-matrix of $\mathbf{A}$ [16].

[4] In this paper we are primarily concerned with "weakly" polynomial time algorithms. The current fastest "strongly" polynomial running time for solving this problem is $O(nm)$ [33].

0272-5428/14 $31.00 © 2014 IEEE
DOI 10.1109/FOCS.2014.52

B. Algorithm Running Times

Using different linear system solvers and considering the linear program formulations of flow problems study by Daitch and Spielman [5] we obtain the following:

- We solve (1) in

$$\tilde{O}(\sqrt{\text{rank}(\mathbf{A})}(\text{nnz}(\mathbf{A}) + (\text{rank}(\mathbf{A}))^{\omega})L)$$

  improving upon $\tilde{O}(\sqrt{m}(\text{nnz}(\mathbf{A}) + (\text{rank}(\mathbf{A}))^{\omega})L)$ [35], [27], [22].[5]
- We solve (1) in $\tilde{O}\left(n^{5/2-3(\omega-2)}m^{3(\omega-2)}L\right)$ improving upon the previous best of $\tilde{O}\left(m^{1.5}nL\right)$ [43].
- We achieve the first $O(\sqrt{\text{rank}(\mathbf{A})}L)$ depth polynomial work method for solving linear programs.
- We solve the minimum cost flow problem in $\tilde{O}(|E|\sqrt{|V|}\log^2(U))$ time improving upon the previous best of $\tilde{O}(|E|^{3/2}\log^2(U))$ [5].
- We produce ϵ-approximate solutions to the generalized lossy flow problem in $\tilde{O}(|E|\sqrt{|V|}\log^2(U/\epsilon))$ improving upon the previous best of $\tilde{O}(|E|^{3/2}\log^2(U/\epsilon))$ [5].
- We parallelize the flow algorithm so the work is the same but the depth is $\tilde{O}(\sqrt{|V|}\log^2 U)$.

C. Path Finding

We achieve our results by extending standard path following techniques for linear programming [35], [13]. Whereas standard path following techniques follow a fixed path through the interior of a polytope our algorithms iteratively find better paths through a broad class paths we call the *weighted central paths*. The idea of using the weights to modify the central path is an old one [8] and it was used ingeniously a recent breakthrough by Aleksander Mądry to solve unit capacity directed maximum flow instances in $\tilde{O}(|E|^{10/7})$ [25]. Our algorithm leverages the common insight that by finding better paths the convergence rate of interior point methods can be improved, but performs this path changing differently and exploits different benefits that can come from path finding (building off the work in [42], [44], [46], [2]). We hope that our analysis of weighting the central path may be of independent interest and help further improve the running time of interior point methods.

D. Paper Organization

Our paper is split into two parts, "Path Finding I : Solving Linear Programs with $\tilde{O}(\sqrt{\text{rank}})$ Linear System Solves" [20] and ""Path Finding II : An $\tilde{O}(m\sqrt{n})$ Algorithm for the Minimum Cost Flow Problem" [21] which can be found on arXiv. This paper is a short primer intended to guide the read through the longer documents available online.

[5]For simplicity, the linear program running times presented throughout the paper hide additional dependencies on L that may arise from the need to carry out arithmetic operations to precision L.

II. Overview of Our Approach

We achieve our results by carefully and systematically generalizing *path following* interior point methods.

A. Path Following

To solve (1) interior point methods maintain a point $\vec{x} \in \mathbb{R}^n$ in the interior of the feasible region, denoted $S^0 \stackrel{\text{def}}{=} \{\vec{x} \in \mathbb{R}^n : \mathbf{A}\vec{x} > \vec{b}\}$, and attempt to iteratively decrease the cost of $\vec{x}$, i.e. $\vec{c}^T\vec{x}$, while maintaining feasibility, i.e. $\mathbf{A}\vec{x} \geq \vec{b}$. Distance from in-feasibility is typically measured as a function of the slack vector, $\vec{s}(\vec{x}) \stackrel{\text{def}}{=} \mathbf{A}\vec{x} - \vec{b}$. Path following methods fix a set of tradeoffs between cost and distance from boundary of the feasible region through a penalized objective function $f_t(\vec{x})$. Formally, they solve

$$\min_{\vec{x}\in\mathbb{R}^n} f_t(\vec{x}) \quad \text{where} \quad f_t(\vec{x}) \stackrel{\text{def}}{=} t \cdot \vec{c}^T\vec{x} + \phi(\vec{s}(\vec{x}))$$

where t is a parameter and $\phi : \mathbb{R}^n \to \mathbb{R}$ is a *barrier function* such that $\phi(\vec{s}(\vec{x})) \to \infty$ as $s(\vec{x})_i \to 0$, i.e. x tends to boundary of the feasible region.

Under mild assumptions on ϕ solving (1) is equivalent to minimizing $f_t(\vec{x})$ for a sufficiently large t. Leveraging this insight, path following methods perform a variety of standard reductions [1] to reduce solving (1) to approximately computing $\vec{x}_t \stackrel{\text{def}}{=} \arg\min_{\vec{x}\in\mathbb{R}^n} f_t(\vec{x})$ given an approximate $\vec{x}_{t'}$ for t' multiplicatively close to t. The $\vec{x}_t$ form a a path, called the *central path*, from some analytical notion of center of the feasible region (at $t = 0$) to the solution of (1) (at $t = \infty$). Path following methods then typically use some sort of modification of Newton's method, i.e. they solve linear systems corresponding to optimizing second order approximations to $f_t(\vec{x})$, to follow the central path.

B. Path Finding

To solve (1) ideally we would just produce a barrier function ϕ such that standard path following methods provably converge in $\tilde{O}(\sqrt{\text{rank}(\mathbf{A})}L)$ iterations and each iteration consists of solving $\tilde{O}(1)$ linear systems. Unfortunately, we are unaware of such a barrier function and for the linear program formulations of maximum flow we consider we know none exists (See Section VIII).

Instead, we manipulate the most common choice of barrier, the standard logarithmic barrier, $\phi(\vec{x}) = -\sum_{i\in[m]}\log(s(\vec{x})_i)$. Note that the behavior of the logarithmic barrier is highly dependent on the representation of (1). Just duplicating a constraint, i.e. repeating a row of $\mathbf{A}$ and the corresponding entry in $\vec{b}$, corresponds to doubling the contribution of some log barrier terms $-\log(s(\vec{x})_i)$ to ϕ. It is not hard to see that repeating a constraint many times can actually slow down the convergence of standard path following methods. In other words, path following methods are not invariant to the representation of the feasible region.

This insight motivates us to consider adding weights to the log barrier that we change during the course of the algorithm.

Every weighting of the log barrier induces a different central path for a new representation of the polytope. Rather than following one fixed initial central path we show how we can simultaneously follow a path and find new paths with nicer local properties.

C. What is a good path?

In Section IV, we make this weighting procedure precise and study the *weighted log barrier function* given by

$$\phi(\vec{x}) = -\sum_{i\in[m]} g_i(s(\vec{x})_i)\cdot\log(s(\vec{x})_i)$$

where $\vec{g}:\mathbb{R}^m_{>0}\to\mathbb{R}^m_{>0}$ is a *weight function* of the current point. We provide a general *weighted path following scheme* and analyze how the running time and convergence rate of this scheme in terms of a set of properties of $\vec{g}$.

In Section V we then connect these properties of the weight function to local geometric properties of the polytope. In particular we show solving an optimization problem corresponding to an approximate John Ellipsoid yields weights that induce a $\tilde{O}(\sqrt{\operatorname{rank}(\mathbf{A})}L)$ iteration interior point method.

D. How well can we find the path?

In Section V we show that by a combination of gradient descent and a Johnson Lindenstrauss based method used frequently in numerical linear algebra [26] we can compute multiplicative approximations to the weight function by solving $\tilde{O}(1)$ linear systems. Unfortunately, naive analysis of our weighted path following analysis requires more precise estimates of the weights.

In Section VI we show how our weighted path following scheme can be modified to work even if only given multiplicative approximations to the weights. We phrase the problem as a two player game which we solve using careful regularization.

E. Faster Running Times

In Section VII we show how by using different algorithms for solving the linear systems our linear programming algorithm improves upon current state of the art running times for solving (1) in various regimes.

Finally, In Section VIII we discuss how our techniques can be applied to the linear program formulation of maximum flow. Unfortunately, all known attempts to write maximum flow in the form of (1) make $\operatorname{rank}(\mathbf{A}) = \Omega(|E|)$ and Nesterov and Nemirovski's results [30] show that for natural formulations of flow problems suggested by Daitch and Spielman [5] there is no $\tilde{O}(|V|)$ self-concordant barrier function. Nevertheless, we introduce new tools to solve these flow problems in $\tilde{O}(\sqrt{|V|})$ iterations and applying further analysis from Daitch and Spielman [5] as well as fast Laplacian system solvers [38] we achieve our desired running times.

III. Previous Work

Linear programming and maximum flow are extremely well studied problems with long histories. Here we focus on recent history that particularly influenced our work.

A. Convergence Rate of Interior Point Methods

In 1984, Karmarkar [16] provided the first proof of an interior point method running in polynomial time. This method required $O(mL)$ iterations where the running time of each iteration was dominated by the time needed to solve a linear system of the form $\left(\mathbf{A}^T\mathbf{D}\mathbf{A}\right)\vec{x} = \vec{y}$ for some positive diagonal matrix $\mathbf{D}\in\mathbb{R}^{m\times m}$ and some $\vec{y}\in\mathbb{R}^n$. Using low rank matrix updates and preconditioning Karmarkar achieved a running time of $O(m^{3.5}L)$.

In 1988, Renegar improved the convergence rate of interior point methods to $O\left(\sqrt{m}L\right)$ iterations [35]. He presented a path following method where the iterations had comparable complexity to Karmarkar's method. Using a combination of techniques involving low rank updates, preconditioning and fast matrix multiplication, the amortized complexity of each iteration was improved [41], [13], [30] yielding the current state of the art running time of $O(m^{1.5}nL)$ [43].

In 1989, Vaidya [46] proposed two barrier functions which were shown to yield $O((m\operatorname{rank}(\mathbf{A}))^{1/4}L)$ and $O(\operatorname{rank}(\mathbf{A})L)$ iteration linear programming algorithms [44], [46], [42]. Unfortunately each iteration of these methods required explicit computation of the projection matrix $\mathbf{D}^{1/2}\mathbf{A}(\mathbf{A}^T\mathbf{D}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{D}^{1/2}$ for a positive diagonal matrix $\mathbf{D}\in\mathbb{R}^{m\times m}$. This was improved by Anstreicher [2] who showed it sufficed to compute the diagonal of this projection matrix. Unfortunately both methods do not yield faster running times unless $m\gg n$.

In 1994 Nesterov and Nemirovski [30] showed that path-following methods can in principle be applied to any solve convex optimization problem, given a suitable barrier function. The number of iterations of their method depends on a square root of a quantity known as the *self-concordance* of the barrier. They showed that for any convex set in $\mathbb{R}^n$, there exists a barrier function, called the *universal barrier* function, with self-concordance $O(n)$ and therefore in principle any convex optimization problem with n variables can be solved in $O\left(\sqrt{n}L\right)$ iterations. However, this result is generally considered to be only of theoretical interest as the universal barrier function is defined as the volume of certain polytopes, a problem which in full-genearality is NP-hard and can only be computed approximately in $O(n^c)$ for some large constant c [23].

These results suggested that you can solve linear programs closer to the $\tilde{O}(\sqrt{\operatorname{rank}(\mathbf{A})}L)$ bound achieved by the universal barrier only if you pay more in each iteration. In this paper we show that this is not the case and up to polylogarithmic factors we achieve the convergence rate of the universal barrier function while only having iterations of cost comparable to that of Renegar's algorithm.

Year	Author	Number of Iterations	Nature of iterations
1984	Karmarkar [16]	$O(mL)$	Linear system solve
1986	Renegar [35]	$O(\sqrt{m}L)$	Linear system solve
1989	Vaidya [45]	$O((m\,\mathrm{rank}(\mathbf{A}))^{1/4}\,L)$	Expensive linear algebra
1994	Nesterov and Nemirovskii [30]	$O(\sqrt{\mathrm{rank}(\mathbf{A})}L)$	Volume computation
2013	This paper	$\tilde{O}(\sqrt{\mathrm{rank}(\mathbf{A})}L)$	$\tilde{O}(1)$ Linear system solves

Figure 1. Interior Point Iteration Improvements

B. Recent Breakthroughs on the Maximum Flow Problem

A beautiful line of work on solving the maximum flow on undirected graphs began with a result of Benzcur and Karger in which they showed how to reduce approximately computing minimum st cuts on arbitrary undirected graphs to the same problem on sparse graphs, i.e. those with $|E| = \tilde{O}(|V|)$ [3]. Pushing these ideas further Karger showed how to perform a similar reduction for the maximum flow problem [15] and Karger and Levine showed how to compute the exact maximum flow in an unweighted undirected graph with maximum flow value F in $\tilde{O}(|E| + |V|F)$ time [14].

In a breakthrough result of Spielman and Teng [38] they showed that a particular class of linear systems, Laplacians, can be solved in nearly linear time. Leveraging these fast Laplacian system solvers Christiano, Kelner, Mądry, and Spielman [4] showed how to compute $(1-\epsilon)$ approximations to the maximum flow problem on undirected graphs in $\tilde{O}(|E| \cdot |V|^{1/3}\epsilon^{-11/3})$. Later Lee, Rao and Srivastava [19] improved the running time to $\tilde{O}(|E| \cdot |V|^{1/3}\epsilon^{-2/3})$ for unweighted undirected graphs. This line of work culminated in recent results of Sherman [36] and Kelner, Lee, Orecchia, and Sidford [17] who showed how to solve the problem in $O(|E|^{1+o(1)}\epsilon^{-2})$, using congestion-approximators, oblivious routings, and techniques developed by Mądry [24].

In 2008, Daitch and Spielman [5] showed that, by careful application of interior point techniques, fast Laplacian system solvers [38], and a novel method for solving M-matrices, they could match (up to polylogarithmic factors) the running time of Goldberg Rao and achieve a running time of $\tilde{O}(|E|^{3/2}\log^2(U))$ not just for maximum flow but also for the minimum cost flow and lossy generalized minimum cost flow problems (see Fig III-B).

Very recently Mądry [25] achieved an astounding running time of $\tilde{O}(|E|^{10/7})$ for solving the maximum flow problem on un-capacitated directed graphs by a novel application and modification of interior point methods. This shattered numerous barriers providing the first general improvement over the running time of $O(|E|\min\{|E|^{1/2}, |V|^{2/3}\})$ for solving unit capacity graphs proven over 35 years ago by Even and Tarjan [7] in 1975.

While our algorithm for solving the maximum flow problem is new, we make extensive use of these breakthroughs. We use sampling techniques first discovered in the context of graph sparsification [37] to re-weight the graph so that we make progress at a rate commensurate with the number of vertices and not the number of edges. We use fast Laplacian system solvers as in [4], [19] to make the cost of interior point iterations cheap when applied to the linear program formulations analyzed by Daitch and Spielman [5]. Furthermore, as in Mądry [25] we use weights to change the central path (albeit for a slightly different purpose). We believe this further emphasizes the power of these tools as general purpose techniques for algorithm design.

Year	Author	Running Time
1972	Edmonds, Karp [6]	$\tilde{O}(\|E\|^2\log(U))$
1984	Tardos [40]	$O(\|E\|^4)$
1984	Orlin [31]	$\tilde{O}(\|E\|^3)$
1986	Galil, Tardos [10]	$\tilde{O}\left(\|E\|\|V\|^2\right)$
1987	Goldberg, Tarjan [12]	$\tilde{O}(\|E\|\|V\|\log(U))$
1988	Orlin [32]	$\tilde{O}(\|E\|^2)$
2008	Daitch, Spielman [5]	$\tilde{O}(\|E\|^{3/2}\log^2(U))$
2013	This paper	$\tilde{O}(\|E\|\sqrt{\|V\|}\log^2(U))$

Figure 2. Minimum Cost Flow Running Times Improvements

IV. Path Finding : Properties of the Weighted Central Path

Here we present our path finding scheme assuming we can always compute weights with suitable properties. In addition to maintaining a feasible point $\vec{x} \in S^0$ and a path parameter t we maintain a set of positive weights $\vec{w} \in \mathbb{R}^m_{>0}$ and attempt to minimize the *weighted penalized objective function* $f_t : S^0 \times \mathbb{R}^m_{>0} \to \mathbb{R}$ given for all $\vec{x} \in S^0$ and $\vec{w} \in \mathbb{R}^m_{>0}$ by

$$f_t(\vec{x}, \vec{w}) \stackrel{\text{def}}{=} t \cdot \vec{c}^T\vec{x} - \sum_{i\in[m]} w_i \log s(\vec{x})_i. \tag{2}$$

We call $(\vec{x}, \vec{w}) \in \{\mathbb{R}^m \times \mathbb{R}^m\}$ feasible if $\vec{x} \in S^0$ and $\vec{w} \in \mathbb{R}^m_{>0}$ and our goal is to compute a sequence of feasible points for increasing t and changing (but bounded) $\vec{w}$ such that $f_t(\vec{x}, \vec{w})$ is nearly minimized with respect to $\vec{x}$. To do this we alternate between increasing t, taking a step to improve the current points distance to the weighted central path, and changing the weights.

A. Measuring Centrality

As with all path following methods we use a variant of Newton's method to improve centrality. Standard path following methods simply minimize the second order approximation of $f_t(\vec{x},\vec{w})$ in terms of $\vec{x}$, i.e. set $\vec{x}_{(new)} := \vec{x} - \vec{h}_t(\vec{x},\vec{w})$, where the Newton step, $\vec{h}_t(\vec{x},\vec{w})$, is

$$\begin{aligned}\vec{h}_t(\vec{x},\vec{w}) &\stackrel{\text{def}}{=} (\nabla^2_{\vec{x}\vec{x}} f_t(\vec{x},\vec{w}))^{-1}\nabla_{\vec{x}} f_t(\vec{x},\vec{w}) \\ &= (\mathbf{A}^T\mathbf{S}_x^{-1}\mathbf{W}\mathbf{S}_x^{-1}\mathbf{A})^{-1}(t\vec{c} - \mathbf{A}^T\mathbf{S}_x^{-1}\vec{w}) \qquad (3)\end{aligned}$$

We measure the proximity of $(\vec{x},\vec{w})$ to the weighted central path, what we call *centrality*, $\delta_t(\vec{x},\vec{w})$, in the standard way, by the size of the Newton step in the Hessian norm

$$\begin{aligned}\delta_t(\vec{x},\vec{w}) &\stackrel{\text{def}}{=} \left\|\vec{h}_t(\vec{x},\vec{w})\right\|_{\nabla^2_{\vec{x}\vec{x}} f_t(\vec{x},\vec{w})} \\ &= \left\|t\vec{c} - \mathbf{A}^T\mathbf{S}_x^{-1}\vec{w}\right\|_{\left(\mathbf{A}^T\mathbf{S}_x^{-1}\mathbf{W}\mathbf{S}_x^{-1}\mathbf{A}\right)^{-1}}. \qquad (4)\end{aligned}$$

where $\mathbf{S}_x$, $\mathbf{W}$ are diagonal matrices with diagonals $\vec{s}(\vec{x})$, $\vec{w}$.

B. Advancing Down the Path

The rate at which increasing t hurts centrality is easily shown to be governed by the current centrality and $\left\|\vec{w}\right\|_1$.

Lemma 1. *For feasible* $(\vec{x},\vec{w})$ *and* $\alpha, t \geq 0$

$$\delta_{(1+\alpha)t}(\vec{x},\vec{w}) \leq (1+\alpha)\delta_t(\vec{x},\vec{w}) + \alpha\sqrt{\left\|\vec{w}\right\|_1}$$

How much Newton's method can decrease centrality is is governed by how much the Hessian of f_t changes. We bound this change by bounding what we call the *slack sensitivity*

Definition 2. For $\vec{s},\vec{w} \in \mathbb{R}^m_{>0}$ the *slack sensitivity* is

$$\gamma(\vec{s},\vec{w}) \stackrel{\text{def}}{=} \max_{i\in[m]} \left\|\mathbf{W}^{-1/2}\vec{\mathbb{1}}_i\right\|_{\mathbf{P}_{\mathbf{S}_x^{-1}\mathbf{A}}(\vec{w})}$$

where $\mathbf{P}_{\mathbf{S}_x^{-1}\mathbf{A}}(\vec{w})$ is the projection matrix given by

$$\mathbf{P}_{\mathbf{S}_x^{-1}\mathbf{A}}(\vec{w}) \stackrel{\text{def}}{=} \mathbf{W}^{1/2}\mathbf{S}_x^{-1}\mathbf{A}\left(\mathbf{A}^T\mathbf{S}_x^{-1}\mathbf{W}\mathbf{S}_x^{-1}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{S}_x^{-1}\mathbf{W}^{1/2}$$

Deviating from standard path following analysis we show that the Newton step can split into a step on $\vec{x}$ and a step on $\vec{w}$ and still make large centering progress

Lemma 3 (Split Newton Step)**.** *For feasible* $\{\vec{x}^{(old)},\vec{w}^{(old)}\}$ *let*

$$\begin{aligned}\vec{x}^{(new)} &\stackrel{\text{def}}{=} \vec{x}^{(old)} - \frac{1}{1+r}\vec{h}_t(\vec{x}^{(old)},\vec{w}^{(old)}), \\ \vec{w}^{(new)} &\stackrel{\text{def}}{=} \vec{w}^{(old)} + \frac{r}{1+r}\mathbf{W}_{(old)}\mathbf{S}^{-1}_{(old)}\mathbf{A}\vec{h}_t(\vec{x}^{(old)},\vec{w}^{(old)})\end{aligned}$$

If $\delta_t \stackrel{\text{def}}{=} \delta_t(\vec{x}^{(old)},\vec{w}^{(old)}) \leq \frac{1}{8\gamma(\vec{x}^{(old)},\vec{w}^{(old)})}$*, Then*

$$\delta_t(\vec{x}^{(new)},\vec{w}^{(new)}) \leq \frac{2}{1+r}\cdot\gamma(\vec{x}^{(old)},\vec{w}^{(old)})\cdot\delta_t^2.$$

C. The Weight Function : Changing Weights

Lemmas 1 and 3 suggest exactly what properties we want of our weight function. By Lemma 3 if we maintain that $\delta_t(\vec{x},\vec{w}) \leq O(\frac{1}{\gamma(\vec{x},\vec{w})})$ then we decrease δ_t rapidly with split steps. Furthermore, for such approximately centered points by Lemma 1 we could increase t by a multiplicative $(1+O((\gamma(\vec{x},\vec{w})\sqrt{\left\|\vec{w}\right\|_1})^{-1})$ and maintain an approximately centered point. This suggesst we could double t while maintaining approximate centrality with $O(\gamma(\vec{x},\vec{w})\sqrt{\left\|\vec{w}\right\|_1})$ split steps.

Thus we want $\gamma(\vec{x},\vec{w})\sqrt{\left\|\vec{w}\right\|_1} = \tilde{O}(\sqrt{\text{rank}(\mathbf{A})})$. To maintain this we assume we have access to some fixed differentiable *weight function* $\vec{g} : \mathbb{R}^m_{>0} \to \mathbb{R}^m_{>0}$ from slacks to positive weights that we will use to pick $\vec{w}$. For $\vec{g}$ to be useful we need to show that when we set the weights to $\vec{g}(\vec{s}(\vec{x}))$ after a split step, this does not hurt centrality too much. Letting $\mathbf{G}(\vec{s}) \stackrel{\text{def}}{=} \mathbf{diag}(\vec{g}(\vec{s}))$ denote the diagonal matrix associated with the slacks and we let $\mathbf{G}'(\vec{s}) \stackrel{\text{def}}{=} \mathbf{J}_{\vec{s}}(\vec{g}(\vec{s}))$ denote the Jacobian of the weight function with respect to the slacks we bound this by bounding the operator norm of $\mathbf{I} + r^{-1}\mathbf{G}(\vec{s})^{-1}\mathbf{G}'(\vec{s})\mathbf{S}$. Lastly, we make a normalizing uniformity assumption that none of the weights are two big.

Definition 4 (Weight Function)**.** A *weight function* is a differentiable function from $\vec{g} : \mathbb{R}^m_{>0} \to \mathbb{R}^m_{>0}$ such that for constants $c_1(\vec{g})$, $c_\gamma(\vec{g})$, and $c_r(\vec{g})$, we have the following for all $\vec{s} \in \mathbb{R}^m_{>0}$:

- *Size* : The size $c_1(\vec{g}) = \left\|\vec{g}(\vec{s})\right\|_1$.
- *Slack Sensitivity*: The slack sensitivity $c_\gamma(\vec{g})$ satisfies $c_\gamma(\vec{g}) \geq 1$ and $\gamma(\vec{s},\vec{g}(\vec{s})) \leq c_\gamma(\vec{g})$.
- *Step Consistency* : The step consistency $c_r(\vec{g})$ satisfies $c_r(\vec{g}) \geq 1$ and $\forall r \geq c_r(\vec{g})$ and $\forall \vec{y} \in \mathbb{R}^m$
  - $\left\|\mathbf{I} + r^{-1}\mathbf{G}(\vec{s})^{-1}\mathbf{G}'(\vec{s})\mathbf{S}\right\|_{\mathbf{G}(\vec{s})} \leq 1$
  - $\left\|\vec{y} + r^{-1}\mathbf{G}(\vec{s})^{-1}\mathbf{G}'(\vec{s})\mathbf{S}\vec{y}\right\|_\infty \leq \left\|\vec{y}\right\|_\infty + c_r\left\|\vec{y}\right\|_{\mathbf{G}(\vec{s})}$
- *Uniformity* : $\left\|\vec{g}(\vec{s})\right\|_\infty \leq 2$.

Using the consistency property and integrating we can bound how much centrality is hurt when we take a split step and then update the weights using the weight function.

Theorem 5 (Centering with Weights)**.** *Let* $\vec{x}^{(old)} \in S^0$ *and*

$$\vec{x}^{(new)} = \vec{x}^{(old)} - \frac{1}{1+c_r}\vec{h}(\vec{x}^{(old)},\vec{g}(\vec{s}^{(old)})). \qquad (5)$$

If $\delta_t(\vec{x}^{(old)},\vec{g}(\vec{s}^{(old)})) \leq \frac{1}{100c_\gamma c_r^2}$ *then*

$$\delta_t(\vec{x}^{(new)},\vec{g}(\vec{s}^{(new)})) \leq \left(1 - \frac{1}{4c_r}\right)\delta_t(\vec{x}^{(old)},\vec{g}(\vec{s}^{(old)})).$$

Combining Theorem 5 and Lemma 1 we see that we can double t while maintaining $\delta_t(\vec{x},\vec{g}(\vec{s})) = O(c_\gamma^{-1}c_r^{-2})$ by a total of $O(c_\gamma^{-1}c_r^{-3}c_1^{-1/2})$ steps of (5) and computing $\vec{g}(\vec{x})$. It simply remains to show $c_\gamma^{-1}c_r^{-3}c_1^{-1/2} = \tilde{O}(\sqrt{\text{rank}(\mathbf{A})})$ and shows that computing the weights approximately suffices.

V. The Weight Function

Here, we present the weight function $\vec{g} : \mathbb{R}^m_{>0} \to \mathbb{R}^m_{>0}$ that when used in the framework proposed in the previous section yields an $\tilde{O}(\sqrt{\text{rank}(\mathbf{A})}L)$ iteration interior point method.

A. The Weight Function

Our weight function was inspired by the volumetric barrier methods of [45], [2]. These papers considered using the *volumetric barrier*, $\phi(\vec{s}) = -\log\det(\mathbf{A}^T\mathbf{S}^{-2}\mathbf{A})$, in addition to the standard log barrier, $\phi(\vec{s}) = -\sum_{i\in[m]}\log s_i$. If viewed as a weight function the standard log barrier has a good slack sensitivity, 1, but a large size, m, and the volumetric barrier has a worse slack sensitivity, $\sqrt{m}$, but better total weight, n. By carefully applying a weighted combination of these two barriers [45] and [2] achieved an $O((m\,\text{rank}(\mathbf{A}))^{1/4}L)$ iteration interior point method at the cost more expensive linear algebra in each iteration.

Instead of using a fixed barrier, our weight function $\vec{g}$ is computed by solving a convex optimization problem whose optimality conditions imply both good size and good slack sensitivity. We define $\vec{g}$ for all $\vec{s} \in \mathbb{R}^m_{>0}$ by $\vec{g}(\vec{s}) \stackrel{\text{def}}{=} \arg\min_{\vec{w}\in\mathbb{R}^m_{>0}} \hat{f}(\vec{s},\vec{w})$ where

$$\hat{f}(\vec{s},\vec{w}) \stackrel{\text{def}}{=} \vec{\mathbb{1}}^T\vec{w} - \frac{1}{\alpha}\log\det(\mathbf{A}_s^T\mathbf{W}^{\alpha}\mathbf{A}_s) - \beta\sum_{i\in[m]}\log w_i \quad (6)$$

where $\mathbf{A}_s \stackrel{\text{def}}{=} \mathbf{S}^{-1}\mathbf{A}$ and the constants α, β are chosen later.

To get a sense for why $\vec{g}$ has the desired properties, suppose for illustration purposes that $\alpha = 1$ and $\beta = 0$. Setting the gradient of (6) to $\vec{0}$, we see that if $\vec{g}$ exists then

$$\vec{g}_i(\vec{s}) = (\mathbf{P}_{\mathbf{A}_s}(\vec{g}))_{ii}$$

for all i and consequently $\max_i \left\|\mathbf{G}^{-1/2}\vec{\mathbb{1}}_i\right\|_{\mathbf{P}_{\mathbf{A}_s}(\vec{g})} = 1$ and $\gamma(\vec{s},\vec{g}(\vec{s})) = 1$. Furthermore, since $\mathbf{P}_{\mathbf{A}_s}$ is an orthogonal projection, $\left\|\vec{g}(\vec{s})\right\|_1 = \text{rank}(\mathbf{A})$ and hence we see that this would yield a weight function with good c_γ and c_1.

Unfortunately picking $\alpha = 1$ and $\beta = 0$ makes the optimization problem for computing $\vec{g}$ degenerate. In the following theorem, we see that by picking better values for α and β, we can a desirable weight function.

Theorem 1. (Properties of Weight Function) Picking $\alpha = 1 - \log_2\left(\frac{2m}{\text{rank}(\mathbf{A})}\right)^{-1}$ and $\beta = \frac{\text{rank}(\mathbf{A})}{2m}$, $\vec{g}$ is a weight function meeting the criterion of Definition 4 with

- Size : $c_1(\vec{g}) = 2\,\text{rank}(\mathbf{A})$.
- Slack Sensitivity: $c_\gamma(\vec{g}) = 2$.
- Step Consistency : $c_r(\vec{g}) = 2\log_2\left(\frac{2m}{\text{rank}(\mathbf{A})}\right)$.

B. Geometric Interpretation of the Barrier

The new weight function is closely related to fundamental problems in convex geometry. If we set $\alpha = 1$, $\beta = 0$, the optimization problem appears in $\vec{g}$ is often referred to as *D-optimal design* and is directly related to computing the John Ellipsoid of the polytope $\{\vec{y} \in \mathbb{R}^n : |[\mathbf{A}(\vec{y} - \vec{x})]_i| \le s(\vec{x})_i\}$ [18]. Hence, one can view our linear programming algorithm as using John Ellipsoids to improve the convergence rate of interior point methods.

Our algorithm is not the first instance of using John Ellipsoids in optimization. In a seminal work of Tarasov, Khachiyan and Erlikh [39], they showed that a general convex problem can be solved in $O(n)$ steps of computing John Ellipsoid and querying a separating hyperplane oracle. Furthermore, Nesterov [28] also demonstrated how to use a John ellipsoid to compute approximation solutions for certain linear programs in $\tilde{O}(n^2m + n^{1.5}m/\epsilon)$ time.

From this geometric perspective, there are two major contributions of this paper. First we show that the logarithmic volume of an approximate John Ellipsoid is an almost optimal barrier function for linear programming and second that computing an approximate John Ellipsoids can be streamlined such that the cost of these operations is comparable to computing the standard logarithmic barrier function. Beyond obtaining a faster linear program solver, this implies that we obtain the fastest algorithm for computing approximate John Ellipsoids in certain regimes. We hope to further elaborate on this connection and show more applications in a followup paper.

C. Computing and Correcting The Weights

We apply the gradient descent method in a carefully scaled space to minimize $\hat{f}(\vec{s},\vec{w})$, it allows us to compute $\vec{g}(\vec{s})$ to high accuracy in $\tilde{O}(1)$ iterations if we have a good estimate of $\vec{g}(\vec{s})$ and can compute the gradient of $\hat{f}$ exactly. The first assumption is not an issue because $\vec{g}$ does not change too much between calls to compute $\vec{g}$. Unfortunately, it is expensive to compute the gradient of $\hat{f}$ exactly. The gradient of $\hat{f}$ is a function of leverage scores $\vec{\sigma}_{\mathbf{A}_s}$. In [37], they showed that we can compute $\vec{\sigma}_{\mathbf{A}_s}$ approximately by solving only polylogarithmically many regression problems (See [26] for more details). These results use the fact that the leverage scores of the the i^{th} constraint, i.e. $[\vec{\sigma}_{\mathbf{A}_s}]_i$ is the ℓ_2 length of vector $\mathbf{P}_{\mathbf{A}}(\vec{x})\vec{\mathbb{1}}_i$ and that by the Johnson-Lindenstrauss Lemma these lengths are persevered up to multiplicative error if we project these vectors onto certain random low dimensional subspace. Consequently, to approximate the $\vec{\sigma}_{\mathbf{A}_s}$ we first compute the projected vectors and then use it to approximate $\vec{\sigma}_{\mathbf{A}_s}$. Using this idea, we prove that a variant of gradient descent method still efficiently computes $\vec{g}$ with adequate error guarantees.

Theorem 2. (Weight Correction) Given a $\vec{s} \in \mathbb{R}^m_{>0}$, a $\vec{w} \in \mathbb{R}^m_{>0}$ such that $\left\|\mathbf{W}^{-1}(\vec{g}(\vec{s}) - \vec{w})\right\|_\infty \le \frac{1}{12c_r}$, and $K \in (0,1)$. We can find a $\vec{w}^{(\text{new})}$ such that

$$\left\|\mathbf{G}(\vec{s})^{-1}(\vec{g}(\vec{s}) - \vec{w}^{(\text{new})})\right\|_\infty \le K$$

with probability $1 - \frac{\tilde{O}(1)}{m}$. The running time is dominated by the time needed to solve $\tilde{O}(c_r^3\log(1/K)/K^2)$ linear systems.

Finally, we show how to compute an initial weight by first computing $\vec{g}$ for a large value of β and then decreasing β gradually.

Theorem 3. (Weight Computation) For $\vec{s} \in \mathbb{R}^m_{>0}$ and $K > 0$, with constant probability we can find a $\vec{w}^{(\text{new})}$ such that

$$\left\|\mathbf{G}(\vec{s})^{-1}(\vec{g}(\vec{s}) - \vec{w})\right\|_\infty \leq K.$$

The running time is dominated by the time needed to solve $\tilde{O}(\sqrt{\text{rank}\,(\mathbf{A})}\log(1/K)/K^2)$ linear systems.

VI. Approximate Weights Suffice

In the previous sections, we analyzed a weighted path following strategy assuming oracle access to a weight function we could compute exactly. However, to achieve the fastest running time for weighted path following presented in this paper, we need to show that it suffices to compute multiplicative approximations to the weight function.

This is a non-trivial statement as the weight function serves several roles in our weighted path following scheme. First, it ensures a good ratio between total weight c_1 and slack sensitivity c_γ. This allows us to take make large increases to the path parameter t after which we can improve centrality. Second, the weight function is consistent and does not differ too much from the c_r-update step direction. This allows us to change the weights between c_r-update steps without moving too far away from the central path. Given a multiplicative approximation to the weight function, the first property is preserved up to an approximation constant however this second property is not.

To effectively use multiplicative approximations to the weight function we cannot simply use the weight function directly. Rather we need to smooth out changes to the weights by using some slowly changing approximation to the weight function. In this section we show how this can be achieved in general.

A. The Chasing 0 Game

The *chasing 0 game* is as follows. There is player, an adversary, and a point $\vec{x} \in \mathbb{R}^m$. The goal of the player is to keep the point close to $\vec{0}$ in ℓ_∞ norm and the goal of the adversary is to move $\vec{x}$ away from $\vec{0} \in \mathbb{R}^m$. The game proceeds for an infinite number of iterations where in each iteration the adversary moves the current point $\vec{x}^{(k)} \in \mathbb{R}^m$ to some new point $\vec{y}^{(k)} \in \mathbb{R}^m$ and the player needs to respond. The player does not know $\vec{x}^{(k)}$, $\vec{y}^{(k)}$, or the move of the adversary. All the player knows is that the adversary moved the point within some convex set $U^{(k)}$ and the player knows some $\vec{z}^{(k)} \in \mathbb{R}^n$ that is close to $\vec{y}^{(k)}$ in ℓ_∞ norm. With this information the player is allowed to move the point a little more than the adversary. Formally, the player is allowed to set the next point to $\vec{x}^{(k+1)} \in \mathbb{R}^m$ such that $\vec{\Delta}^{(k)} \stackrel{\text{def}}{=} \vec{x}^{(k+1)} - \vec{y}^{(k)} \in (1+\epsilon)U$ for some fixed $\epsilon > 0$.

The question we would like to address is, how close the player can keep $\vec{x}^{(k+1)}$ to $\vec{0}$ in ℓ_∞ norm? We show that assuming that the $U^{(k)}$ are sufficiently bounded then there is strategy that the player can follow to ensure that that $\left\|\vec{x}^{(k)}\right\|_\infty$ is never too large. Our strategy simply consists of taking "gradient steps" using the following potential function

$$\Phi_\mu(\vec{x}) \stackrel{\text{def}}{=} \sum_{i\in[m]} p_\mu(x_i) \text{ where } p_\mu(x) \stackrel{\text{def}}{=} e^{\mu x} + e^{-\mu x}.$$

In other words, for all k we simply set $\vec{\Delta}^{(k)}$ to be the vector in $(1+\epsilon)U^{(k)}$ that minimizes the potential function Φ_μ for an appropriate choice of μ. In the following theorem we show that this suffices to keep $\Phi_\mu(\vec{x}^{(k)})$ small.

Theorem 4. Suppose that each $U^{(k)}$ is a symmetric convex set that contains an ℓ_∞ ball of radius r_k and is contained in a ℓ_∞ ball of radius $R_k \leq R$. Let $0 < \epsilon < \frac{1}{5}$ and consider the strategy

$$\vec{\Delta}^{(k)} = (1+\epsilon)\arg\min_{\vec{\Delta}\in U^{(k)}} \left\langle \nabla\Phi_\mu(\vec{z}^{(k)}), \vec{\Delta}\right\rangle \text{ where } \mu = \frac{\epsilon}{12R}.$$

Let $\tau \stackrel{\text{def}}{=} \max_k \frac{R_k}{r_k}$ and suppose $\Phi_\mu(\vec{x}^{(0)}) \leq \frac{8m\tau}{\epsilon}$ then $\Phi_\mu(\vec{x}^{(k)}) \leq \frac{8m\tau}{\epsilon}$ for all k. In particular, we have $\left\|\vec{x}^{(k)}\right\|_\infty \leq \frac{12R}{\epsilon}\log\left(\frac{8m\tau}{\epsilon}\right)$ for all k.

B. Centering Step With Noisy Weight

Now, we show how to use Theorem 4 to improve the centrality of $\vec{x}$ while maintaining the invariant that $\vec{w}$ is close to $\vec{g}(\vec{x})$ multiplicatively. Given a feasible point $(\vec{x}, \vec{w})$, we measure the distance between the current weights $\vec{w}$, and the weight function $\vec{g}(\vec{s})$, in log scale $\vec{\Psi}(\vec{s}, \vec{w}) \stackrel{\text{def}}{=} \log(\vec{g}(\vec{s})) - \log(\vec{w})$. Our goal is to keep $\left\|\vec{\Psi}(\vec{s}, \vec{w})\right\|_\infty \leq K$ for some error threshold K. We choose K to be just small enough that we can still decrease $\delta_t(\vec{x}, \vec{w})$ linearly and approximate $\vec{g}(\vec{s})$. Furthermore, we ensure that $\vec{\Psi}$ doesn't change too much in either $\left\|\cdot\right\|_\infty$ or $\left\|\cdot\right\|_{\mathbf{W}}$ and thereby ensure that the centrality does not increase too much as we move $\vec{w}$ towards $\vec{g}(\vec{s})$.

We meet these goals by playing the chasing 0 game where the vector we wish to keep near $\vec{0}$ is $\vec{\Psi}(\vec{s}, \vec{w})$, the adversaries moves are c_r-steps, and our moves change $\log(\vec{w})$. The c_r-step decreases δ_t and since we are playing the chasing 0 game we keep $\vec{\Psi}(\vec{s}, \vec{w})$ small. Finally, since by the rules of the chasing 0 game we do not move $\vec{w}$ much more than $\vec{g}(\vec{s})$ has moved, we have by similar reasoning to the exact weight computation case, Theorem 5, that changing $\vec{w}$ does not increase δ_t too much. This *inexact centering* operation and the analysis are formally defined and analyzed below.

Theorem 5 (Centering with Inexact Weights)**.** Given a point $(\vec{x}, \vec{w})$, an error parameter $K \leq \frac{1}{8c_r}$. Assume that

$$\delta_t(\vec{x}, \vec{w}) \leq \frac{K}{240c_\gamma c_r^2 \log\left(26c_r c_\gamma m\right)}$$

and $\Phi_\mu(\vec{\Psi}(\vec{x}, \vec{w}))$ is small enough. Then, we can find a new point $(\vec{x}^{(\text{new})}, \vec{w}^{(\text{new})})$ such that

$$\delta_t(\vec{x}^{(\text{new})}, \vec{w}^{(\text{new})}) \leq \left(1 - \frac{0.5}{1+c_r}\right)\delta_t(\vec{x}, \vec{w})$$

and

$$\Phi_\mu(\vec{\Psi}(\vec{x}^{(\text{new})}, \vec{w}^{(\text{new})})) \leq \Phi_\mu(\vec{\Psi}(\vec{x}, \vec{w})).$$

Also, we have $\left\| \log(\vec{g}(\vec{s}^{(\text{new})})) - \log(\vec{w}^{(\text{new})}) \right\|_\infty \leq K$.

VII. Running Times for Solving Linear Programs

Combining Theorem 5, Theorem 1 and Lemma 1 , we see that we can increase t by a factor of $1 + \tilde{\Omega}\left(\frac{1}{\sqrt{\text{rank}\,\mathbf{A}}}\right)$ and maintain centrality with $\tilde{O}(1)$ linear system solves. In the full version, we explain how to find an initial central point and how to round the central point to a solution for large enough t efficiently. Hence, the total running time of the algorithm now depends on how fast we can solve the linear systems involved. Using the fast matrix multiplication and recent results on solving overdetermined systems, we obtain the following result.

Theorem 6. Consider a linear program of the form (1) where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\vec{b} \in \mathbb{R}^m$, and $\vec{c} \in \mathbb{R}^n$ have integer coefficients. Let L be the bit complexity of (1). Suppose for any positive definite diagonal matrix $\mathbf{D} \in \mathbb{R}^{m \times m}$ with condition number $2^{\tilde{O}(L)}$, we can find a $\mathbf{B}$ such that

$$\left\|\mathbf{B} - (\mathbf{DA})^+ \vec{b}\right\|_{\mathbf{A}^T\mathbf{D}^2\mathbf{A}} \leq \epsilon \left\|(\mathbf{DA})^+ \vec{b}\right\|_{\mathbf{A}^T\mathbf{D}^2\mathbf{A}}$$

in time $O\left(\mathcal{T}\log(1/\epsilon)\right)$ for any $\epsilon > 0$ with probability at least $1 - \frac{1}{m}$. Then, there is an algorithm to solve the linear program in expected time $\tilde{O}\left(\sqrt{\text{rank}(\mathbf{A})}\left(\mathcal{T} + \text{nnz}(\mathbf{A})\right)L\right)$, i.e, find the active constraints of an optimum or prove that the program is unfeasible or unbounded.

Using [27], [22] to solve the linear systems, we obtain an algorithm that solves (1) in time

$$\tilde{O}\left(\sqrt{\text{rank}(\mathbf{A})}\left(\text{nnz}(\mathbf{A}) + (\text{rank}(\mathbf{A}))^\omega\right)L\right).$$

where $\omega < 2.3729$ [47] is the matrix multiplication constant.

Since all of our operations in each iteration are parallelizable, we achieve the first $\tilde{O}(\sqrt{\text{rank}\,(\mathbf{A})}L)$ depth polynomial work method for solving linear programs.

Theorem 7. There is an $\tilde{O}(\sqrt{\text{rank}\,(\mathbf{A})}L)$ depth polynomial work algorithm to solve linear program of the form (1).

Finally, we adapt techniques of Vaidya [43] to decrease the iteration costs of our method. This is based on the observation that the linear systems involved do not change too much from iteration to iteration and therefore this sequence of the linear system can be solved faster than considering them individually.

Theorem 8. There is an $\tilde{O}\left(n^{5/2-3(\omega-2)}m^{3(\omega-2)}L\right)$ time algorithm to solve linear program of the form (1).

For $\omega = 2.3729$, we have $\tilde{O}\left(m^{1.1187}n^{1.3813}L\right)$ which is strictly better than the previously fastest linear programming algorithm for dense matrix $\tilde{O}\left(m^{1.5}nL\right)$ in [43]. To appreciate this running time, we note that it takes time $O\left(mn^{1.3729}\right)$ to exactly solve a dense linear system of size $m \times n$, which is a necessary step to compute the solution exactly. However, we want to emphasis that linear solvers with low (amortized) cost can be designed for specific class of linear programs, such as the maximum flow problem. Therefore, running times better than $\tilde{O}\left(m^{1.1187}n^{1.3813}L\right)$ can be obtained for specific situations.

VIII. Maximum Flow Problem and More

In this section, we provide a brief overview of how we generalize our results and solve the various flow problems. Given $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\vec{b} \in \mathbb{R}^n$, $\vec{c} \in \mathbb{R}^m$, $\vec{l} \in \mathbb{R}^m$, $\vec{u} \in \mathbb{R}^m$, we consider the linear program

$$\begin{aligned} \min \quad & \vec{c}^T\vec{x}. \\ \vec{x} \in \mathbb{R}^m \ &: \ \mathbf{A}^T\vec{x} = \vec{b} \\ \forall i \in [m] \ &: \ l_i \leq x_i \leq u_i \end{aligned} \tag{7}$$

Without the upper constraints for x_i, the dual of this problem has only n variables and hence results from previous sections are applicable. Unfortunately, it is not clear how to apply our previous results in this setting and naive attempts to write (7) in the form of (1) without increasing $\text{rank}(\mathbf{A})$ fail. In particular, the maximum flow problem has rank $|V|$ if written in the form of (7) but we only know how to write this problem with rank $\Omega(|E|)$ in the form of (1) where $|V|$ is the number of vertices and $|E|$ is the number of edges.

A. The Difficulty

The analysis in previous sections rely on the fact that certain ellipsoid around the current point approximates the polytope well and therefore, so long as our Newton step maintain such an ellipsoid, we converge quickly. However, we cannot repeat such ℓ^2 analysis here. For example, when $\mathbf{A}$ is an empty matrix, we face the failure of the sphere to approximate a box and hence only able to obtain an $\Omega(\sqrt{m})$ iterations algorithm if each move is inside a certain ellipsoid inside the polytope. Hence, it is not obvious how such ℓ^2 type analysis can lead to an algorithm which converges in $\tilde{O}(1)$ iterations for $\mathbf{A}$ is an empty matrix.

Even more troubling, achieving a faster than $\tilde{O}(\sqrt{m}L)$ iterations interior point method for solving general linear programs in this form would break a long-standing barrier for the convergence rate of interior point methods. In a seminal result of Nesterov and Nemirovski [30], they provided a unifying theory for interior point methods and showed that given the ability to construct a *ν-self concordant barrier* for a convex set one can minimize linear functions over that convex set with a convergence rate of $O(\sqrt{\nu})$. To the best of our knowledge, there is no general purpose interior point method that achieves a convergence rate faster than the self concordance of the best barrier of the feasible region. Furthermore, using lower bounds results of Nesterov and Nemirovski [30, Proposition 2.3.6], it is not hard to see that any general barrier for (7) must have self concordance $\Omega(m)$.

B. Our Solution

We achieve our running time by a novel extension of the ideas in previous sections to work with the linear program (7) directly. Using an idea from [9], we create a 1-self concordant barrier for each of the $l_i \leq x \leq u_i$ constraints and run a primal path following algorithm with the sum of these barriers. While this would naively yield a $O(\sqrt{m}L)$ iteration method, we show how to use weights in a similar manner as in previous sections to improve the convergence rate to $\tilde{O}(\sqrt{\text{rank}(\mathbf{A})}L)$.

Now, when we apply the standard primal path following method to this problem the steps involve projecting the Newton step direction onto the kernel of $\mathbf{A}^T$. As before we want to re-weight the constraints (which here is the same as re-weighting $\mathbb{R}^m$) with weights such that $\left\|\vec{w}\right\|_1 \approx \text{rank}(\mathbf{A})$. However, if we only use the standard notion of centrality it may be the case that although the size of the projected Newton step is small in weighted ℓ_2 it may be large in weighted ℓ_∞ and thus we would be forced to take small steps). Our early argument that we can trivially bound the change in slacks from centrality and slack sensitivity simply breaks. To overcome this issue we simply explicitly keep track of the size of the Newton step in the appropriate ℓ_∞ norm. We define a mixed norm of the form $\left\|\cdot\right\| = \left\|\cdot\right\|_\infty + C_{\text{norm}}\left\|\cdot\right\|_{\mathbf{W}}$ for suitable constant C_{norm} and perform all analysis in this norm, extending many of our earlier arguments. Our reasoning about ℓ_∞ directly helps explain why our method outperforms the self-concordance of the best barrier for the space (which is a primarily ℓ_2 focused analysis).

Using this mixed norm idea and results about self concordant barriers, we extend results in previous sections to (7).

Theorem 6. *Suppose we have an interior point $\vec{x} \in S^0$ for the linear program (7). We can find an $\vec{x}$ such that $\vec{c}^T\vec{x} \leq OPT + \epsilon$ in $\tilde{O}\left(\sqrt{\text{rank}(\mathbf{A})}\log\left(U/\epsilon\right)\left(\mathcal{T}+m\right)\right)$ time where $\mathcal{T}$ is the time needed to solve certain linear systems related to $\mathbf{A}$ to enough accuracy and U is related to the size of the variables involved.*

This is the first general interior point method we aware of that converges at a faster rate than the self concordance of the best barrier of the feasible region.

We apply Theorem 6 to the maximum flow problem and the minimum cost flow problem. To do this, we use the reduction by Daitch and Spielman [5]. They showed how to write the generalized minimum cost flow problem into a linear problem in the form above with an explicit interior point. Furthermore, they showed that the linear systems occurs in interior point methods for solving certain flow problems can be reduced to Laplacian systems. Using their reduction, a recent nearly linear work polylogarithmic depth Laplacian system solver of Spielman and Peng [34] and our new interior point method, we obtain the promised generalized minimum cost flow algorithm.

Theorem 7. *There is a randomized algorithm to compute an approximate generalized minimum cost maximum flow in $\tilde{O}(\sqrt{|V|}\log^2(U/\epsilon))$ depth $\tilde{O}(|E|\sqrt{|V|}\log^2(U/\epsilon))$ work. Furthermore, there is an algorithm to compute an exact standard minimum cost maximum flow in $\tilde{O}\left(\sqrt{|V|}\log^2(U)\right)$ depth and $\tilde{O}(|E|\sqrt{|V|}\log^2(U))$ work.*

IX. Acknowledgments

We thank Yan Kit Chim, Andreea Gane, Jonathan A. Kelner, Lap Chi Lau, Aleksander Mądry, Cameron Musco, Christopher Musco, Lorenzo Orecchia, Ka Yu Tam and Nisheeth Vishnoi for many helpful conversations. This work was partially supported by NSF awards 0843915 and 1111109, NSF Graduate Research Fellowship (grant no. 1122374) and Hong Kong RGC grant 2150701. Finally, we thank the referees for extraordinary efforts and many helpful suggestions.

References

[1] Kurt M Anstreicher. Potential reduction algorithms. In *Interior point methods of mathematical programming*, pages 125–158. Springer, 1996.

[2] Kurt M. Anstreicher. Volumetric path following algorithms for linear programming. *Math. Program.*, 76:245–263, 1996.

[3] András A Benczúr and David R Karger. Approximating st minimum cuts in õ (n 2) time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 47–55. ACM, 1996.

[4] Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 273–282. ACM, 2011.

[5] Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 451–460. ACM, 2008.

[6] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[7] Shimon Even and R Endre Tarjan. Network flow and testing graph connectivity. *SIAM journal on computing*, 4(4):507–518, 1975.

[8] Robert M Freund. Projective transformations for interior point methods, part ii: Analysis of an algorithm for finding the weighted center of a polyhedral system. 1988.

[9] Robert M Freund and Michael J Todd. Barrier functions and interior-point algorithms for linear programming with zero-, one-, or two-sided bounds on the variables. *Mathematics of Operations Research*, 20(2):415–440, 1995.

[10] Zvi Galil and Éva Tardos. An o (n 2 (m+ n log n) log n) min-cost flow algorithm. *Journal of the ACM (JACM)*, 35(2):374–386, 1988.

[11] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[12] Andrew V Goldberg and Robert E Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[13] Clovis C Gonzaga. Path-following methods for linear programming. *SIAM review*, 34(2):167–224, 1992.

[14] David Karger and Matthew Levine. Random sampling in residual graphs. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 63–66. ACM, 2002.

[15] David R Karger. Better random sampling algorithms for flows in undirected graphs. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 490–499. Society for Industrial and Applied Mathematics, 1998.

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

[17] Jonathan A Kelner, Lorenzo Orecchia, Yin Tat Lee, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. *arXiv preprint arXiv:1304.2338*, 2013.

[18] Leonid G Khachiyan. Rounding of polytopes in the real number model of computation. *Mathematics of Operations Research*, 21(2):307–320, 1996.

[19] Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *The 45th ACM Symposium on Theory of Computing (STOC)*, pages 755–764, 2013.

[20] Yin Tat Lee and Aaron Sidford. Path-finding i: Solving linear programs with õ (sqrt (rank)) linear system solves. *arXiv preprint arXiv:1312.6676*, 2013.

[21] Yin Tat Lee and Aaron Sidford. Path-finding ii : An õ (m sqrt (n)) algorithm for the minimum cost flow problem. *arXiv preprint arXiv:1312.6713*, 2013.

[22] Mu Li, Gary L Miller, and Richard Peng. Iterative row sampling. 2012.

[23] László Lovász and Santosh Vempala. Simulated annealing in convex bodies and an $o^*(n^4)$ volume algorithm. *J. Comput. Syst. Sci.*, 72(2):392–417, 2006.

[24] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *FOCS*, pages 245–254, 2010.

[25] Aleksander Madry. Navigating central path with electrical flows: from flows to matchings, and back. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science*, 2013.

[26] Michael W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3(2):123–224, 2011.

[27] Jelani Nelson and Huy L Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. *arXiv preprint arXiv:1211.1002*, 2012.

[28] Yu. Nesterov. Rounding of convex sets and efficient gradient methods for linear programming problems. *Optimization Methods Software*, 23(1):109–128, February 2008.

[29] Yu E Nesterov and Michael J Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations research*, 22(1):1–42, 1997.

[30] Yurii Nesterov and Arkadii Semenovich Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13. Society for Industrial and Applied Mathematics, 1994.

[31] James B Orlin. Genuinely polynominal simplex and non-simplex algorithms for the minimum cost flow problem. 1984.

[32] James B Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations research*, 41(2):338–350, 1993.

[33] James B Orlin. Max flows in o (nm) time, or better. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 765–774. ACM, 2013.

[34] Richard Peng and Daniel A Spielman. An efficient parallel solver for sdd linear systems. *arXiv preprint arXiv:1311.3286*, 2013.

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

[36] Jonah Sherman. Nearly maximum flows in nearly linear time. In *Proceedings of the 54th Annual Symposium on Foundations of Computer Science*, 2013.

[37] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *STOC*, pages 563–568, 2008.

[38] Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2004.

[39] SP Tarasov, LG Khachiyan, and I Erlikh. The method of inscribed ellipsoids. In *Soviet Mathematics Doklady*, volume 37, pages 226–230, 1988.

[40] Éva Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

[41] Pravin M. Vaidya. An algorithm for linear programming which requires o($((m+n)n^2+(m+n)^{1.5}n)$l) arithmetic operations. In *STOC*, pages 29–38, 1987.

[42] Pravin M. Vaidya. A new algorithm for minimizing convex functions over convex sets (extended abstract). In *FOCS*, pages 338–343, 1989.

[43] Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 332–337. IEEE, 1989.

[44] Pravin M. Vaidya. Reducing the parallel complexity of certain linear programming problems (extended abstract). In *FOCS*, pages 583–589, 1990.

[45] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. *Mathematical Programming*, 73(3):291–341, 1996.

[46] Pravin M Vaidya and David S Atkinson. A technique for bounding the number of iterations in path following algorithms. *Complexity in Numerical Optimization*, pages 462–489, 1993.

[47] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898, 2012.